Constructing Semantically Sound Object-Logics for UML/OCL Based Domain-Specific Languages

Frédéric Tuong

université PARIS-SACLAY

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir®

NNT: 2016SACLS085

THÈSE DE DOCTORAT
DE
L'UNIVERSITÉ PARIS-SACLAY
PRÉPARÉE À
L'UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE N° 580
Sciences et Technologies de l'Information et de la Communication

Spécialité de Doctorat : Informatique

Par

M. Frédéric TUONG

Constructing Semantically Sound Object-Logics for UML/OCL Based Domain-Specific Languages

Thèse présentée et soutenue à Orsay, le 6 avril 2016 :

Composition du jury :

M. Stéphane MAAG	Professeur, Télécom SudParis	Président
Mme Catherine DUBOIS	Professeur, ENSIIE	Rapporteur
M. Bernhard RUMPE	Professeur, RWTH Aachen University	Rapporteur
M. Achim D. BRUCKER	Maître de Conférences, University of Sheffield	Examinateur
M. Safouan TAHA	Maître de Conférences, CentraleSupélec	Examinateur
M. Burkhart WOLFF	Professeur, LRI	Directeur de thèse

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Remerciements

@ Stéphane Maag, Catherine Dubois, Bernhard Rumpe, Achim D. Brucker, Safouan Taha, Burkhart Wolff

Je souhaiterais naturellement adresser à Burkhart Wolff, mon directeur de thèse mes sincères remerciements. Depuis le début de ce doctorat, ses conseils et ses encouragements m'ont permis de faire grandement évolué ces travaux de thèse, jusqu'à terminer sereinement ce manuscrit. Je suis très fier du résultat, à mon avis, il a réussi à me transmettre les bonnes bases pour bien continuer, en le félicitant, j'aimerais le remercier avant tout !

Merci à l'ensemble des membres du jury ! Merci à Catherine Dubois et Bernhard Rumpe mes deux rapporteurs de thèse qui ont attentivement examiné les travaux de cette thèse et donné leurs avis favorables pour la soutenance, merci pour les remarques et les corrections du manuscrit, ainsi que les échanges qu'on a eu plus directement, que ce soit avant la soutenance et après. Merci à Stéphane Maag, Achim D. Brucker et Safouan Taha pour m'avoir adressé par la suite de précieux conseils en vue de l'amélioration du manuscrit, et pour nos riches échanges ayant lieu lors des questions d'ouverture.

@ Romain Aïssat, Thibaut Balabonski, Sergio Bezzecchi, Etienne Borde, Frédéric Boulanger, Achim D. Brucker, Marina Egea, Abderrahmane Feliachi, Marie-Claude Gaudel, Martin Gogolla, Fateh Guenab, Antoine Jaouën, Chantal Keller, Ali Koudri, Zheng Li, Delphine Longuet, Yakoub Nemouchi, Huu Nghia Nguyen, Hai Nguyen Van, Laurent Pautet, Valentin Perrelle, Smail Rahmoun, Elie Soubiran, Safouan Taha, Benoît Valiron, Frédéric Voisin, Makarius Wenzel, Burkhart Wolff, Laurent Wouters, Lina Ye, Fatiha Zaïdi, ..., FSF, VALS

Merci à vous cher(e)s collègues de FSF et VALS sans exceptions, programmer et discuter à vos côtés au quotidien était fort agréable !

J'aimerais spécialement remercier Delphine Longuet pour nos échanges passés lors de ce doctorat, merci pour ses conseils lors de mes séances de répétitions d'exposés, ainsi que les diverses préparations d'articles me permettant par la suite d'éclaircir plus en détail certains points du manuscrit présent.

J'aimerais également remercier Achim D. Brucker pour la qualité technique des infrastructures informatiques mises à disposition, aussi bien les accès SVN que l'environnement Jenkins, et la base bibliographique LaTeX associée. Merci aussi pour son soutien et les discussions favorisant l'inspiration et le développement de ce travail.

Merci à l'Université Paris-Sud et à l'IRT SystemX pour la création de ce contrat doctoral, leur collaboration, ainsi qu'au programme Investissements d'Avenir supportant ces travaux de recherche.

@ ∞

Encore merci à vous ! Même si ce joker me permet maintenant d'obtenir un document de taille raisonnable, ce symbole reflète tout à fait l'étendu des remerciements que je tiens à vous adresser. S'il y a une boucle infinie à lancer en profondeur (en shallow-mode), c'est effectivement maintenant.

Je remercie la succession des différents événements finalement reliés dans un certain sens à ce doctorat, notamment l'accueil de Fabrice Le Fessant à Inria/OCamlPro, et aussi l'accueil de Frédéric Blanqui au sein du projet FORMES. Merci globalement à mes anciens collègues respectifs de ces équipes projets, à mes amis et aux personnes qui m'ont encouragé à poursuivre vers un doctorat, et forcément au-delà.

Avec une pensée particulière et chaleureuse pour mon beau-frère et ma nièce, je voudrais à présent remercier de la meilleure façon possible mes parents et mes deux sœurs en leur dédiant ce manuscrit.

Contents

Abstract

Object-based and object-oriented specification languages (like UML/OCL, JML, Spec#, or Eiffel) allow for the creation and destruction, casting and test for dynamic types of statically typed objects. On this basis, class invariants and operation contracts can be expressed; the latter represent the key elements of object-oriented specifications. A formal semantics of object-oriented data structures is complex: imprecise descriptions can often imply different interpretations in resulting tools.

In this thesis we demonstrate how to turn a modern proof environment into a *meta-tool* for definition and analysis of formal semantics of object-oriented specification languages. Given a representation of a particular language embedded in Isabelle/HOL, we build for this language an extended Isabelle environment by using a particular *method* of code generation, which actually involves several variants of code generation. The result supports the asynchronous editing, type-checking, and formal deduction activities, all "inherited" from Isabelle.

Following this method, we obtain an *object-oriented modelling tool* for textual UML/OCL. We also integrate certain idioms not necessarily present in UML/OCL— in other words, we develop support for domain-specific dialects of UML/OCL.

As a meta construction, we define a meta-model of a part of UML/OCL in HOL, a meta-model of a part of the Isabelle API in HOL, and a translation function between both in HOL. The meta-tool will then exploit two kinds of code generation to produce either *fairly efficient code*, or *fairly readable code*. Thus, this provides two animation modes to inspect in more detail the semantics of a language being embedded: by loading at a native speed its semantics, or just delay at another "meta"-level the previous experimentation for another type-checking time in Isabelle, be it for performance, testing or prototyping reasons.

Note that generating "fairly efficient code", and "fairly readable code" include the generation of *tactic code* that proves a collection of theorems forming an object-oriented datatype theory from a denotational model: given a UML/OCL class model, the proof of the relevant properties for casts, type-tests, constructors and selectors are automatically processed. This functionality is similar to the *datatype theory packages* in other provers of the HOL family, except that some motivations have conducted the present work to program high-level tactics in HOL itself.

This work takes into account the most recent developments of the UML/OCL 2.5 standard. Therefore, all UML/OCL types including the logic types distinguish two different exception elements: `invalid` (exception) and `null` (non-existing element). This has far-reaching consequences on both the logical and algebraic properties of object-oriented data structures resulting from class models.

Since our construction is reduced to a sequence of conservative theory extensions, the approach can guarantee logical soundness for the entire considered language, and provides a methodology to soundly extend domain-specific languages.

Keywords

Object-oriented Data Structures, Path Expressions, Featherweight OCL, Null, Invalid, Formal Semantics, Isabelle, Reflection, UML, OCL.

CHAPTER 1

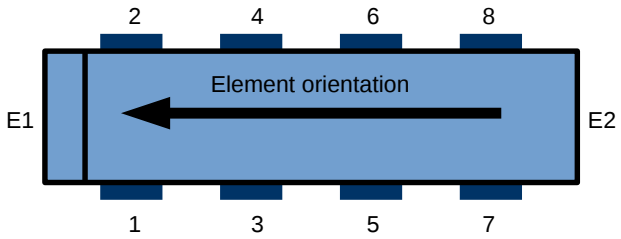
Introduction

Reproduction of goods and objects in assembly lines has lead to major turning points throughout centuries in mankind history. As a well-known wonder of the Anciant World, we only cite the Great Pyramid of Giza, still internally composed of an estimated two million of smaller blocks. The total assembling took decades of manual work. Following the first and second Industrial Revolutions, constructions by hand, be it collaborative, have become all the more assisted by machines to save labour and workforce. *Computers* are at the heart of recent major inventions, to assist mankind when conceiving *objects*, to appropriately control machines and automate other technical artifacts of our lives. With the gigantic amount of calculations that can be routinely handled in recent calculators, this resulted a modern form of industrial revolution: the "Information Age", as called by science historians.

Objects we daily encounter and manipulate have certain properties and characteristics that must precisely be taken into account in form of a *model* before they can be treated by a computer. Indeed, objects are the basis of much larger concepts in many intellectual domains: ranging from abstract objects to concrete objects, biological animated objects, physical objects in experimental sciences, or as opposed to subjects in philosophy, if not mentioning objects of desire. Because building a car does not require the same chemical components as that of a train, models include as fundamental information the characteristic of manipulated objects, called *attributes*, as well as the *operations* or set of actions that objects are supposed to support or not.

The sub-discipline of informatics treating the design, the programming and the analysis of computer systems *interacting* with objects of the real world is called *embedded systems*. It is most relevant in domains where objects are embedding one or more (physical) computing objects. Examples are avionics, railway and automotive systems, medical devices like pace-makers, but can also be found in multifunction smartphone technology.

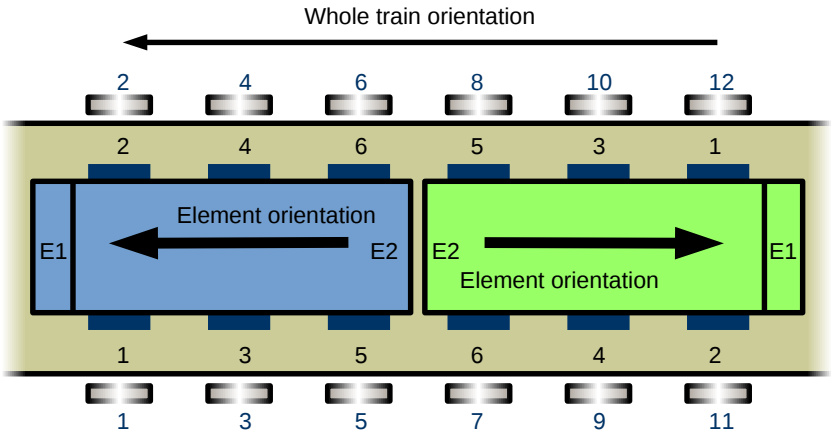
As example, consider the excerpt Figure 1.1 drawn from a system specification, by courtesy of Alstom, that we heavily abstracted for the purpose of this presentation. From this quite exemplary *system design document* used in industrial practice, one can draw for the corresponding model the following information: there are *trains* (with the *orientation* as attribute) containing *doors* (with an *index number*, besides the possibility to be *opened* or *closed*).



[...] *Train doors indexing for a single unit train:*
The convention for RSD indexing on a single unit train is the following: In the single train element reference (oriented in the direction $E2 \rightarrow E1$):

- Side A (left) has odd indexes,
- Side B (right) has even indexes,
- Index N°1 corresponds to the first doors (closest to the element front extremity $E1$) on the left,
- Index N°2 corresponds to the first doors (closest to the element front extremity $E1$) on the right.

Figure 1.1: Train element



[...] *Train doors indexing for a multiple unit train:*
The convention for RSD indexing on a multiple unit train is the same as for a single unit train, but for the reference which is the whole train, with the orientation of the train performing the process.

Figure 1.2: A train in a platform

Later on, the system design document of Figure 1.2 continues to describe the geometry of a train located in a *platform* of a railway station, and under which condition both match (which is an *operation* in the sense above). Thus, security critical operations like "open_doors" can be modelled, analysed and implemented in the train control system.

From the above said, one can conclude two observations:

1. It would be hopeless and useless to represent *all* properties of objects in a corresponding model that a computer can handle: in the railway network, if the goal is to estimate the number of running trains, including only locomotives and magnetic trains, then trains running on different networks or with other characteristics have to be ignored, so the color, number of wheels, and weight are for instance irrelevant. Thus, models are necessarily deliberately conceived as *abstractions* of the physical world, implying that in one system, one may actually have several abstractions of one physical object a system has to deal with.

2. The notion of object as modelling entity comes with the notion of a *class*. Objects belonging to a class have a number of attributes and characteristics in common, and this allows operations to work uniformly on them, i. e. in a *type safe* manner. (The notion of type and type-safeness will be substantially refined in the subsequent chapters.) Moreover, even classes have a number of attributes and characteristics in common: the class of *doors* could be divided into several *subclasses*: *manual doors*, *automatic doors*, *emergency doors*, etc... It is desirable that this particular relationship between classes, called *inheritance*, is technically supported in models which are organized in this *object-oriented* way.

Object-oriented modelling has seen its birth in the late sixties in the context of *programming languages* for computers, i. e. specific formal languages that are suited to be processed and executed by computers. The language Simula [DN66] is usually seen as the ancestor of this development, which led over languages such as Smalltalk [Kay93] to the current mainstream languages Java [AGH00] and C++ [Str86] to recent offsprings such as Scala [Oa04] and Swift [App16].

The growing influence of these languages in informatics raised the need of languages, that are not necessarily executable on a computer. Rather, the emphasis is again on *modelling*: modelling embedded systems, as well as the possibility to analyse systems and programs before they are actually implemented. This way, languages can be used to analyse if critical operations (like "open_doors") can actually be described in an unambiguous way and does not lead to undesired consequences.

Describing critical operations in an unambiguous way is all the more easy if the language rejects the possibility to form absurd sentences, where for example critical operations and non-critical operations are considered in a sentence as equivalent. This is especially fundamental for modelling languages, which are evolving in many forms to appropriately capture the description of new phenomena, and to enounce problems in a more suitable context than another. The generalization goes to languages supporting mathematical shapes and geometry, languages used to assert properties on objects of the real world, and simply speaking human communication languages, whenever they are dealing with *logical* sentences.

To determine the truth of a logical sentence, one can exploit a particular class of software for this task: a *proving system* comes with some specialized utilities to perform logical reasoning, so to fundamentally prove theorems as in mathematics. Their ability to state theorems depends on a *small* core environment, or *logical framework*, which is small enough to be understood by logicians. Pragmatically, higher-order logic (HOL) is built on top of the small core, and constitutes one of the many variations of *object-logics* enabling to tackle modern problems in mathematics.

Logical frameworks are particularly suitable to be extended with new object-logics, depending on the domain-specific problem one is encountering during the modelling activity. Logically safe extensionality has been a key feature of interactive theorem proving (ITP) systems in the HOL family, which goes back to the influential LCF system in 1979 [GMW79]. This goal motivated key principles like correctness by construction for primitive inferences in a fairly small kernel, flexible programmability in userspace via ML protecting this kernel by its type discipline, and top-level command interaction allowing for the development of layers of commands over this kernel. The principle of extensionality is still maintained in ITP systems like Coq [BC04] and Isabelle [NPW09], which offer an own, more high-level command language interface such as Gallina (Coq) [Hue92] and Isar (Isabelle) [Wen02]. Extensionality leveraged the scalability of the definitional principles of the LCF approach, paving the way for specific support of specification constructs for, e. g., datatypes or recursive function definitions.

Support implementations for such constructs are called *packages*. (To our knowledge, the term was first used for a datatype package described in Thomas F. Melham's work [Mel91].) A package takes a piece of (abstract) syntax, for example the following datatype *command* defines natural numbers in Isabelle/HOL:

datatype $\quad \mathrm{Nat} = 0 \mid \mathrm{Suc}\,\mathrm{Nat}$

This datatype triggers the generation of a *datatype theory*, i. e., a collection of definitions and *logical rules*, which are HOL theorems. This datatype comprises the declaration of the type Nat, the constants 0 and the inductive closure of naturals formed with the successor Suc, as well as the rules $0 \neq \mathrm{Suc}\,x$ (distinctness), $\mathrm{Suc}\,y = \mathrm{Suc}\,x \Longrightarrow y = x$ (injectivity), induction etc., with one word: the Peano axioms. In our system of reference Isabelle/HOL, this datatype theory is automatically generated from the syntax above, together with a number of rules allowing for efficient code generation or automatic proof support. These datatype theories are in many systems like Spec# [BLS05], Dafny [KL12], ACSL [BCF+13], or JML [LPC+13] generated as a collection of declarations and axiomatisations of its rules; in contrast, Isabelle/HOL, following the tradition of LCF-like systems like HOL Light [Har14] and HOL4 [NS14], *derives* these rules by *defining* the constructors 0 and Suc as functions on a Lisp-like S-Expression universe [McC65], i. e., by giving the constructors a denotational semantics rich enough to serve as a model for the datatype theories.

Writing packages is a highly complex task which is mastered only by a handful of engineers behind the different HOL systems. In this thesis, however, we address the issues of building packages to support, by a series of packages, an entire *formal method* behind a domain specific language $\mathcal{L}$ (we will describe after), *beyond* the necessary prerequisite of getting its semantics right. We aim at building tools that provide a domain specific formal environment of development

for the embedded language, so that domain experts of $\mathcal{L}$ have to only acquaint a semantically sound subset of $\mathcal{L}$ to use the resulting tool. Simultaneously, other domain experts could be interested to practice automated reasoning on $\mathcal{L}$ by only programming with the theorem prover where the tool is relying on. To this end, we demonstrate what technical mechanisms and abstractions of the Isabelle framework can be combined to the construction of such multidisciplinary tool. The choice of the Isabelle framework is also due to its rapid evolution in the last ten years, which might influence other systems to use similar mechanisms (for example, the Paral-ITP Isabelle/Coq project[1] [BTT15]).

In this thesis, we instantiate the resulting formal method tool, to address in parallel a key problem of defining formal semantics for object-oriented programming languages and specification languages semantics, namely the representation of the underlying *object-oriented datatype theory*. By object-oriented datatype theory, we mean a set of rules related with the foundation of object-oriented languages, like class definitions as in the following Java code:

```
class A {

  char s;
  A(){
    ...
  }
}
```

```
class C1
  extends A {
  int a;
  C1() {
    ...
  }
}
```

```
class C2
  extends A {
  boolean b;
  C2() {
    ...
  }
}
```

These class definitions must be semantically represented in a "background theory" in systems like Spec#, JML, or Dafny. This formal theory reflecting the semantics of this code will comprise the type declaration `A`, `C1` and `C2`, definitions of constructor functions must be given (representing the effect of object creation as in `C1 c = new C1();`) as well as cast operations (such as `A a = (A) c`) that can change the *static type* of an object to make it acceptable to interfaces requiring an `A` object; these coercions are usually inserted by the compiler (as in `A d = c;`) but have to be declared and defined for a formal treatment in a verification environment. Moreover, there are operations that test the *dynamic type* of an object, i. e., the type under which it is dynamically created.[2] In Java, this test is written `d instanceof C1` which will thus yield true, while the static type of `d` is of course `A`. Together with the accessor (or destructor) functions to fields in objects like `d.s`, this results in a quite rich theory, with logical rules like: $(X$ `instanceof C1`$) \not\Longleftrightarrow (X$ `instanceof C2`$)$ or `((C1)(A)`Y`)` $= Y$, i. e., "an object Y cast up and down again semantically equals to itself." Here, Y is a free variable, for which a Milner style type-inference will infer the type `C1` since the logical equality $\_ = \_$ has type $\alpha \Rightarrow \alpha \Rightarrow$ bool. This "upcast downcast" property is vital in object-oriented datatype theories, e. g., for the implementation of generics in Java. In the following code:

```
ArrayList<A> elements = new ArrayList<>();
elements.add(c);
```

[1] `http://paral-itp.lri.fr/`

[2] In the Java documentation, the dynamic type is called "actual type" in contrast to the static type referred as "apparent type".

c is not only cast up to `A`, but to `Object` and casts back to `A` again during the access, so `elements.get(i)` has the static type `A`. It turns out that object-oriented datatype theories are amazingly complex for fairly small class systems.

1.1 Contributions

As a basis for this work, we demonstrate for a particular formal method how in the Isabelle framework a *formal method tool* can be constructed. We developed a machine checked semantics for a large fragment of the Object Constraint Language OCL [Obj12] in the interactive theorem prover Isabelle/HOL [NPW09]. The result, called *HOL-OCL 2.0* (which is a successor of HOL-OCL [BW08a, BDW06c, BW02a]), supports OCL specifications over UML class models using a textual notation. HOL-OCL 2.0 as a tool is based on a library defining its core semantic concepts called *Featherweight OCL* [BTW14], which also serves as basis for the ongoing OCL 2.5 standardisation at the OMG. Our formalisation already helped to find inconsistencies, e.g. in the semantics of the logical connectives, that are fixed in the last update of the standard. The opportunity to influence the standardisation of an object-oriented language that is widely used in industry is not the only motivation for choosing UML/OCL as basis for our work. We understand UML/OCL as a representative of a large family of object-oriented languages and, thus, our work provides a generic technique for formalising object-oriented languages as well as insights into properties of object-oriented systems in general. In particular, UML/OCL provides

1. a statically typed object model, offering a fairly "conventional" object-oriented datatype theory;
2. associations, two state interpretations of paths, and the distinction between strict and non-strict exceptional elements;
3. a compromise between an object-oriented specification and a programming language, that can be easily compiled to other members of the object-oriented language family.

Our tool HOL-OCL 2.0 addresses the fragment in UML concerned with object-oriented data modelling. HOL-OCL 2.0 comes with a number of specialized packages, for instance the *Class Model Package* to set up the underlying object-oriented datatype theory, or the *Invariant & Operation Package* supporting a formal contract language to define methods issued from a class model. Its design pursues several objectives, namely:

1. providing an environment for studying the semantics of the embedded language (e. g., UML/OCL); this formal semantics is currently part of an initiative to provide a new "Annex A" for the semantic definition of OCL 2.5 [BTW14],
2. providing an environment for proving properties over artefacts expressed in this domain specific language (are invariants consistent? are method contracts implementable?), and
3. providing an environment for animation, code- and test case generation for models expressed in UML/OCL.

To this end, we propose a new method to develop packages in an ITP system: instead of writing a package in the sole implementation language, the aforementioned objectives have suggested us to take even more advantage of the overall capacities of the ITP system. Our implementation comprises the development of the core packaging function in HOL, the use of a code generator to convert it to the meta-language of the ITP system, and the use of specific binding to command level syntax. Thus, the resulting tool reuses the infrastructure of the ITP platform, such as the asynchronous front-end Prover IDE, code and documentation generation facilities, and, last but not least, automated and interactive proof support. In more detail, we provide:

1. A model (or abstract syntax) of (a part of) the Isabelle API. This model has been published during this thesis [TW15] and can be potentially reused by developers of other packages.

2. A model (or abstract syntax) of (a part of) UML/OCL. A simplified version of this model has been partly published in the same document [TW15], together with a functional working example.

3. A "compiler" mapping UML/OCL class diagrams to Isabelle/HOL definitions and Isabelle/Isar proofs (this part is not yet published but the present thesis will give a more detailed overview of its content).

Thus, similar to conventional datatype packages, a component is built that derives the lemmas of an "object-oriented datatype theory" from a class model. Being the basis for more abstract proofs from the problem domain, they allow for formal code verification, refinement and test generation techniques that UML models usually lack.

As consequence, our work can be seen as a major case study for our technique to develop packages. From the work done, it can be safely concluded that fairly large and complex packages *can be implemented this way*, without neither a sensible penalty with respect to efficiency nor to loss of interactivity: the Prover IDE continuous build and continuous check workflow handles as usual (without interruptions) all proof activities in the background.

HOL-OCL 2.0: A Formal Method Tool for UML/OCL

We introduce UML/OCL by a small example of a class model together with its class invariants and a method contract in OCL. Figure 1.3 describes a set of clients owning bank accounts in different banks using a textual representation that we share with other tools such as USE [GBR07]. Each account is either a `Current` account or a `Savings` account, and belongs to exactly one bank and one client. Clients younger than 25 years are allowed to overdraft by 250 €. Moreover, the balance of a savings account must be between `0` and `max`.

First, to enable OCL users to use HOL-OCL 2.0 for analysing UML/OCL specifications, details of the actual embedding need to be hidden as many as possible behind a suitable user interface. We use the flexibility of Isabelle's Isar language [Wen02] as well as the extensionality of the Isabelle/jEdit Prover IDE to achieve this goal. Figure 1.5 shows the user interface of HOL-OCL 2.0 that is based on the Isabelle/jEdit Prover IDE. A domain expert can easily define (in the red frame of Figure 1.5) a UML/OCL model similar as Figure 1.4 using the

```
Class Bank
      Attributes name       : String

Class Client
      Attributes clientname: String
                 address    : String
                 age        : Integer

Class Account
      Attributes id         : Integer
                 balance    : Currency

Class Savings < Account
      Attributes max        : Currency

Context c: Savings
  Inv 0 < c.max

  Inv 0 <= c.balance
       and c.balance <= c.max
```

```
Association clients  Between
    Bank     [1 .. *] Role banks
    Client   [1 .. *] Role clients

Association bank     Between
    Account  [1 .. *] Role b_accounts
    Bank     [1]      Role bank

Association owner    Between
    Account  [1 .. *] Role c_accounts
    Client   [1]      Role owner

Class Current < Account
      Attributes overdraft : Currency

Context c: Current
  Inv 25 <= c.owner.age   implies
        c.overdraft = 250
  Inv 25 >  c.owner.age   implies
        c.overdraft = 0
```

```
Context Bank :: create(clientname:String, age:Integer)
  Pre  self.clients->forAll(c | c.clientname <> clientname or  c.age <> age)
  Post self.clients->exists(c | c.clientname =  clientname and c.age =  age)
```

Figure 1.3: A simple class model with OCL constraints capturing a bank account.

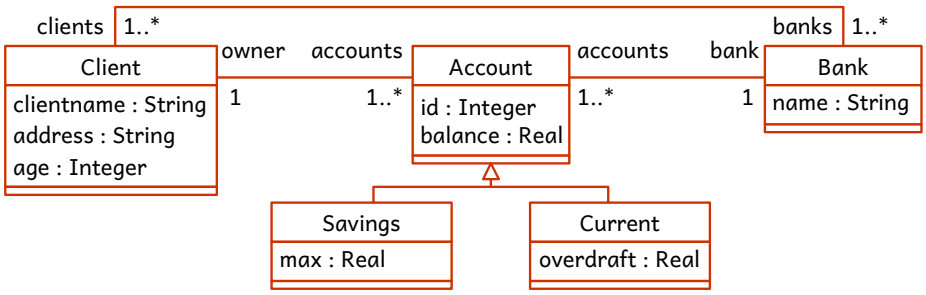


Figure 1.4: A simple class model capturing a bank account

textual notation from Figure 1.3, as well as use the standard Isar commands for theorems and proofs over this UML/OCL specification. Even the automatically UML/OCL level type information is accessible to the user by hovering over sub-expressions. Here, not only the encoded HOL types are shown, our implementation is able to show the actual OCL types which hides the complexity of the actual embedding from the users of HOL-OCL 2.0 . Clicking on operations inside OCL expressions allows for the navigation into their semantic definitions in the library.

Second, to enable a high-degree of automation as well as a user friendly syntax for defining UML/OCL models, instances of models, or proof obligation, we implement in HOL-OCL 2.0 the following packages:

- *Class Model Package* for declaring a UML data model, i. e., classes, associations, aggregations, enumerations.
- *Invariant & Operation Package* for declaring, in the context of an already

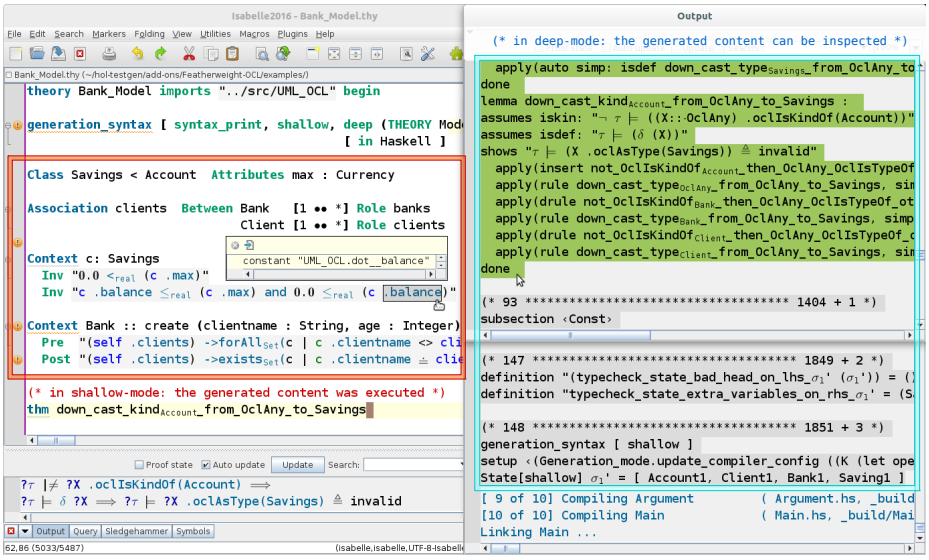


Figure 1.5: The HOL-OCL 2.0 system (user interface)

defined class model, OCL class invariants and operation contracts.

- *Instance Package* for declaring class instances, i. e., objects.
- *State Package* for grouping objects together in a common state.
- *Transition Package* for transition properties over a pair of pre- and post-state.

For example, after defining our exemplary data model using the Class Model Package (recall Figure 1.5), we can use the Instance command provided by the *Instance Package* for defining objects over this class model:[3]

$$\begin{array}{llll} \text{Instance} & \text{S1} & :: \texttt{Account} & = & ([\texttt{max} = 2000] :: \texttt{Savings}) \\ \text{and} & \text{C1} & :: \texttt{Client} & = & [\texttt{c\_accounts} = \text{S1}, \texttt{banks} = \text{B1}] \\ \text{and} & \text{A1} & :: \texttt{Account} & = & [\texttt{id} = 250, \texttt{owner} = \text{C1}] \\ \text{and} & \text{B1} & :: \texttt{Bank} & = & [\texttt{b\_accounts} = [\text{S1}, \text{A1}]] \end{array}$$

This command generates a set of definitions using the appropriate definitions in terms of the Featherweight OCL library:

$$\begin{array}{ll} \text{definition}\ \text{S1}_{\text{Account}} & = \text{mk}_{\text{Savings}}\ (\text{mk}\,\mathcal{EXT}_{\text{Savings}}\ \text{oid3}\ \text{None}\ \text{None})\ \lfloor 2000 \rfloor \\ \text{definition}\ \text{S1} & = ((\lambda\_.\lfloor\lfloor \text{S1}_{\text{Account}} \rfloor\rfloor) :: \cdot\texttt{Savings}).\texttt{oclAsType(Account)} \end{array}$$

Since the Instance command is tightly connected with the typing engine of Isabelle, it becomes possible to infer most of the OCL types without explicit type annotations. For the case of Instance, even the inference of multiplicities is fully automatic (and respect the bidirectional sense): after the type-checking stage,

[3] This command is not present as such in USE, instead it manipulates objects and instances with a special imperative language.

we does *not* have that "C1 `.c_accounts` is equal to `Set{S1}`", instead, we have correctly that it is equal to `Set{S1, A1}`, since C1 appears as an "owner" of A1.

Besides definitions, HOL-OCL 2.0 packages also prove various user-defined properties (lemmata) over the UML/OCL model. In our example, the Class Model Package already proved that down casting an object X from the topmost class `OclAny` to `Savings` does yield an error if X is not a subtype of `Account`:

lemma assumes $\tau \not\models X$`.oclIsUndefined()`
assumes $\tau \not\models X$`.oclIsKindOf(Account)`
shows $\tau \models (X :: \cdot$`OclAny`$)$`.oclAsType(Savings).oclIsInvalid()`

1.2 Organization of this Thesis

(Chapter 2) After a more detailed high-level introduction into the formal framework Isabelle in which this work is done,

(Chapter 3) we give an introduction on object-oriented modelling in UML/OCL and provide an in-depth comparison of UML/OCL to other object-oriented languages such as Eiffel or JML.

Then, we present the *main contributions* of our work:

(Chapter 4) we introduce our tool for UML/OCL, namely HOL-OCL 2.0, with an emphasis on Featherweight OCL, its semantic foundation,

(Chapter 5) we reveal its technical architecture and implementation based on the Isabelle framework,

(Chapter 6) we provide several means to practice meta theorem proving and interactive generations in HOL-OCL 2.0,

(Chapter 7) we construct the object-oriented datatype theory inside HOL, and instantiate the resulting formal semantics issued from the tool. We give a method to describe sub-typing semantically and embed it into languages with Milner-style type inference.

(Chapter 8) Finally, from an end-user perspective, we evaluate our system in a collection of medium-sized case studies, with a discussion of *corner-cases* and consequences resulting from semantic decisions, in particular with regard to the two exception elements `invalid` and `null`, and

(Chapter 9) discuss our lessons learned from following two different implementation strategies for building a formal UML/OCL tool based on Isabelle/HOL.

CHAPTER 2

Background: The Isabelle Framework

2.1 A Gentle Introduction to Isabelle

Isabelle [NPW02] is a *generic* theorem prover. New object-logics can be introduced by specifying their syntax and natural deduction inference rules. Among many logics, Isabelle supports First-Order Logic (FOL), Zermelo-Fraenkel set theory, and for instance Church's Higher-Order Logic (HOL).

The core language of Isabelle is a typed λ-calculus providing a uniform term language T in which all logical entities are represented:[1]

$$T \ ::= \ C \mid V \mid \lambda V.\ T \mid T\ T$$

where:

- C is the set of *constant symbols* like operators on pairs "fst" or "snd". Isabelle's syntax engine supports mixfix notation for terms. "$(\_ \Longrightarrow \_)\ A\ B$" or "$(\_ + \_)\ A\ B$" can be parsed and respectively printed as "$A \Longrightarrow B$" or "$A + B$".

- V is the set of *variable symbols* like x, y, z... Variables standing in the scope of a λ-operator are called *bound* variables, all others are *free* variables.

- $\lambda\ V.\ T$ is called a λ-*abstraction*, like as example the identity function $\lambda\ x.\, x$. A λ-abstraction forms a scope for the variable V.

- $T\ T'$ is called an *application*.

These concepts are not at all Isabelle specific and can be found in many modern programming languages ranging from Haskell [HHJW07] over Python [vR07] to Java.

Terms are associated to *types* by a set of *type inference rules*, similar to the Hindley-Milner type system [Hin69, Mil78, DM82]. Only terms for which a type can be inferred are considered as legal input to the Isabelle system, such terms

[1] In the Isabelle implementation, there are actually two further variants, they are irrelevant for this presentation and can be therefore omitted.

are *typed terms*. The type τ of typed terms can be inductively defined:[2]

$$\tau \ ::= \ TV \mid TV :: \Xi \mid \tau \ \Rightarrow \ \tau \mid (\tau, \ldots, \tau)\, TC$$

- TV is the set of *type variables* like $'\alpha$, $'\beta$, ... The syntactic categories V and TV are disjoint, thus $'x$ is a possible type variable.
- Ξ is a set of *type-classes* [WB89] like "ord", "order", "linorder"... This feature in the Isabelle type system is inspired by Haskell type classes. A *type class constraint* such as $'\alpha :: \text{order}$ expresses that the type variable $'\alpha$ may range over any type that has the algebraic structure of a partial ordering (as it is configured in the Isabelle/HOL library).
- The type $\tau_1 \Rightarrow \tau_2$ denotes the total function space from τ_1 to τ_2.
- TC is a set of *type constructors* like "$('\alpha)\,\text{list}$" or "$('\alpha)\,\text{tree}$". Again, Isabelle's syntax engine supports mixfix notation for type terms: e. g. cartesian products $'\alpha \times '\beta$ are understood as $('\alpha,'\beta)\,\text{prod}$. Also null-ary type-constructors like "$()\,\text{bool}$", "$()\,\text{nat}$" and "$()\,\text{int}$" are possible, although the parentheses of nullary type constructors are usually omitted.

In the following, to designate elements in TV, we will usually omit the quote "$'$" symbol in front of lowercase Greek letters.

Isabelle accepts also the notation $t :: \tau$ as type assertion in the term language, where $t :: \tau$ means "t is required to have the type τ". The type of typed terms *can* contain free type variables, like in the types of x and y when the system is automatically inferring this term $x + y = y + x$. By convention, free type variables are implicitly universally quantified.

An environment providing Ξ, TC and a map from constant symbols C to types (built over these Ξ and TC) is called a *global context*. It provides a kind of signature or a mechanism to construct the syntactic material of a logical theory.

The most basic (built-in) global context of Isabelle provides just a language to construct logical rules. More concretely, it provides a constant declaration for the (built-in) *meta-level implication* $\_ \Longrightarrow \_$ allowing to form constructs like $A_1 \Longrightarrow \cdots \Longrightarrow A_n \Longrightarrow A_{n+1}$, which are viewed as a *rule* of the form "from assumptions A_1 to A_n, infer conclusion A_{n+1}" and which is written in Isabelle syntax as:

$$[\![A_1; \ldots; A_n]\!] \Longrightarrow A_{n+1} \quad \text{or also usually seen as:} \quad \frac{A_1 \quad \cdots \quad A_n}{A_{n+1}}$$

Moreover, the built-in meta-level quantification $\text{Forall}(\lambda\, x.\ E\, x)$, pretty-printed and parsed as $\bigwedge x.\ E\ x$, captures the usual side-constraints "x must not occur free in the assumptions" for quantifier rules. Meta-quantified variables can be considered as "fresh" free variables. Meta-level quantification leads to a generalization of Horn-clauses of the form:

$$\bigwedge x_1, \ldots, x_m.\ [\![A_1; \ldots; A_n]\!] \Longrightarrow A_{n+1}$$

[2] Our presentation is again slightly different than the Isabelle implementation to improve readability.

Isabelle supports forward and backward reasoning on rules. For backward-reasoning, a *proof-state* can be initialized in a given global context and further transformed during the proof. For example, a proof of ϕ, using the Isabelle/Isar [Wen02] language, will look as follows in Isabelle:

```
lemma label : φ
  apply (case_tac [...])
  apply simp_all
done
```

(In this document, we will sometimes simply abbreviate Isabelle/Isar as Isar.) This proof script instructs the Isabelle system to prove ϕ by case distinction followed by a simplification of all resulting proof states ("The simplifier" is described in section 9.3 in the manual [Wen16b]). Such a proof state is a sequence of generalized Horn-clauses (called *subgoals*) $\phi_1, \ldots, \phi_n$ with a *goal* ϕ. Proof states are usually represented in mathematical textbooks as:

$$\begin{array}{rl} \text{label}: & \phi \\ 1. & \phi_1 \\ & \vdots \\ n. & \phi_n \end{array}$$

Subgoals and goals may be extracted from the proof state into theorems of the form $[\![\phi_1; \ldots; \phi_n]\!] \Longrightarrow \phi$ at any time.

By extending global contexts with theorems, axioms and proofs, we get at the end a *theory* which has been constructed step by step. Beyond the basic mechanism of extending a global context with raw types (with type constructors, type class, constant definitions, or axioms), Isabelle offers a number of *commands* that allow for more complex extensions of theories in a logically safe way, i. e., by directly avoiding the use of axioms. In this document, we will use the same colour for commands as they appear in Isabelle/jEdit. Although commands appear most of the time in blue: "lemma", "datatype", "theory"; certain commands are also rendered in red like apply or done. However to simplify the presentation, in this document, the colour of commands can merely be considered as a syntactic indication with no particular meaning (i. e., in the source code, commands are essentially seemingly built).[3]

2.2 Higher-Order Logic (HOL)

Higher-Order Logic (HOL) [Chu40, And02] is a classical logic based on a simple type system. Isabelle/HOL is a theory extension of the basic Isabelle core language with operators and the seven axioms of HOL. Together with large libraries, the overall constitutes an implementation of HOL. (Thus we will sometimes simply abbreviate Isabelle/HOL by HOL in this document.) Isabelle/HOL provides the usual logical connectives like $\_ \wedge \_$, $\_ \rightarrow \_$, $\neg \_$ as well as the object logical quantifiers $\forall x.\ P\ x$ and $\exists x.\ P\ x$. In contrast to FOL, quantifiers may range over arbitrary types, including total functions $f :: \tau_1 \Rightarrow \tau_2$. HOL is centered around

[3] Although the command end is normally rendered in green, to avoid potential confusions we will depict it in black and in bold format, mostly in Chapter 5 and Chapter 6: so "**end**".

extensional equality $\_ = \_ :: \alpha \Rightarrow \alpha \Rightarrow$ bool. Extensional equality means that two functions f and g are equal if and only if they are point-wise equal. This is captured by the rule: $(\bigwedge x.\ f\ x = g\ x) \Longrightarrow f = g$. HOL is more expressive than FOL, since among many other things, induction schemes can be expressed inside the logic. For example, the standard induction rule on natural numbers in HOL:

$$P\ 0 \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow P\ (x+1)) \Longrightarrow P\ x$$

is just an ordinary rule in Isabelle which is in fact a proven theorem in the theory of natural numbers. This example exemplifies an important design principle of Isabelle: theorems and rules are technically the same, paving the way to *derived rules* and automated decision procedures based on them. This has the consequence that these procedures are consequently sound by construction with respect to their logical aspects (they may be incomplete or failing, though).

On the one hand, Isabelle/HOL can be viewed as a functional programming language like SML [Mil97] or Haskell, by reading Isabelle/HOL definitions as one is reading any declarations in a functional **programming** language, i. e. by omitting the reading of Isar proof scripts. Conversely, type definitions in a functional programming language can be viewed as formulae part of the **specification** language of Isabelle/HOL. Generally in this document, we will simply abbreviate elements belonging to the Isabelle/Isar language or the Isabelle/HOL language as just Isabelle/Isar_HOL (or Isar_HOL).

Isabelle/HOL offers support for a particular methodology to extend given theories in a logically safe way: a theory extension is *conservative* if the provability of a formula in the extended theory is the same as in the original theory. Then the consistency of an extended theory depends on the consistency of the original one. Conservative extensions apply to different families of definitions: *constant definitions*, *type definitions*, *datatype definitions*, *primitive recursive definitions* and *well founded recursive definitions*.

Embedding a language $\mathcal{L}$ into an object-logic like HOL consists to assimilate the largest possible subset of $\mathcal{L}$ as integral constituent of the object-logic. Consequently, the aim is to maximize the support of $\mathcal{L}$ in an unambiguous way, assuming the trust one might have on the object-logic. Trust also depends on how embeddings are performed. *Deep embedding* and *shallow embedding* are seen as the two possible complementary method of embedding for a language [BGG+93]. Generally, using shallow embeddings for a formal specification or programming language in HOL is by no means a new technique [JS94, ACM94, BRW03, BW09]. Over the years, a substantial body of languages and tools have been developed along this line, which have seen substantial applications—we cite only the current flagships of this development Isabelle/SIMPL [Sch08] and the seL4 verification project [KAE+10].

Some Libraries and Operations of Isabelle/HOL

Isabelle/HOL provides a large collection of theories like sets, lists, orderings, and various arithmetic theories. Theories only contain rules derived from conservative definitions. As an example of conservative extension, the library includes the type constructor $\tau_\bot := \bot|_{\llcorner\_} :: \alpha_\lrcorner$ that assigns to each type τ a type $\tau_\bot$ *disjointly extended* by the exceptional element $\bot$. The function $\ulcorner\_\urcorner :: \alpha_\bot \Rightarrow \alpha$ is the inverse of $\llcorner\_\lrcorner$ (it is unspecified for $\bot$). Partial functions $\alpha \rightharpoonup \beta$ are defined as

functions $\alpha \Rightarrow \beta_\perp$ supporting the usual concepts of domain "dom _" and range "ran _".

As another example, typed sets are conservatively built in the Isabelle libraries on top of the kernel of HOL as functions to bool. Consequently, the constant definitions for membership is as follows:[4]

type_synonym	α set	$= \alpha \Rightarrow$ bool	
definition	Collect	$:: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha$ set	— set comprehension
where	Collect S	$\equiv S$	
definition	member	$:: \alpha \Rightarrow \alpha$ set $\Rightarrow$ bool	— membership test
where	member $s\ S$	$\equiv Ss$	

Isabelle's syntax engine is instructed to accept the notation $\{x \mid P\}$ for Collect $(\lambda x.\ P)$ and the notation $s \in S$ for member $s\ S$. As it can be inferred from the example, constant definitions are axioms that introduce a fresh constant symbol (which must not be based on a recursive expression, or having free variables). This type of axiom is logically safe since it works like an abbreviation. The syntactic side conditions of so-introduced axioms are mechanically checked. Then it becomes straightforward to express the usual operations on sets as conservative extensions too, like for example $\_ \cup \_$, $\_ \cap \_ :: \alpha$ set $\Rightarrow \alpha$ set $\Rightarrow \alpha$ set.

Similarly, a set of logical rules are "compiled" from the following statements, which introduce the types option and list:

datatype	α option	$=$ None		$\mid$ Some α	
datatype	α list	$=$ Nil	("[]")	$\mid$ Cons α "α list"	(infixr "#" 65)

Here "[]" or "_#_" are an alternative syntax for Nil or Cons $a\ l$. Moreover, the commands syntax and translations [Wen16b] can additionally (recursively) define $[a, b, c]$ as an alternative syntax for $a\#b\#c\#[]$. Besides the *constructors* None, Some, [] and Cons, there is the matching operation to conditionally return a term by case analysis provided a general term x, whose type has been defined with datatype, as example:

case x of None $\Rightarrow F \mid$ Some $a \Rightarrow G\ a$

The datatype package automatically derives a set of properties in front of each command datatype [BW99, BHL+14, BDP+16]. One way to understand this command is to view it as a kind of macro (albeit its syntax is inspired by functional programming languages), which generates a number of constant definitions and theorems from the type declaration option or list. So the generated lemmas are also implicitly proved in the background, this command constructs a model of the constructors and derive its properties:

$$(\text{case } [] \text{ of } [] \Rightarrow F \mid (a\#r) \Rightarrow G\ a\ r) = F$$
$$(\text{case } b\#t \text{ of } [] \Rightarrow F \mid (a\#r) \Rightarrow G\ a\ r) = G\ b\ t$$
$$[] \neq a\#t \quad \text{– distinctness}$$
$$(a = [] \Longrightarrow P) \Longrightarrow (\bigwedge x\ t.\ a = x\#t \Longrightarrow P) \Longrightarrow P \quad \text{– exhaust}$$
$$P\ [] \Longrightarrow (\bigwedge a\ t.\ P\ t \Longrightarrow P(a\#t)) \Longrightarrow P\ x \quad \text{– induct}$$

[4] To increase readability, we use a slightly simplified presentation. The complete details can be inspected in `$ISABELLE_HOME/src/HOL/Set.thy` (in Isabelle version 2016).

Besides datatype, other packages are natively present when starting Isabelle. For example the fun command serves to define well-founded recursive functions [Kra06, Kra16]. Thus, we may define the sort operation on linearly ordered lists as follows:

```
fun    ins              ::[α :: linorder, α list] ⇒ α list
where  ins x [ ]        = [x]
       ins x (y#ys)     = if x < y then x#y#ys else y#(ins x ys)
fun    sort             ::(α :: linorder) list ⇒ α list
where  sort [ ]         = [ ]
       sort(x#xs)       = ins x (sort xs)
```

Similar as datatype, the fun command can again be seen as a kind of macro: a conservative construction is implied; the derivation of the equations ins $x[\,] = [x]$ and ins $x(y\#ys) =$ if $x < y$ then $x\#y\#ys$ else $y\#(\text{ins}\ x\,ys)$ is done automatically involving a termination proof (most of the time automatically proved for basic functions). This involved construction assures logical safeness: in general, just adding axioms for recursive equations causes inconsistency for non-terminating functions. The resulting equations can now be used in the Isabelle simplifier.

The library of Isabelle/HOL constitutes a comfortable basis for defining the OCL library or embed a specification language. In particular, Isabelle manages a set of *executable types and operators*, i. e., types and operators for which a compilation to external languages is possible, using *code-generation* [Haf09, HN10, Haf16]. The supported external languages in Isabelle for code-generation are currently Haskell, OCaml [LDF+14], Scala and SML. As one example, arithmetic types such as int are appropriately optimized to be executed fast depending on the chosen external language. Datatypes and recursive functions are as well supported to be executed in these external languages (assuming their definitions contain only executable operators).

Another mean to do executions in Isabelle is to use the value command (whose functioning resembles to how code-generation works) [Wen16b]. Then, after typing value "3 + 7" in Isabelle/jEdit [Wen12, Wen16c], we will get 10 as result. Generally value can work with many ground expressions (with no free variables). So most of OCL ground terms are in fact executable in Isabelle, due to prior special setups in the Featherweight OCL library.

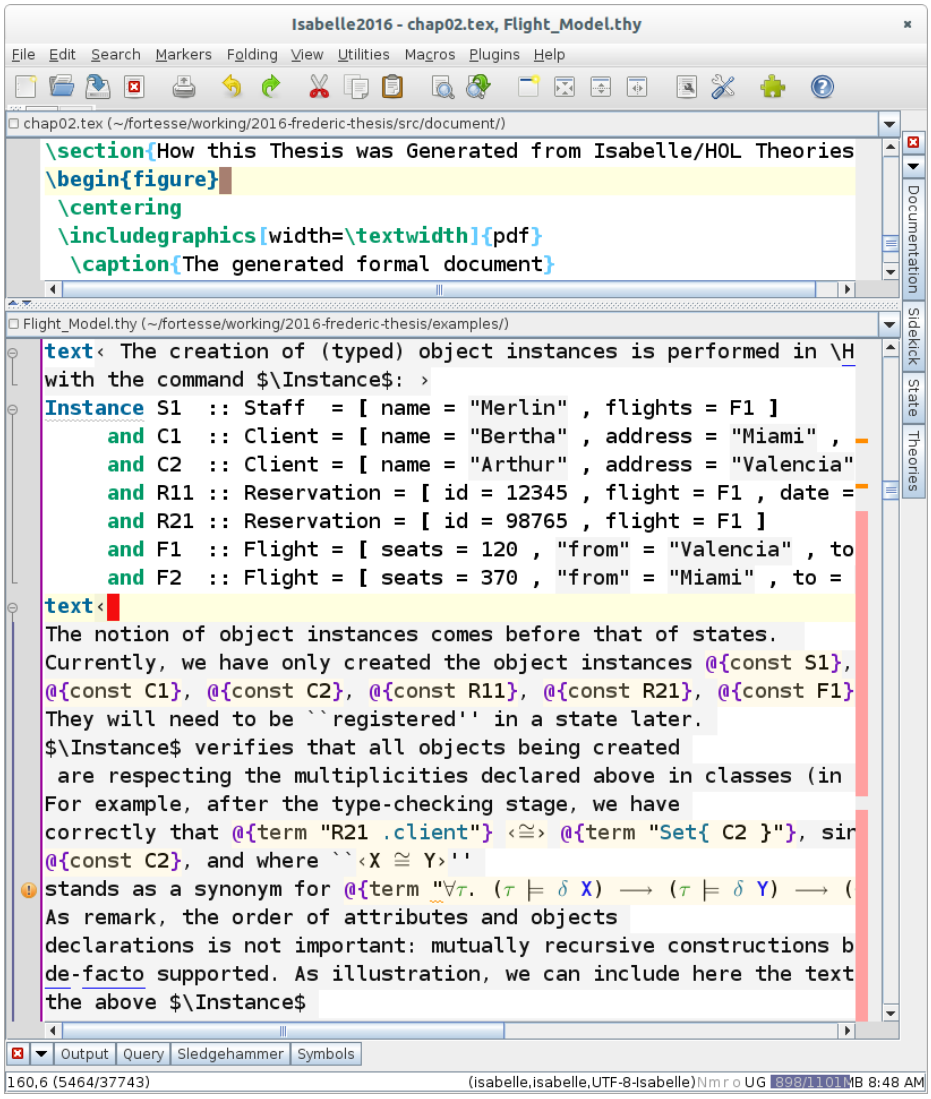


Figure 2.1: The Isabelle/jEdit environment

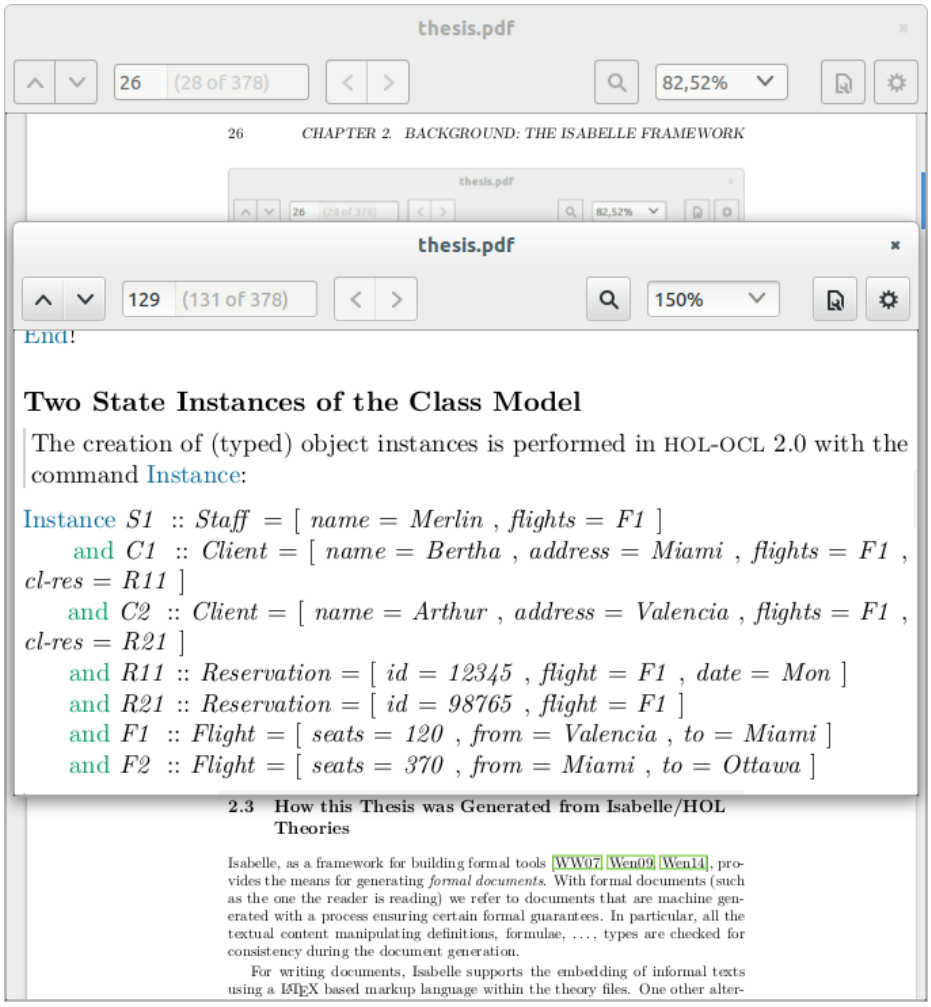


Figure 2.2: The generated formal document

2.3 How this Thesis was Generated from Isabelle/HOL Theories

Isabelle, as a framework for building formal tools [WW07, Wen09, Wen14], provides the means for generating *formal documents*. With formal documents (such as the one the reader is reading) we refer to documents that are machine generated with a process ensuring certain formal guarantees. In particular, all the textual content manipulating definitions, formulae, ..., types are checked for consistency during the document generation.

For writing documents, Isabelle supports the embedding of informal texts using a LaTeX based markup language within the theory files. One other alternative to embed informal documents is to directly write LaTeX code in usual "`_.tex`" files, and then link them with the formal content generated by Isabelle. Generally, by manually inspecting the source code of Isabelle theory files, one

can have a clear estimation of the size of informal texts versus formal texts of a given project. Many similar recommendations regarding certification practices can be found for example in a recent LRI's technical report [NFWP15]. In this document, all the formal contents are respectively situated in:

- Section 8.2, Appendix A: the Appendix version is a version where proofs are displayed.
- Appendix B, Appendix C
- Appendix D
- Appendix E, Appendix F, Appendix G, Appendix H
- Appendix I, Appendix J

Everything else was "informally" written by hand. This does not mean however that the formal contents have also been written by hand, in particular Appendix B and Appendix C were generated. As remark, not all formal contents have been included in the present thesis, otherwise we would obtain a document exceeding 1000 pages (and without counting the size of generated content like Appendix B and Appendix C).

Still, to ensure consistencies of certain informal parts, Isabelle supports the use of *antiquotations* within informal texts, that refer to the formal parts and that are checked while generating the actual document as PDF. For example, in an informal text, the antiquotation "`@{`thm *OclNot_not*`}`" will instruct Isabelle to abort the generation with an error in case no OCL theorems with the name *OclNot_not* were found, otherwise the system will replace the antiquotation with the actual theorem, i. e. "$\texttt{not}\,(\texttt{not}\,X) = X$" (as it is the case here). So one can notice at this point that the size of informal content also depends on the size of the (expanded) generated content.

We illustrate the approach: Figure 2.1 shows the jEdit-based development environment of Isabelle. At the bottom, we have an excerpt of one of the core theories of this thesis, mixing both informal texts and formal texts (with some antiquotations in the informal texts), whereas at the top we have a "true" informal content in LaTeX. Figure 2.2 shows only two superimposed windows, offering different views on the generated PDF document, where in particular all corresponding antiquotations have been correctly resolved.

CHAPTER 3

Background: UML/OCL

3.1 UML/OCL and its Semantics

Object-oriented, class-based constraint or generally behavioral specification languages, such as ACSL, JML, or Spec# are domain-specific logical languages used to express properties (usually in the form of contracts, invariants of classes as well as pre-conditions and post-conditions of methods) on the manipulated models, e. g., object-oriented data models or object graphs. These object-oriented data models are usually defined in an object-oriented modelling or programming language. For example, OCL [Obj12] allows to express constraints over data models defined in UML [Obj11b] while the Java Modeling Language (JML) is employed to specify constraints over Java programs. In the following, we will introduce UML/OCL as an example of an object-oriented specification language and, thereafter, will briefly compare with other specification languages, such as ACSL, JML, or Spec#.

UML/OCL as OO Specification Languages

The Unified Modelling Language (UML) [Obj11a, Obj11b] is one of the most widely used diagrammatic object-oriented modelling language in industry. Besides a number of widely popular visualisation formats for some aspects of an "UML Model," it offers a normed abstract-syntax (defined by the "UML Meta-Model") processed by several IDE's; the language and tool-support is particularly suited for defining domain-specific (sub)-languages. UML is defined in an open process by the Object Management Group (OMG), i. e., an industry consortium. For some parts of the language formal analysis tools are available based on an OMG standardised or tool-vendor specific formal semantics. While UML is mostly known as diagrammatic modelling language (e. g., visualizing class models), it also comprises a textual language, called Object Constraint Language (OCL) [Obj12]. OCL is a textual annotation language, originally conceived as a three-valued logic, that turns substantial parts of UML into a formal language. Unfortunately the semantics of this specification language, captured in the "Annex A" (originally, based on the work of Mark Richters [Ric02]) of the OCL standard leads to different interpretations of corner cases. Many of these corner cases had been subject to formal analysis since more than nearly fifteen years (for

example [LTW14, BLTW13, BKLW10, BW02b, CKM+02, MC99, HCH+98]).

At its origins [Ric02, Obj97], OCL was conceived as a strict semantics for undefinedness (e. g., denoted by the element `invalid`[1]), with the exception of the logical connectives of type `Boolean` that constitute a three-valued propositional logic. At its core, OCL comprises four layers:

1. Operators (e. g., `_ and _`, `_ + _`) on built-in data structures such as `Boolean`, `Integer`, or typed sets (`Set(_)`).
2. Operators on the user-defined data model (e. g., defined as part of a UML class model) such as accessors, type casts and tests.
3. Arbitrary, user-defined, side-effect-free methods called *queries*,
4. Specification for invariants on states and contracts for operations to be specified via pre- and post-conditions.

Motivated by the need for aligning OCL closer with UML, recent versions of the OCL standard [Obj06, Obj12] added a second exception element. While the first exception element `invalid` has a strict semantics, `null` has a non strict semantic interpretation. Unfortunately, this extension results in several inconsistencies and contradictions. These problems are reflected in difficulties to define interpreters, code-generators, specification animators or theorem provers for OCL in a uniform manner and resulting incompatibilities of various tools.

For the OCL community, the semantics of `invalid` and `null` as well as many related issues resulted in the challenge to define a consistent version of the OCL standard that is well aligned with the recent developments of the UML. A syntactical and semantical consistent standard requires a major revision of both the informal and formal parts of the standard. To discuss the future directions of the standard, several OCL experts met in November 2013 in Aachen to discuss possible mid-term improvements of OCL, strategies of standardization of OCL within the OMG, and a vision for possible long-term developments of the language [BCC+13]. During this meeting, a Request for Proposals (RFP) for OCL 2.5 was finalized and meanwhile proposed. In particular, this RFP requires that the future OCL 2.5 standard document shall be generated from a machine-checked source. This will ensure

- the absence of syntax errors,
- the consistency of the formal semantics,
- a suite of corner-cases relevant for OCL tool implementors.

As a basis of this work, we develop in this thesis *HOL-OCL 2.0* in Isabelle/HOL [NPW02][2]. HOL-OCL 2.0 comes with a machine-checked library formalizing a core language of OCL, called *Featherweight OCL* [BTW14][3]. The

[1] In earlier versions of the OCL standard, this element was called `OclUndefined`.

[2] The development version of HOL-OCL 2.0 can be inspected online: `https://projects.brucker.ch/hol-testgen/log/trunk/hol-testgen/add-ons/Featherweight-OCL`.

[3] The updated machine-checked version is maintained by the Isabelle Archive of Formal Proofs (AFP), see also the Bitbucket repository `https://bitbucket.org/isa-afp/afp-devel` and its list of maintainers `https://bitbucket.org/isa-afp/profile/members`.

semantic theory of Featherweight OCL is based on a "shallow embedding" and focuses on a formal treatment of the key-elements of OCL (rather than a full treatment of all operators and thus, a "complete" implementation). In contrast to full OCL, it comprises just the logic captured in `Boolean`, the basic datatypes `Void`, `Integer`, `Real` and `String`, the collection types `Set`, `Pair`, `Sequence` and `Bag`. The generic construction principle of class models is also supported [TW15][4], we will precisely demonstrate in Chapter 5 how to generate this type-safe construction, with respective instantiations in Chapter 7, Appendix B and Appendix C. The formal semantics developed in Featherweight OCL is intended to be a proposal for the standardization process of OCL 2.5, which should ultimately replace parts of the mandatory part of the standard document [Obj12] as well as replace completely its informative "Annex A."

The semantic definitions are in large parts executable, namely the essence of `Set`, `Pair`, `Sequence` and `Bag` constructions (as remark, HOL is a classical logic where some parts could be not constructively defined). The first goal of its construction is *consistency*, i. e., it should be possible to apply logical rules and/or evaluation rules for OCL in an arbitrary manner always yielding the same result. Moreover, except in pathological cases, this result should be unambiguously defined, i. e., represent a value.

To motivate the need for logical consistency and also the magnitude of the problem, we focus on one particular feature of the language as example: `Tuples`. Recall that tuples (in other languages known as *records*) are n-ary Cartesian products with named components, where the component names are used also as projection functions: the special case `Pair{x:First, y:Second}` stands for the usual binary pairing operator `Pair{true, null}` and the two projection functions `x.First()` and `x.Second()`. For a developer of a compiler or proof-tool (based on, say, a connection to an SMT solver designed to animate OCL contracts) it would be natural to add the rules `Pair{X, Y}.First() = X` and `Pair{X, Y}.Second() = Y` to give pairings the usual semantics. At some place, the OCL Standard requires the existence of a constant symbol `invalid` and requires all operators to be strict. To implement this, the developer might be tempted to add a generator for corresponding strictness axioms, producing among hundreds of other rules `Pair{invalid, Y} = invalid`, `Pair{X, invalid} = invalid`, `invalid.First() = invalid`, `invalid.Second() = invalid`, etc. Unfortunately, this "natural" axiomatization of pairing and projection together with strictness is already inconsistent. One can derive:

```
Pair{true, invalid}.First() = invalid.First()
                            = invalid
```

and:

```
Pair{true, invalid}.First() = true
```

which then results in the absurd logical consequence that `invalid = true`. Obviously, we need to be more careful on the side-conditions of our rules. And obviously, only a mechanized check of these definitions, following a rigorous

[4] Again, the updated machine-checked version is maintained by the Isabelle Archive of Formal Proofs.

methodology, can establish strong guarantees for logical consistency of the OCL language.

This leads us to our second goal of this document: it should not only be usable by logicians, but also by developers of compilers and proof-tools. For this end, we *derive* from the Isabelle framework, many definitions and *logical rules* for formal interactive and automated proofs on UML/OCL specifications. These logical rules are necessary for *execution rules* and *test-cases* to reveal potential corner-cases related with the semantics the implementors are defining.

OCL is an annotation language for UML models, in particular class models allowing for specifying data and operations on them. As such, it is a *typed* object-oriented language. This means that it is—like Java or C++—based on the concept of a *static type*, that is the type that the type-checker infers from a UML class model and its OCL annotation, as well as a *dynamic type*, that is the type at which an object is dynamically created[5]. Types are not only a means for efficient compilation and a support of separation of concerns in programming, there are of fundamental importance for our goal of logical consistency: it is impossible to have sets that contain themselves, i. e., to state Russell's paradox in OCL typed set-theory. Moreover, object-oriented typing means that types can be in sub-typing relation; technically speaking, this means that any object X can be *cast* with the operator $(\_ :: C_i)\texttt{.oclAsType(}C_j\texttt{)}$ from one class types C_i to another class types C_j, and under particular conditions (to be later described), these casts are semantically *lossless*:

$$(X :: C_i)\texttt{.oclAsType(}C_j\texttt{).oclAsType(}C_i\texttt{)} = X$$

Furthermore, object-oriented means that operations and object-types can be grouped to *classes* on which an inheritance relation can be established; the latter induces a sub-type relation between the corresponding types.

Here is a feature-list of Featherweight OCL:

- it specifies key built-in types such as `Boolean`, `Void`, `Integer`, `Real` and `String` as well as generic types such as `Pair(T,T')`, `Sequence(T)`, `Bag(T)` and `Set(T)`.
- it defines the semantics of the operations of these types in *denotational form* (to be explained in Chapter 4), and thus in an unambiguous (and in Isabelle/HOL executable or animatable) way.
- it develops the *theory* of these definitions, i. e., the collection of lemmas and theorems that can be proven from these definitions.
- all types in Featherweight OCL contain the elements `null` and `invalid`; including in particular the `Boolean` type, so we obtain a four-valued logic. Consequently, Featherweight OCL contains the derivation of the *logic* of OCL.
- collection types may contain `null` (so `Set{null}` is a defined set) but not `invalid` (`Set{invalid}` is just `invalid`).

[5] As side-effect free language, OCL has no object-constructors, but with `OclIsNew()`, the effect of object creation can be expressed in a declarative way.

- With respect to the static types, Featherweight OCL is a strongly typed language in the Hindley-Milner tradition. So the explicit usage of casts are needed whenever for example one attempts to apply an attribute a to an object $X :: C_i$, and where a has been defined in C_j (so not in C_i). On the other hand, one can also assume there is a pre-processing to automatically introduce these explicit conversions (i. e., to remove the need to write `.oclAsType(_)`).[6]

- Featherweight OCL types may be arbitrarily nested. For example, the expression `Set{Set{1,2}} = Set{Set{2,1}}` is legal and true.

- All object types are represented in an object universe[7]. The universe construction also gives semantics to type casts, dynamic type tests, as well as functions such as `allInstances()`, or `oclIsNew()`. The object universe construction is conceptually described and demonstrated at an example.

- As part of the OCL logic, Featherweight OCL develops the theory of equality in UML/OCL. This includes the standard equality, which is a computable strict equality using the object references for comparison, and the not necessarily computable logical equality, which expresses the Leibniz principle that "equals may be replaced by equals" in OCL terms.

- Technically, Featherweight OCL is a *semantic embedding* into a powerful semantic meta-language and environment, namely Isabelle/HOL [NPW02]. It is a so-called *shallow embedding* in HOL; this means that types in OCL are mapped one-to-one to types in Isabelle/HOL. Ill-typed OCL specifications can therefore not be represented in Featherweight OCL and a type in Featherweight OCL contains exactly the values that are possible in OCL.

3.2 A Running Example for UML/OCL

The Unified Modelling Language (UML) [Obj11a, Obj11b] comprises a variety of model types for describing static (e. g., class models, object models) and dynamic (e. g., state-machines, activity graphs) system properties. One of the more prominent model types of the UML is the *class model* (visualized as *class diagram*) for modelling the underlying data model of a system in an object-oriented manner.

Throughout this document, we will use a small example describing a set of flights and their passengers, being clients with reservations or staff onboard. The journey of a client may be a sequence of flights, each one departing from the city of arrival of the previous one. The client must have a reservation on all the flights composing his journey. The passengers of a flight are the clients having reservation for this flight and the staff working onboard. A flight cannot have more clients onboard than the number of seats.

Figure 3.1 shows the UML class diagram of this particular flight example. Figure 3.2 shows an alternative textual representation of this model (which is

[6] The details of such a pre-processing are present in HOL-OCL [Bru07] and can be similarly adapted for Featherweight OCL.

[7] following the tradition of HOL-OCL [BW08b]

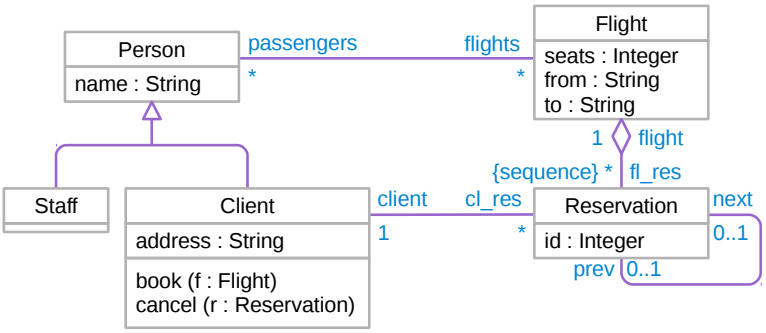


Figure 3.1: A simple class model capturing flight reservations.

```
Class Flight
 Attributes seats    : Integer
            from     : String
            to       : String  End

Class Reservation
 Attributes id       : Integer End

Class Person
 Attributes name     : String  End

Class Client < Person
 Attributes address : String   End

Class Staff  < Person          End
```

```
Association passengers    Between
 Person        [*]        Role passengers
 Flight        [*]        Role flights End

Aggregation flights       Between
 Flight        [1]        Role flight
 Reservation   [*]        Role fl_res
                Sequence              End
Association reservations Between
 Client        [1]        Role client
 Reservation   [*]        Role cl_res  End

Association connection    Between
 Reservation   [0 .. 1]   Role next
 Reservation   [0 .. 1]   Role prev    End
```

Figure 3.2: Modelling flight reservations in HOL-OCL 2.0: data part

supported by our HOL-OCL 2.0 tool as well as the USE tool [RG02]). This example contains the major constructs in UML class models: classes and inheritance hierarchies, collection annotations and cardinalities on association ends, (self) associations and (self) aggregations, along which navigations are possible.

We model persons and flights as *classes* `Person` and `Flight`, as we would do in Java.[8] Classes can have attributes (e. g. the number of `seats`) as well as *associations* to other classes. Associations allow us to model relations between objects. For example, we model the relation of being a passenger as an association between the classes `Person` and `Flight`. Overall, associations in UML are very similar to relations in entity-relationship (ER) models [Che76] and as relations in ER models, UML associations are equipped with *multiplicities*. The multiplicity `*` of the *association end* `flights` models that each instance of the class, i. e. each *object* `Person` can be associated to arbitrary many (including zero) instances of the class `Flight` (a person can be no passenger at all or a passenger of one or several flights). An association may be more than a simple relation between classes when these classes participate in a whole/part relationship [BHOG01]: such an association is called an aggregation and is depicted by an unfilled diamond. In the example, a flight is associated to a sequence of reser-

[8] For sake of simplicity, we assume that the same flight on different dates is in fact represented by different instances of the class `Flight`, without explicitly modelling this by an attribute of type `Date`.

```
Context f: Flight
  Inv B : f.fl_res->size() <= f.seats
  Inv C : f.passengers->select(p|p.oclIsTypeOf(Client))
            = f.fl_res->collect(r|r.client.oclAsType(Person))->asSet()

Context r: Reservation
  Inv A : r.id > 0
  Inv B : r.next <> null implies r.flight.to = r.next.flight.from
  Inv C : r.next <> null implies r.client    = r.next.client

Context c: Client :: book (f : Flight)
 Pre : f.passengers->excludes(c.oclAsType(Person))
       and f.fl_res->size() < f.seats
 Post: f.passengers =
  f.passengers@pre->including(c.oclAsType(Person)) and
   let r = c.cl_res->select(r|r.flight = f)->any() in
   r.oclIsNew() and r.prev = null and r.next = null
```

Figure 3.3: Modelling flight reservations in HOL-OCL 2.0: OCL part

vations by an aggregation, meaning that this sequence is part of the description of a flight, as the number of seats is.

Many object-oriented programming languages, such as Java, do not support associations (or relations) as first-class citizens: associations are usually represented as if they were aggregations, by collection type attributes for the association ends together with additional constraints that need to ensure the consistency of the objects taking part in the association. Besides associations, UML supports the *inheritance* relation between classes (also called *generalization*): in the example, the class `Client` is a *sub-class* (sub-type) of the class `Person` (*superclass*).

Such data models (as Figure 3.2) are, usually, not precise enough: our data model would allow flights with zero or even a negative number of seats. Object-oriented constraint languages allow to refine such data models and, thus, to avoid such unwanted states. We can use a simple *class invariant* to state that flights need to have a positive number of seats:

```
Context f: Flight
  Inv A: f.seats > 0
```

We can also use *operation contracts* to specify operations' behaviour in terms of pre- and post-conditions. For example, for a client to cancel a reservation, this client must own the reservation (pre-condition). When the cancellation is done, this client does not have a reservation for this flight anymore (post-condition). The construct `@pre` does a referencing in the pre-state (like `\old` in JML and `OldValue` in Spec#).

```
Context c: Client :: cancel(r : Reservation)
 Pre : r.client = c
 Post: c.cl_res -> select(res|res.flight = r.flight@pre)
                -> isEmpty()
```

OCL is a four-valued logic with quantifiers, supporting the non-strict exception element `null` and the strict exception element `invalid`. Moreover, OCL supports a rich library of built-in datatypes including integers and typed sets.

In our Isabelle/HOL formalisation, HOL-OCL 2.0, we can express such invariants and contracts using a slightly different syntax for the context specification.

Figure 3.3 additionally shows a number of OCL constraints for our flight example. For instance, the first two constraints respectively state that the number of reservations must not exceed the number of seats; and the passengers taking a flight are exactly equal to those who have reserved the flight.

CHAPTER 4

Semantic Layers of Featherweight OCL

The semantic theory of Featherweight OCL is organized in several *semantic layers*. The following three layers will provide a "minimal" core semantics of built-in data-structures, so to support in particular the OCL type `Boolean`.

- The first layer, called the *denotational semantics* comprises a set of definitions of the operators of the language. Presented as *definitional axioms* inside Isabelle/HOL, this part assures the logically consistency of the overall construction. The denotational definitions of types, constants and operations, and OCL contracts represent the "gold standard" of the semantics.
- The second layer, called *logical layer*, is derived from the former and centered around the notion of validity of an OCL formula P. For a state-transition from pre-state σ to post-state σ', a validity statement is written $(\sigma, \sigma') \models P$. Its major purpose is to logically establish facts (lemmas and theorems) about the denotational definitions.
- The third layer, called *algebraic layer*, also derived from the former layers, tries to establish algebraic laws of the form $P = P'$; such laws are amenable to equational reasoning and also help for automated reasoning and code-generation. For an implementor of an OCL compiler, these consequences are of most interest.

Then come the next semantic layers covering construction of UML class models, composed of:

- the *state layer* describing state-related operations like `allInstances()`, and
- the *object-oriented datatype layers* giving semantics to UML class models over this, comprising the theory of accessors, type casts and tests.

For space reasons, we will restrict ourselves in this document to a few operators and make a traversal through all five layers to give a high-level description of our formalization. Especially, the details of the semantic construction for sets, sequences, bags are excluded from a presentation here, but can all be found in our associated formalization [BTW14]. Similarly, the semantics of UML/OCL operations and invariants is further made precise in that document.

4.1 Denotational Semantics of Types

Definition "**UML/OCL types**":
The syntactic material for type expressions, called $\mathrm{TYPES}(C, E)$, is inductively defined as follows:

- $C \subseteq \mathrm{TYPES}(C, E)$ are object types.
- $E \subseteq \mathrm{TYPES}(C, E)$ are enumerate types. Enumerate types are basically sum types: a form of Isabelle datatype without polymorphic parameters.
- `Void`, `Boolean`, `Integer`, `Real`, `String` are base types $\mathrm{T}_{base} \subseteq \mathrm{TYPES}(C, E)$.
- $\texttt{Sequence}_m\texttt{(}X\texttt{)}$, $\texttt{Set}_m\texttt{(}X\texttt{)}$, and $\texttt{Pair(}X\texttt{,}Y\texttt{)}$ are collection types in $\mathrm{TYPES}(C, E)$ if $X, Y \in \mathrm{TYPES}(C, E)$.

  These collection types are particular dependent types [SU06]: the multiplicity m is a list of intervals constraining the size of the corresponding sequence or set. An interval $\texttt{[}i_{min}\texttt{..}i_{max}\texttt{]}$ is composed of two lifted naturals nat of the form $(\mathrm{nat}_\bot \times \mathrm{nat}_\bot)$ where the bottom element is conventionally represented as a star "`*`", this additional element means an arbitrary allowed number. For a sequence or set to be classified as well-typed, it must exist one interval in the list m such that $i_{min} \leq s \leq i_{max}$, with s the size of the sequence or set.

  Whenever m evaluates to the interval $*$[1], the multiplicity information can be omitted and in this case we will just write $\texttt{Sequence(}X\texttt{)}$ and $\texttt{Set(}X\texttt{)}$.

  A syntactic sugar is provided for building arbitrary tuples: $\texttt{(}X_1\texttt{,}\cdots\texttt{,}X_n\texttt{)}$ is a shorthand for $\texttt{Pair(}X_1\texttt{,}\cdots\texttt{Pair(}X_{n-2}\texttt{,Pair(}X_{n-1}\texttt{,}X_n\texttt{))}\cdots\texttt{)}$ for $n \geq 2$. Types in tuples can be preceded with additional labelling variables $\texttt{(}x_1\texttt{:}X_1\texttt{,}\cdots\texttt{,}x_n\texttt{:}X_n\texttt{)}$ where $x_1, \cdots, x_n$ are labels for naming individuals of the respective types $X_1, \cdots, X_n$. These labels are typically used when defining UML/OCL contracts.
- $X\texttt{:}Y$ are functional types in $\mathrm{TYPES}(C, E)$ if $X, Y \in \mathrm{TYPES}(C, E)$.

  Like tuples, $\texttt{(}x\texttt{:}X\texttt{)}\texttt{:}Y$ is an additional syntax for describing functional types, where x is a stamped label. Functional types mainly appear together with tuples when writing UML/OCL contracts. Depending on the context, in positions where no ambiguities with tuples occur, functional types can be shorten to $\texttt{(}x_1\texttt{:}X_1\texttt{,}\cdots\texttt{,}x_n\texttt{:}X_n\texttt{)}$ (where $n \geq 1$), in this case the absent type Y has the same semantics as `Void`.

  As another notation, we can use $X\texttt{->}Y$ to represent functional types. Thus $\texttt{(}X_1\texttt{,}\cdots\texttt{,}X_n\texttt{)->}Y$ can be used without labelling names (as this does not conflict with tuples).

We define $\mathrm{TYPES}_0(C, E)$ as the smallest subset of $\mathrm{TYPES}(C, E)$ built without using functional types in all recursive calls. In the following, $\mathrm{TYPES}_0(C, E)$ and $\mathrm{TYPES}(C, E)$ will be respectively shorten to TYPES_0 and TYPES.

[1] The interval $*$ is a shortcut for $\texttt{[*..*]}$. We will abbreviate intervals $\texttt{[}i_{min}\texttt{..}i_{max}\texttt{]}$ by a single i_{min} if we have $i_{min} = i_{max}$.

The OCL core language is composed of

1. operators on built-in data structures such as `Boolean`, `Integer` or `Set(_)`,
2. operators of the user-defined data model such as accessors, type casts and tests, and
3. user-defined, side-effect-free methods.

Conceptually, an OCL expression in general and Boolean expressions in particular (i. e. *formulae*) depends on a pair (σ, σ') of pre- and post-states.

Featherweight OCL as semantic theory is organised as a "shallow embedding." Besides the use of higher-order abstract syntax, this means that types of UML/OCL are represented by types in Isabelle/HOL in an injective way, and that the semantic representation of operators will respect this mapping. For example, logical equality of HOL ($\_ = \_$) coincides to semantic equivalence of OCL; the operations `not` or `_and_` with their OCL type `Boolean -> Boolean` resp. `(Boolean,Boolean) -> Boolean` are represented in Featherweight OCL by `not` :: $\mathfrak{A}$ Boolean $\Rightarrow$ $\mathfrak{A}$ Boolean resp. `_and_` :: $\mathfrak{A}$ Boolean $\Rightarrow$ $\mathfrak{A}$ Boolean $\Rightarrow$ $\mathfrak{A}$ Boolean, where $\mathfrak{A}$ Boolean is a type synonym for a HOL type different from, say, $\mathfrak{A}$ Integer (both introduced in the next paragraph). Thus, Featherweight OCL cannot represent ill-typed OCL expressions, having the consequence that type-related side-conditions can be completely omitted in all derived rules of this language, be it in the OCL library or a given datatype theory, which is vital for their usability in proofs and symbolic executions.

The recent versions of the OMG standard require all OCL types to possess explicit `invalid` and `null` elements, a decision that has major consequences for its logic and data theories. To uniformly represent this phenomenon in Featherweight OCL, we use type classes as in Haskell supported in Isabelle. Parametric polymorphic type variables α can be respectively constrained via type classes $\alpha :: \underline{\text{bot}}$ or $\alpha :: \underline{\text{null}}$ to types containing a bottom element, called bot, and an additional other element (different than bot), called null (where classes are marked by underlining throughout this document). Using the option type written $\_\!\__\perp$ (the None-constructor is written $\perp$ and the Some-constructor $\lfloor\_\_\rfloor$) it is possible to "double lift" types via $(\tau_\perp)_\perp$ and identify $\perp$ with the bot-element of the class, and $\lfloor\perp\rfloor$ with the null-element. Thus, any doubly lifted type is an instance of the type class $\underline{\text{null}}$. Since any OCL expression of type `T` may contain accessors to objects living in a pre and a post state, they represent *valuations* depending from these two states yielding the representation type $\tau_{\mathtt{T}}$ in HOL. This motivates the type synonym:

$$V_{\mathfrak{A}}(\tau_{\mathtt{T}}) \equiv (\mathfrak{A})\,\text{state} \times (\mathfrak{A})\,\text{state} \Rightarrow \tau_{\mathtt{T}} :: \underline{\text{null}}$$

that is used to construct the types for OCL expressions (the precise form of "$(\mathfrak{A})$ state" will be discussed in Section 4.5).

By double-lifting bool and int, which are the standard types from HOL, we declare the following abbreviations:

type_synonym $\quad \text{Boolean}_{\text{base}} := \mathit{bool}_{\perp\perp}$

type_synonym $\quad \text{Integer}_{\text{base}} := \mathit{int}_{\perp\perp}$

As a consequence of these type definitions, we have the elements $\bot$, $\llcorner\bot\lrcorner$, $\sqcup\!\sqcup$True$\sqcup\!\sqcup$, $\sqcup\!\sqcup$False$\sqcup\!\sqcup$ in the carrier-set of $\text{Boolean}_{\text{base}}$. The type $\mathfrak{A}$ Boolean used above is therefore an abbreviation for $V_{\mathfrak{A}}((\text{bool}_{\bot})_{\bot})$, the type $\mathfrak{A}$ Integer stands for $V_{\mathfrak{A}}((\text{int}_{\bot})_{\bot})$:

type_synonym $\quad \text{Boolean}_{\mathfrak{A}} := V_{\mathfrak{A}}(\text{Boolean}_{\text{base}})$

type_synonym $\quad \text{Integer}_{\mathfrak{A}} := V_{\mathfrak{A}}(\text{Integer}_{\text{base}})$

4.2 Denotational Semantics of Constants and Operations

Recall that $\_ = \_ :: \alpha \Rightarrow \alpha \Rightarrow \text{bool}$ is the logical equality of HOL. By using a shallow embedding of OCL in HOL, logical equality becomes then accessible to OCL terms as a mean to express semantic equivalence. If we want to emphasise definitions, we will use $\_ \equiv \_$ for logical equality as alternative notation. As a further notational convenience following common use in mathematical textbooks, we use the notation $I[\![\_]\!]$ mimicking a semantic interpretation function separating concrete syntax of a language to be defined from other constructs defining their semantics. Since a shallow embedding of OCL in HOL is used (higher-order-syntax, operators defined by constant definitions, injective type representation), $I[\![\_]\!]$ is just the identity function. In Isabelle theories, this particular presentation of definitions paves the way for an automatic check that the underlying equation has the form of an *axiomatic definition* and is therefore logically safe.

The generic constants `invalid` and `null` together with the non-strict tests for `invalid` and `null` required by the OMG standard are now defined as follows:

$$I[\![\texttt{invalid} :: V_{\mathfrak{A}}(\alpha :: \underline{\text{bot}})]\!]\tau = \text{bot} :: \alpha$$
$$I[\![\texttt{null} :: V_{\mathfrak{A}}(\alpha :: \underline{\text{null}})]\!]\tau = \text{null} :: \alpha$$

where bot and null are the two elements provided when defining the type classes bot and null. For the concrete `Boolean`-type, we define similarly the boolean constants `true` and `false` as well as the fundamental tests for definedness and validity (generically defined for all types):

$$\begin{aligned}
I[\![\texttt{true} :: \mathfrak{A}\ \text{Boolean}]\!]\tau &= \sqcup\!\sqcup\text{True} :: \text{bool}_{\sqcup\!\sqcup} \\
I[\![\texttt{false} :: \mathfrak{A}\ \text{Boolean}]\!]\tau &= \sqcup\!\sqcup\text{False} :: \text{bool}_{\sqcup\!\sqcup} \\
I[\![X\texttt{.oclIsUndefined()}]\!]\tau &= (\text{if } I[\![X]\!]\tau \in \{\text{bot}, \text{null}\} \text{ then } I[\![\texttt{true}]\!]\tau \\
&\qquad\qquad \text{else } I[\![\texttt{false}]\!]\tau) \\
I[\![X\texttt{.oclIsInvalid()}]\!]\tau &= (\text{if } I[\![X]\!]\tau = \text{bot} \text{ then } I[\![\texttt{true}]\!]\tau \text{ else } I[\![\texttt{false}]\!]\tau)
\end{aligned}$$

On this basis, one can define the core logical operators `not` and `and` as follows:

$$\begin{aligned}
I[\![\texttt{not } X]\!]\tau \quad = (&\text{case } I[\![X]\!]\tau \text{ of} \\
&\bot \quad\ \Rightarrow \bot \\
&|\lfloor\bot\rfloor \quad\ \Rightarrow \lfloor\bot\rfloor \\
&|\lfloor\lfloor x\rfloor\rfloor \quad\ \Rightarrow \lfloor\lfloor\neg x\rfloor\rfloor)
\end{aligned}$$

$$
\begin{array}{lll}
I[\![X \texttt{ and } Y]\!]\tau & = (\text{case } I[\![X]\!]\tau \text{ of} & \\
& \quad \bot & \Rightarrow (\text{case } I[\![Y]\!]\tau \text{ of} \\
& & \quad \bot \Rightarrow \bot \\
& & \quad |\lfloor \bot \rfloor \Rightarrow \bot \\
& & \quad |\lfloor\lfloor \text{True} \rfloor\rfloor \Rightarrow \bot \\
& & \quad |\lfloor\lfloor \text{False} \rfloor\rfloor \Rightarrow \lfloor\lfloor \text{False} \rfloor\rfloor) \\
& \quad |\lfloor \bot \rfloor & \Rightarrow (\text{case } I[\![Y]\!]\tau \text{ of} \\
& & \quad \bot \Rightarrow \bot \\
& & \quad |\lfloor \bot \rfloor \Rightarrow \lfloor \bot \rfloor \\
& & \quad |\lfloor\lfloor \text{True} \rfloor\rfloor \Rightarrow \lfloor \bot \rfloor \\
& & \quad |\lfloor\lfloor \text{False} \rfloor\rfloor \Rightarrow \lfloor\lfloor \text{False} \rfloor\rfloor) \\
& \quad |\lfloor\lfloor \text{True} \rfloor\rfloor & \Rightarrow (\text{case } I[\![Y]\!]\tau \text{ of} \\
& & \quad \bot \Rightarrow \bot \\
& & \quad |\lfloor \bot \rfloor \Rightarrow \lfloor \bot \rfloor \\
& & \quad |\lfloor\lfloor y \rfloor\rfloor \Rightarrow \lfloor\lfloor y \rfloor\rfloor) \\
& \quad |\lfloor\lfloor \text{False} \rfloor\rfloor & \Rightarrow \lfloor\lfloor \text{False} \rfloor\rfloor)
\end{array}
$$

These non-strict operations are used to define the other logical connectives in the usual classical way:

$$
\begin{array}{l}
X \texttt{ or } Y \equiv \texttt{not } ((\texttt{not } X) \texttt{ and } (\texttt{not } Y)) \\
X \texttt{ implies } Y \equiv (\texttt{not } X) \texttt{ or } Y
\end{array}
$$

For reasons of conciseness, we will write $\delta\ X$ for $\texttt{not}(X\texttt{.oclIsUndefined()})$ and $\upsilon\ X$ for $\texttt{not}(X\texttt{.oclIsInvalid()})$ throughout this document.

The default semantics for an OCL library operator is strict semantics; this means that the result of an operation f is `invalid` if one of its arguments is `invalid` or `null`. The definition of the addition for integers as default variant reads as follows:

$$
\begin{array}{ll}
I[\![X \texttt{ + } Y]\!]\tau = & \text{if } I[\![\delta\ X]\!]\tau = I[\![\texttt{true}]\!]\tau \wedge I[\![\delta\ Y]\!]\tau = I[\![\texttt{true}]\!]\tau \\
& \text{then } \lfloor\lfloor\lceil\lceil I[\![X]\!]\tau\rceil\rceil + \lceil\lceil I[\![Y]\!]\tau\rceil\rceil\rfloor\rfloor \\
& \text{else } \bot
\end{array}
$$

where the operator "`+`" on the left-hand side of the equation denotes the OCL addition of type `(Integer,Integer)->Integer` while the "$+$" on the right-hand side of the equation of type $[\text{int}, \text{int}] \Rightarrow \text{int}$ denotes the integer-addition from the HOL library.

4.3 Logical Layer

The topmost goal of the logic for OCL is to define the *validity statement*:

$$(\sigma, \sigma') \models P$$

where σ is the pre-state and σ' the post-state of the underlying system and P is a formula, i. e., an OCL expression of type `Boolean`. Informally, a formula P is valid if and only if its evaluation in (σ, σ') (i. e., τ for short) yields `true`. Formally this means:

$$\tau \models P \equiv (I[\![P]\!]\tau = I[\![\texttt{true}]\!]\tau)$$

On this basis, classical, two-valued inference rules can be established for reasoning over the logical connectives, the different notions of equality, definedness and validity. The core inference rules are:

- `Boolean`:

  $\tau \models \texttt{true} \quad \neg(\tau \models \texttt{false}) \quad \neg(\tau \models \texttt{invalid}) \quad \neg(\tau \models \texttt{null})$

- `not`:

  $\tau \models \texttt{not}\ P \Longrightarrow \neg(\tau \models P)$

- `and`:

  $\tau \models P\ \texttt{and}\ Q \Longrightarrow \tau \models P \qquad \tau \models P\ \texttt{and}\ Q \Longrightarrow \tau \models Q$

- `or`:

  $\tau \models P \Longrightarrow \tau \models P\ \texttt{or}\ Q \qquad \tau \models Q \Longrightarrow \tau \models P\ \texttt{or}\ Q$

- `if`...`then`...`else`...`endif`:

  $\tau \models \quad P \Longrightarrow I[\![\texttt{if}\ P\ \texttt{then}\ B_1\ \texttt{else}\ B_2\ \texttt{endif}]\!]\tau = I[\![B_1]\!]\tau$
  $\tau \models \texttt{not}\ P \Longrightarrow I[\![\texttt{if}\ P\ \texttt{then}\ B_1\ \texttt{else}\ B_2\ \texttt{endif}]\!]\tau = I[\![B_2]\!]\tau$

  or equivalently:

  $\tau \models \quad P \Longrightarrow (\texttt{if}\ P\ \texttt{then}\ B_1\ \texttt{else}\ B_2\ \texttt{endif})\ \tau = B_1\ \tau$
  $\tau \models \texttt{not}\ P \Longrightarrow (\texttt{if}\ P\ \texttt{then}\ B_1\ \texttt{else}\ B_2\ \texttt{endif})\ \tau = B_2\ \tau$

- $\delta\ \_$ and $\upsilon\ \_$:

  $\tau \models P \Longrightarrow \tau \models \delta\ P \qquad \tau \models \delta\ X \Longrightarrow \tau \models \upsilon\ X$

By the latter two properties, it can be inferred that any valid property P (so for example, a valid invariant) is defined, which allows to infer for terms composed by strict operations that their arguments and finally the variables occurring in it are valid or defined.

The mandatory part of the OCL standard refers to an equality (written X `=` Y or X `<>` Y for its negation), which is intended to be a strict operation (thus: `invalid` `=` Y evaluates to `invalid`) and which uses the references of objects in a state when comparing objects, similarly to C++ or Java. In order to avoid confusions, we will use the following notations for equality:

1. The symbol $\_ = \_$ remains to be reserved to the HOL equality, i. e., the equality of our semantic meta-language,
2. The symbol $\_ \triangleq \_$ will be used for the *strong logical equality*, which follows the general logical principle that "equals can be replaced by equals,"[2] and is at the heart of the OCL logic,

[2]Strong logical equality is also referred as "Leibniz"-equality.

3. The symbol $\_ \doteq \_$ is used for the strict referential equality, i. e., the equality the mandatory part of the OCL standard refers to by the "_ = _" symbol.

The strong logical equality is a polymorphic concept which is defined using polymorphism for all OCL types by:

$$I[\![X \triangleq Y]\!]\tau \equiv {}_{\sqcup\!\bot}I[\![X]\!]\tau = I[\![Y]\!]\tau_{\bot\!\sqcup}$$

It enjoys nearly the laws of a congruence:

$$\tau \vDash (X \triangleq X)$$

$$\tau \vDash (X \triangleq Y) \Longrightarrow \tau \vDash (Y \triangleq X)$$

$$\tau \vDash (X \triangleq Y) \Longrightarrow \tau \vDash (Y \triangleq Z) \Longrightarrow \tau \vDash (X \triangleq Z)$$

$$\mathrm{cp}\, P \Longrightarrow \tau \vDash (X \triangleq Y) \Longrightarrow \tau \vDash (P\, X) \Longrightarrow \tau \vDash (P\, Y)$$

where the predicate cp stands for *context-passing*, a property that is true in Featherweight OCL for all pure OCL expressions (but not arbitrary mixtures of OCL and HOL):

$$\mathrm{cp}\, P \equiv \exists f.\, \forall X\ \tau.\ I[\![P\, X]\!]\tau = I[\![f\ (I[\![X]\!]\tau)]\!]\tau$$

The necessary side-calculus for establishing cp can be fully automated; the reader interested in the details is referred to the machine-checked formalization [BTW14].

The strong logical equality of Featherweight OCL gives rise to a number of further rules and derived properties, that clarify the role of strong logical equality and the `Boolean` constants in OCL specifications:

$$\tau \vDash \delta\, X \lor \tau \vDash X \triangleq \texttt{invalid} \lor \tau \vDash X \triangleq \texttt{null}$$

$$\begin{array}{lll}
(\tau \vDash A \triangleq \texttt{invalid}) & = & (\quad \tau \vDash \texttt{not}\ (\upsilon\ A) \qquad\quad) \\
(\tau \vDash A \triangleq \texttt{null}\quad) & = & (\quad \tau \vDash \upsilon\ A\ \texttt{and not}\ (\delta\ A)) \\
(\tau \vDash A \triangleq \texttt{true}\quad) & = & (\quad \tau \vDash A \qquad\qquad\quad) \\
(\tau \vDash A \triangleq \texttt{false}\quad) & = & (\quad \tau \vDash \texttt{not}\ A \qquad\quad) \\
(\tau \vDash \texttt{not}\ (\delta\ X)\quad) & = & (\neg\, \tau \vDash \delta\ X \qquad\quad) \\
(\tau \vDash \texttt{not}\ (\upsilon\ X)\quad) & = & (\neg\, \tau \vDash \upsilon\ X \qquad\quad)
\end{array}$$

Thus with these rules, one can convert an OCL formula represented in its four-valued world into a representation that is classically two-valued, and let the processing with standard SMT solvers such as CVC3 [BT07] or Z3 [dMB08]. δ-closure rules for all logical connectives have the following format (for example):

$$\begin{array}{l}
\tau \vDash \delta\, X \Longrightarrow (\tau \vDash \texttt{not}\ X) = (\neg(\tau \vDash X)) \\
\tau \vDash \delta\, X \Longrightarrow \tau \vDash \delta\, Y \Longrightarrow (\tau \vDash X \quad \texttt{and} \quad Y) = ((\tau \vDash X)\ \land\ (\tau \vDash Y)) \\
\tau \vDash \delta\, X \Longrightarrow \tau \vDash \delta\, Y \Longrightarrow (\tau \vDash X\ \texttt{implies}\ Y) = ((\tau \vDash X) \longrightarrow (\tau \vDash Y))
\end{array}$$

With the conjunction of these rules (comprising the above mentioned case distinction: $\tau \vDash \delta\, X \lor \tau \vDash X \triangleq \texttt{invalid} \lor \tau \vDash X \triangleq \texttt{null}$), we can automatically proceed to the simplification of a formula by case analysis, in order to quickly reach a contradiction, whenever we know that a variable X is `invalid` or `null`. For example, we can infer from an invariant $\tau \vDash X \doteq Y$ `- 3` that we have

$\tau \models X \doteq Y$ `- 3` $\wedge\ \tau \models \delta\, X \wedge \tau \models \delta\, Y$. We call the latter formula the δ-closure of the former. Now, we can convert a formula like $\tau \models X$ `> 0 or 3 *` Y `>` X `*` X into the equivalent formula $\tau \models X$ `> 0` $\vee\ \tau \models$ `3 *` Y `>` X `*` X and thus internalize the four-valued logic of OCL, as if we have a classical (and more tool-conform) logic.

4.4 Algebraic Layer

Based on the logical layer, we build a system with simpler rules which are amenable to automated reasoning. We restrict ourselves to pure equations on OCL expressions.

Our denotational definitions on `not` and `and` can be re-formulated in the following ground equations:

- υ _:

  υ `invalid` = `false` $\qquad$ υ `null` = `true`
  υ `true` = `true` $\qquad$ υ `false` = `true`

- δ _:

  δ `invalid` = `false` $\qquad$ δ `null` = `false`
  δ `true` = `true` $\qquad$ δ `false` = `true`

- `not`:

  `not invalid` = `invalid` $\qquad$ `not null` = `null`
  `not true` = `false` $\qquad$ `not false` = `true`

- `and`:

  - `invalid`:

    (`invalid and true`) = `invalid` $\qquad$ (`invalid and false`) = `false`
    (`invalid and null`) = `invalid` $\qquad$ (`invalid and invalid`) = `invalid`

  - `null`:

    (`null and true`) = `null` $\qquad$ (`null and false`) = `false`
    (`null and null`) = `null` $\qquad$ (`null and invalid`) = `invalid`

  - `true`:

    (`true and true`) = `true` $\qquad$ (`true and false`) = `false`
    (`true and null`) = `null` $\qquad$ (`true and invalid`) = `invalid`

  - `false`:

    (`false and true`) = `false` $\qquad$ (`false and false`) = `false`
    (`false and null`) = `false` $\qquad$ (`false and invalid`) = `false`

On this core, the structure of a conventional lattice arises:

$$X \texttt{ and } X = X$$
$$X \texttt{ and } Y = Y \texttt{ and } X$$
$$X \texttt{ and } (Y \texttt{ and } Z) = X \texttt{ and } Y \texttt{ and } Z$$
$$\texttt{false and } X = \texttt{false} \qquad X \texttt{ and false} = \texttt{false}$$
$$\texttt{true and } X = X \qquad X \texttt{ and true} = X$$

as well as the dual equalities for _ `or` _ and the De Morgan rules. This wealth of algebraic properties makes the understanding of the logic easier, and enables automated analysis: for example, by computing the DNF of some invariant systems (by term-rewriting techniques) which are a prerequisite for δ-closures.

The above equations explain the behaviour for the most important non-strict operations. The clarification of the exceptional behaviours is of key importance for a semantic definition of the standard and the major deviation point from HOL-OCL [BW08a, BW06] to HOL-OCL 2.0 as presented here.

4.5 States Layer and Well Formed States

As detailed in Section 4.1, all OCL operations discussed so far represent special valuations $V_{\mathfrak{A}}(\_)$ depending from a pair of pre state and post state, both of the form "$(\mathfrak{A})$ state". As a first approximation, a state can be thought of as a polymorphic array, where the polymorphic value $\mathfrak{A}$ represents the place where an object (of type $\mathfrak{A}$) can be dynamically stored. The index of the array is the object identifiers: we assume an enumerable type for object identifiers "oid" used for defining states (where the type oid is an abbreviation of the type nat representing HOL natural numbers). Since a UML/OCL state consists of a partial map of oids to object representations and a representation of the associations, it is natural to model it with the command record [Wen16b]:

$$\begin{array}{lll} \text{record } (\mathfrak{A})\,\text{state} = & \text{heap} & :: \text{oid} \rightharpoonup \mathfrak{A} \\ & \text{assocs} & :: \text{oid} \rightharpoonup \text{oid list list list} \end{array}$$

Moreover, we can join an inverse operation "OidOf :: $\mathfrak{A} \Rightarrow$ oid" to retrieve the oid of an object, but the function OidOf particularly depends on $\mathfrak{A}$, which explicit form will be discussed in the next section.

However, we will require *well-formed states* (WFF), where all oids in all assocs are actually contained in the domain of the heap and furthermore the oids stored in object representations are actually their references in the memory, i. e. that there is a "one-to-one" correspondence between object representations and oids:

$$\begin{aligned} \text{definition WFF } \tau = {} & \forall obj \in \text{ran}(\text{heap}(\text{fst } \tau)).\ \lceil \text{heap}(\text{fst } \tau)(\text{OidOf } obj) \rceil = obj \\ & \wedge \forall obj \in \text{ran}(\text{heap}(\text{snd } \tau)).\ \lceil \text{heap}(\text{snd } \tau)(\text{OidOf } obj) \rceil = obj \end{aligned}$$

This condition is also mentioned in the OMG's specifications [Obj12, Annex A] and goes back to Richters [Ric02]; however, we state this condition as a constraint on states for some logical rules rather than a global axiom. It can, therefore, not be taken for granted that an oid makes sense both in pre- and post-states of OCL expressions.

As a polymorphic concept, the strong logical equality $\_ \triangleq \_$ does not have to be redefine again. This relation also applies on objects, so two objects are equal if their denotations are semantically equal. We formally proved that within well-formed states and for valid objects, the referential equality $\_ \doteq \_$ coincides with strong logical equality [BTW14]. This justifies that the former can be used for the latter for efficiency reasons.

4.6 A Denotational Space for Class Models: The Naïve Attempt

We turn now to the issue of giving a more detailed semantics for a class model. The theory of states can be developed generically once and for all; however, the key point is that we need a common type $\mathfrak{A}$ for the set of all possible *object representations*. Object representations model "a piece of typed memory," i. e., a kind of record comprising administration information and the information for all attributes of an object; here, the primitive types as well as collections over them are stored directly in the object representations, class types and collections over them are represented by oid's (respectively lifted collections over them). So a (typed) universe of object representations which will be a concrete instance of the type variable $\mathfrak{A}$ has to be constructed for a concrete class model.

In a shallow embedding which must represent UML types one-to-one by HOL types, there are two fundamentally different ways to construct such a set of object representations, which we call an *object universe* $\mathfrak{A}$:

1. an object universe can be constructed from a given class model, leading to *closed world semantics*, and
2. an object universe can be constructed for a given class model *and all its extensions by new classes added into the leaves of the class hierarchy*, leading to an *open world semantics*.

For the sake of simplicity, the present semantics of HOL-OCL 2.0 chose the first option, while HOL-OCL [BW08b] used an involved construction allowing the latter.

A naïve attempt to construct $\mathfrak{A}$ would look like this: the class type C_i induced by a class will be the type of such an object representation: $C_i := (\text{oid} \times A_{i_1} \times \cdots \times A_{i_k})$ where the types $A_{i_1}, \ldots, A_{i_k}$ are the attribute types (including inherited attributes) with class types substituted by oid. The function OidOf projects the first component, the oid, out of an object representation. Then the object universe will be constructed by the type definition:

$$\mathfrak{A} := C_1 + \cdots + C_n \,.$$

It is possible to define constructors, accessors, and the referential equality on this object universe. However, the treatment of type casts and type tests cannot be faithful with common object-oriented semantics, be it in UML or Java: casting up along the class hierarchy can only be implemented by loosing information, such that casting up and casting down will *not* give the required identity. Whenever $C_k < C_i$ and X is valid, we would like to obtain instead:

$$X.\texttt{oclIsTypeOf}(C_k) \;\; \texttt{implies} \;\; X.\texttt{oclAsType}(C_i).\texttt{oclAsType}(C_k) \doteq X$$

To overcome this limitation, we need to slightly revise how class types are fundamentally built. However, instead of providing at present the solution, we will do it later in Chapter 7. This is because in any case *both* the naïve construction presented here and our new solution can nevertheless *not* be done directly in HOL: both constructions involve quantifications and iterations over the "set of types". Rather, a *meta-level* construction is needed.

Such meta-level construction is required for building the object-oriented *datatype theory*. Like for a datatype package in other HOL-systems, the semantics for class models can be given by a datatype theory, i. e., a conservative theory extension consisting of a number of conservative definitions for accessor-, cast- and type-tests, and automated tactic proofs establishing a number of rules for these operations. Besides the usual laws on casting and dynamic typing, these operations are designed to reflect the strictness principles with respect to `null` and `invalid`. To provide an infra-structure for these definitions, the generic "meta-tool" provided in Chapter 5 will not only perform the necessary meta-level constructions required to setup the object universe $\mathfrak{A}$, but also generally any constructions requiring to reach a suitable meta-level of expressivity (irrespective of languages being embedded into HOL).

Finally, equipped with this meta-tool, we will recover our semantical investigation on class types in Chapter 7, with together the resulting properties of object navigation.

4.7 A Comparison to Related Work

There is a large variety of implementations that use a fragment of the OCL syntax and compile it together with some extensions (temporal logic, dynamic-logic...) more or less directly to some tool (Maude, ASM, KodKod, Prolog...); it is characteristic for these approaches that a direct, efficient reuse of existing tools and the possibility to experiment with class models is a more important concern than compliance to the OCL standard.

In this thesis, we address UML/OCL in the sense of the discussion in the OCL group and major compiler implementations [Dre16, Pap16] which drive the OCL standard's evolution and to which this work contributes a formalisation of the forthcoming OCL 2.5 proposal [BTW14]. Besides compilers, there is a number of great animation tools, mostly based on older 3-valued versions of the UML/OCL standard, USE, Kodkod or OCLexec [RG02, KG12, KK08] just to cite a few. The present work, however, attempts to provide foundations for *deductive* methods, be it for symbolic evaluation methods necessary for test case generation or verification methods based on interactive or automated proof.

Restricting us to the category of more or less standard compliant, deduction oriented methods, we see HOL-OCL [BW08b], which is also based on Isabelle/HOL and with which consequently our work has a lot in common; however, besides technical differences in the front-end, HOL-OCL uses three-valued logic and a simpler data model for associations, which are compiled to aggregations. On the other hand, the object universe construction of HOL-OCL uses an involved construction representing "holes" in the universe by polymorphic variables, thus leveraging a kind of modular "open-world" semantics; our approach remains in the simpler "closed-world" interpretation of class models. Avoiding these

```
int seats;
/* @ global invariant A:
              0 < seats; */
```

Listing 4.1: ACSL

```
class
    FLIGHT
feature
    seats: INTEGER
invariant
    0 < seats
end
```

Listing 4.2: Eiffel

```
public class Flight {
  int /*@spec_public@*/ seats;
  /*@invariant 0 < seats; @*/
}
```

Listing 4.3: JML

```
public class Flight {
  public int seats;
  invariant 0 < seats;
}
```

Listing 4.4: Spec#

Figure 4.1: The `Flight` class with the invariants on seats in various languages

type variables dramatically improves the usability in practical proofs. Tools like ESC/Java2 for JML or Boogie for Spec# also follow a closed-world approach.

The OCL2FOL$^+$-project [DC13, ADEM14] is to our knowledge the first deduction-based tool that uses the same logics as used in our work, but targets via a clever compilation an SMT-solver and enables automated deduction tools for UML/OCL in the security domain; this approach provides first evidence that tackling with a standard-conform semantics is indeed feasible and promising. The fear that the use of a multi-valued logics results in inherent efficiency problems is at least not justifiable on theoretic grounds [Hӧ4]. They generate an axiomatisation of the object-oriented datatype theories, while we automatically derive them from a denotational model to ensure logical consistency—thus, our work is a semantic foundation for their approach in this respect.

In the following, we focus on deductive verification approaches for object-oriented data theories in a wider sense.

Eiffel

Eiffel [Mey97] pioneered the idea of *class invariants* adapting this concept going back to Dijkstra, Floyd, and Hoare in the 1960's to object-oriented languages, and popularised the idea of pre- and post-conditions to a "design-by-contract" methodology. Eiffel is a remarkable exception to the other languages, as the contract language was part of its design right from the beginning. The contract specifications are part of the Eiffel language specification and are supported by all Eiffel development tools. The Eiffel specification language is a two-valued logic that provides an explicit definedness (non-null) test: `_ /= Void`. This kind of test needs to be stated explicitly to ensure that no void references are accessed (while OCL can handle this implicitly, see previous section). Moreover, Eiffel requires exceptions to be handled explicitly, i. e., well-defined behaviour in case an exception is thrown. OCL handles this implicitly, but for the only exception `invalid`.

In the following, we will briefly introduce other, rather widely used, contract or behavioural interface specification languages. Figure 4.1 introduces them

with a very simple example: specifying the invariant from our example that the number of seats is positive.

JML

The Java Modeling Language (JML) [LPC+13] is a constraint language for Java which is, for example, supported by the ESC/Java2 tool [LNS00], which allows for both runtime checking of assertions and static verification.

The logic of JML is two-valued. As in Eiffel, exceptions are explicitly modelled and declared and the language provides an explicit definedness test. The actual burden of writing definedness tests is reduced significantly by making non-null types the default. Only types that are explicitly declared as "nullable" need to be checked for definedness.

Spec#

Spec# [BLS05] is a constraint language for C# that, for example, is supported by the program verification environment Boogie [BLS05].

Overall, Spec# is very similar to JML. The main difference is that non-null types are not the default, but supported by a type inference. The logic is, again, two-valued and exceptions need to be modelled explicitly.

ACSL and VCC

The ANSI/ISO C Specification Language (ACSL) [BCF+13] and VCC [BLW08] are interface specification languages for C that are supported by Frama-C [BCF+13] and Visual Studio [CDH+09]. Users can write assertions, data invariants, and behavioural contracts over C programs.

The logics of ACSL and VCC are two-valued; via particular predicates, regions of valid memory have to be specified explicitly in contracts to ensure that no invalid references are accessed. As conversions to byte-level representations of memory are possible, data invariants are particularly tricky to formulate—a complication necessary to verify machine-level C code. As C does not support exceptions, ACSL and VCC do not either. The less abstract memory model does not include inheritance and subtyping. Later versions of VCC also support a refined concept of memory ownership that allows for verifying concurrent C programs [CDH+09], whereas OCL is strictly sequential (methods are atomic actions).

CHAPTER 5

The Object-Logic Theory Generator

Reproduction, as a terminology, has been firstly employed (to our knowledge) by Klaus Aehlig and Felix Joachimski to characterize the idempotence property that certain λ-terms are exhibiting: when converting them back and forth, between their initial syntactic representation to another representation qualified as "semantic", then back to their previous syntactic representation [AJ04]. These particular endomorphic conversions are feasible because the semantic denotation of programs is expressed with the help of functions, and functions naturally appear in λ-terms. The idea of exploiting this technique to compute the normal form of λ-terms (if such normal form exists) is called *normalization by evaluation* and has been deeply investigated both theoretically and practically over years [ML75, BS91] (the reader interested in the details is referred to some lecture notes [Dan98, DF00]).

In the present work, the term reproduction will still be related with the notion of "some semantics to be preserved", but we will use it slightly differently: by thinking about a partial ordering, instead of an idempotence property for example. For the moment, reproduction can be simply understood as copy or duplication, the implying interplay between syntax and semantics will be made further clear along the document. More precisely, this chapter is focusing on the reproduction of particular λ-terms, namely *editing sessions* ("duplication of editing sessions"). This concerns the ability of the Isabelle framework to write an embedding function supporting a language $\mathcal{L}$ inside this framework, and the ability of the framework to immediately provide means to edit in $\mathcal{L}$ afterwards. In Isabelle, we characterize this ability as *dynamic* because the overall reproduction is performed without leaving the editing session of the one used to write the embedding. A technical cloning illusion will happen when crossing the ML layer (being at the foundation of the Isabelle system), but the overall approach can nevertheless be considered as part of the Isabelle framework. As a reproductive process, one has at the end the possibility to edit in Isar_HOL or $\mathcal{L}$ at the same time. So previous embedded languages can be utilized to embed one next language $\mathcal{L}'$ using the same process, with the so-augmented capacities of the underlying editor, and inheriting from the existing theorem proving infrastructure.

The objective of this chapter is to detail the key components implementing the reproduction process, so that one can ultimately alternate between $\mathcal{L}$ and

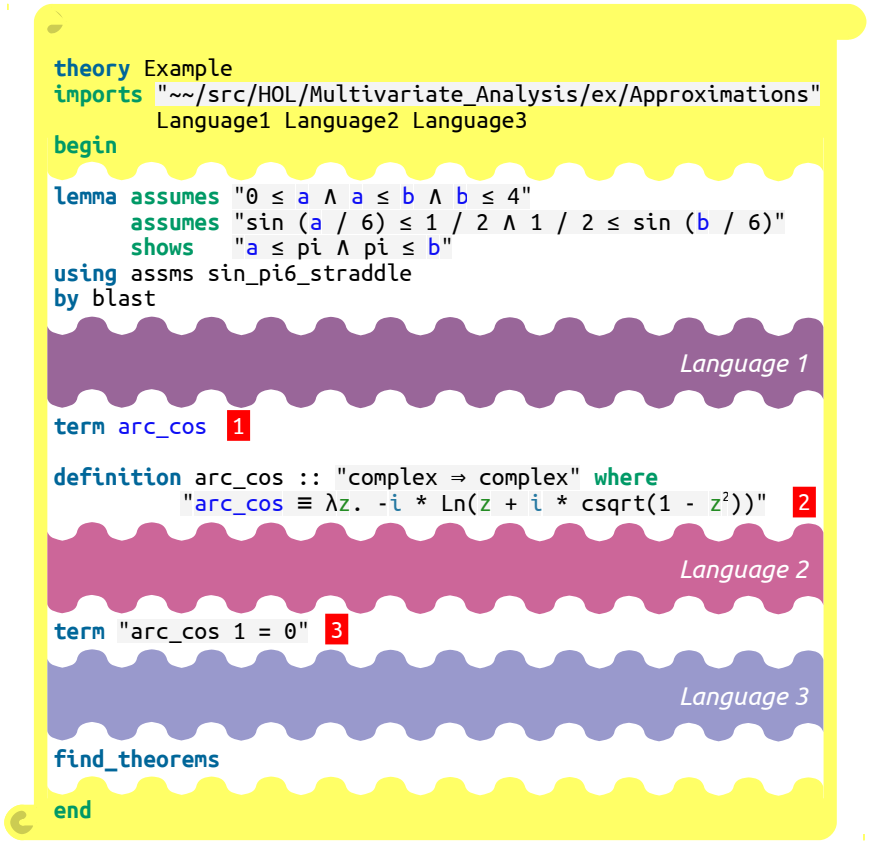


Figure 5.1: One editing window of Isabelle/jEdit after loading a theory file

$\mathcal{L}'$ at any positions inside an editing window of Isabelle, in order to better experiment formal methods activities in $\mathcal{L}$ or $\mathcal{L}'$.

5.1 Isar_HOL as First Language (if not Meta)

The Isabelle framework integrates an optimized environment for the development of specifications and proofs. The environment is initially configured to be edited by default in the Isar_HOL language, because Isar is specialized to support the writing of tactic methods for resolving proofs, and HOL comes with a rich library of mathematic operations. Figure 5.1 presents a window of a running Isabelle session (e. g., in Isabelle 2015). Normally the background is completely white, the color yellow and three sine waves have been added here just for this presentation. User-interaction to Isabelle is *document oriented*, i. e. each file belonging to a session is annotated by the prover while editing it as usual like in any modern IDE. Annotations can consist, for example, in:

- colors (the underlying white indicates that Isabelle checked these commands and executed them without error),
- types (to be explored by tool-tips via the hovering gesture),

- or values associated to computations inside these commands (displayed in a separate "output window" when pointing to them, the output window will have a certain role to play in Chapter 6).

Although collaborative editions are asynchronously supported [Wen14, RL14], we have depicted the position of three red cursors just for the example. In Isabelle/jEdit there is (by default) only one cursor with no number inside. On the other hand, the grey color appearing around quoted terms (like "$a \leq pi \wedge pi \leq b$") is automatically added by Isabelle/jEdit, its purpose is to highlight HOL content from the environment where the user usually poses theorems and proofs irrespective of a particular logic. Indeed, as a *logical* framework, Isabelle offers a small logical core-engine that can be reused by a variety of logics [Wen16b] such as first-order logics (FOL), constructive-type theory (CTT) and Church's higher-order logics (HOL), which is also the basis of this work. Consequently the framework can be globally seen as a kind of *meta-proving environment*, where all terms are specially belonging to a particular logical language. In addition, some facilities are also provided to lighten various aspects of terms from the underlying logic. For instance, at the position of the cursor 1, the term *arc_cos* is coloured in blue meaning that it is a free variable, whereas starting from position 2, it appears in black (like *Ln* and *csqrt*) since it has meanwhile been defined and accepted as a definition or function (and in this context an "HOL function"). Additionally, the variable z is in green as being bound inside the definition; on the other hand the letter i is just a syntactic abbreviation, characterized by its light blue color. Finally the definition of *arc_cos* depends on some libraries related with multivariate analysis, again in HOL, as made precise in the header.

At the end, the last command find_theorems displays various information about theorems, for example the number of currently proved theorems at the precise position where this command is written. Various refinement criteria allow furthermore to fine grain control the searching engine, for instance to discard or explicitly filter particular patterns in the names of theorems. In Isabelle/jEdit, the associated display where the result of these informations are shown is usually located in a separate sub-window, the output window (not represented in the picture), whose purpose is to inform about the state of the current proving environment or to guide users in real time with informative messages depending on the position of the cursor. Indeed as a read-only buffer, refreshment of the output window is automatically triggered (by default) as soon as the cursor is moving from one command to another one in the editing window.[1] Globally, theorems shown by find_theorems do not directly mention themselves if they have been proved in HOL or in other object-logics. However, a theory file can more generally be seen in Isabelle as a particular container embedding multiple languages, and we will particularly take advantage of its flexibility to support new (specification) languages. Hence in the example, we are showing four languages: "Language 1", "Language 2", "Language 3", and a set of commands part of the more general language "Isar_HOL", represented by the white sine wave (and as side remark also comprising the yellow frame).

[1] One can also open several instances of output windows to keep the results of different cursor positions actively displayed. This is performed by manually deactivating the automatic refreshment option in each instance.

Instead of starting from null, lines of evolution have motivated us to imagine the birth of the *formal method tool* presented in this thesis, as a reproductive process, where not only has the tool been built inside the proving system Isabelle, but also brought up as a particular extension of this system, then inheriting the deductive capacities of the framework and its editing environment, thus the name *meta-tool*. More precisely, we use the type "$\mathcal{L} \Rightarrow$ Isar_HOL" to represent the process by which one can extend Isabelle to support a new language $\mathcal{L}$. For the moment, this function can be thought of as a shallow embedding from $\mathcal{L}$ to Isar_HOL. Then, the methodology to support the programming and proving activity in $\mathcal{L}$ in the framework can basically be summarized as follows:

- provided an arbitrary sentence in $\mathcal{L}$,
- it suffices to compute the result of $\mathcal{L} \Rightarrow$ Isar_HOL (this function is called translation function or embedding function),
- to obtain at the end a piece of code written in Isar_HOL to be natively processed by Isabelle (representing the initial sentence in $\mathcal{L}$).

On the one hand, one particularity of this work is that we are mainly emphasizing the notion of *object-logic theory*: during the embedding, any newly language $\mathcal{L}$ becomes understood as an object-logic, coming with a theory, a set of definitions and proved theorems (due to the wide range of expressions that can be represented in the output Isar_HOL). On the other hand, following the idea of practising formal methods with many languages, we are now going to generalize the above methodology and reason with a family of languages $\mathcal{L}_1 \cdots \mathcal{L}_n$. Similarly as $\mathcal{L}$, this family represents a set of functions acting as extensions on the Isabelle framework, so they are of the form $\mathcal{L}_1 \Rightarrow$ Isar_HOL, ..., $\mathcal{L}_n \Rightarrow$ Isar_HOL. However, we also include the possibility to extend one language from another language: $\mathcal{L}_i \Rightarrow \mathcal{L}_j$ (for any i and j, equal or not), this is particularly useful if for example $\mathcal{L}_j \Rightarrow$ Isar_HOL has already been defined.

Figure 5.2 pictures a sequential chain of embeddings without firstly focusing on how the reproductive processes behind the grey arrows are linking the whole chain of embeddings. The presented file is divided into four parts, representing the incremental evolution of the editing activity growing from the top to the bottom. Instead of drawing rectangular cursors like in Figure 5.1, here the notion of *sessions* is particularly emphasized this time by using three sine waves in pink to separate the four blocks of code. Internally a session is represented by some purely functional data-structure describing the state of the editing environment (comprising logical definitions, proofs, text documentation, etc). As a first approximation, a session can be thought of as a list containing all encountered Isar_HOL commands until the actual position of the cursor, with the addition that the semantical consistency of sessions is moreover guaranteed: by checking that all commands are well-typed before their adding to the list. Thus sessions are heavily varying during the editing activity: if the cursor is moving up, the list behind the session will be adjusted accordingly by removing some (well-typed) elements, and if the cursor is moving down, previous deleted elements will be added back. So this allows for example commands like find_theorems to return a consistent result depending on any positions where the cursor could be, no matter where it is called on the file. As remark, to be precise, the session $S2$ should normally extend $S1$, similarly for $S3$ which should extend $S2$ and $S1$,

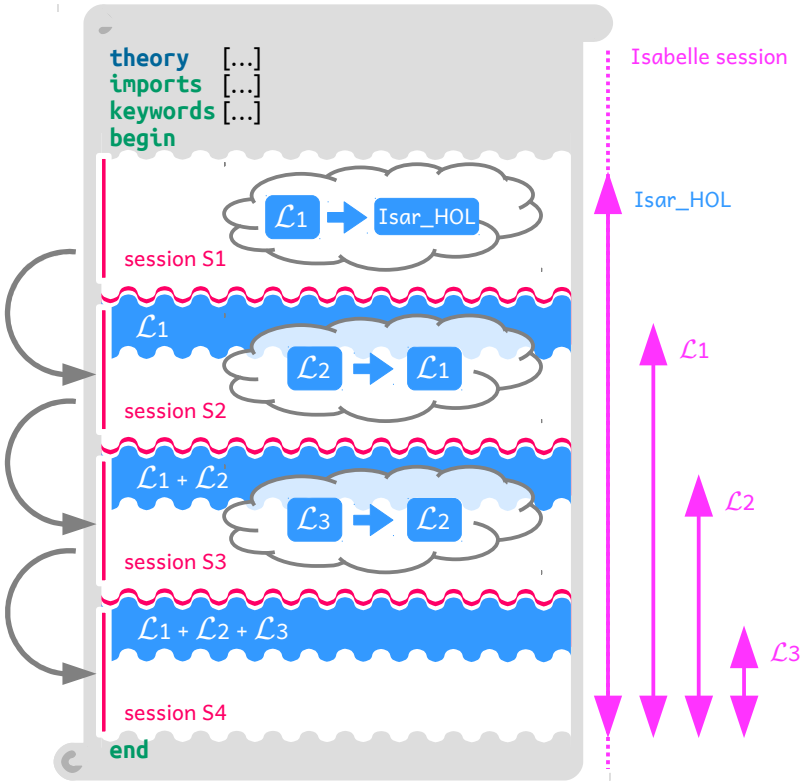


Figure 5.2: The evolution of the reproduction process (sequential embedding)

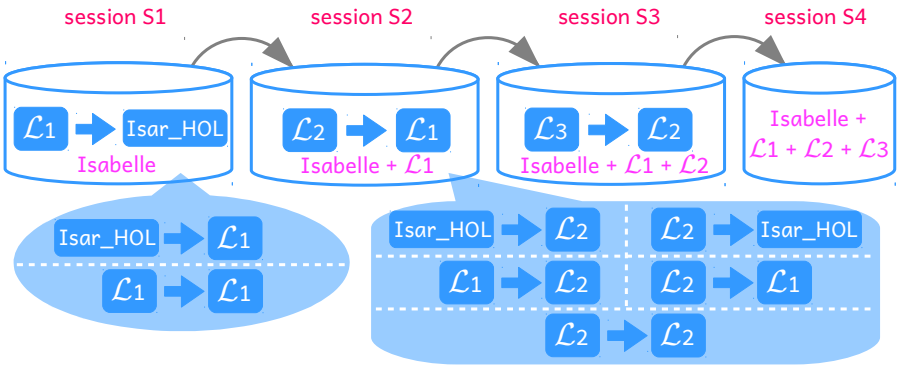


Figure 5.3: The evolution of the reproduction process (sequential embedding)

etc. However by abuse of language, instead of using four continuous vertical lines starting at the same point on the left, we have four lines following each other sequentially.

In Figure 5.2, after embedding $\mathcal{L}_1$ in session $S1$, i. e. after defining a translation process from $\mathcal{L}_1$ to Isar_HOL, it becomes possible starting from $S2$ to program in $\mathcal{L}_1$. The four vertical arrows on the right mention the possibility or not to program in a particular language, so Isar_HOL can (at least) be used from $S1$ to $S4$, $\mathcal{L}_1$ is supported from $S2$ to $S4$, $\mathcal{L}_2$ is supported from $S3$ to $S4$, etc. Consequently when defining the semantics of $\mathcal{L}_2$ in $S2$, this semantics can actually be written in either $\mathcal{L}_1$ or Isar_HOL. Similarly, when defining the semantics of $\mathcal{L}_3$ in $S3$, this semantics can actually be written in either $\mathcal{L}_2$, $\mathcal{L}_1$ or Isar_HOL.

More abstractly, we use cylinders in Figure 5.3 to emphasize that sessions are part of the dynamic editing activity which occurs in RAM memory. Moreover, we generalize the way how a new language can be embedded to some language parent, by considering the graph induced by the inverse relation of $\_ \Rightarrow \_$. At each session S_n, the embedding of $\mathcal{L}_n$ in Isar_HOL can be performed by naively checking if for all nodes $m \leq n$, there is a path strongly connecting Isar_HOL to $\mathcal{L}_m$. So for instance in session S_2, any combinations of the form $\mathcal{L}_i \Rightarrow \mathcal{L}_j$ are possible as long as the strongly connecting condition is respected. That includes all $\mathcal{L}_i \Rightarrow \mathcal{L}_j$ involving at least $\mathcal{L}_2$ in $\mathcal{L}_i$ or $\mathcal{L}_j$ as represented in the box. Additionally, all relations coming from $S1$ can also be inherited to connect any paths in $S2$, such as $\mathcal{L}_1 \Rightarrow \mathcal{L}_1$ and Isar_HOL $\Rightarrow \mathcal{L}_1$. Again, while not mentioned, the relation Isar_HOL $\Rightarrow$ Isar_HOL can also appear to connect any paths. In the same spirit, the embedding of $\mathcal{L}_3$ can rely on any combinations of $\mathcal{L}_1$, $\mathcal{L}_2$ and Isar_HOL. Similarly for S_4, the editor is ready to consider the embedding of a possibly new language $\mathcal{L}_4$, or continue as usual the theorem proving activity with all or any combinations of $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$ and Isar_HOL.

As remark, the presented embedding has been defined sequentially, in the sense that several grey arrows were involved one after another one. A similar result can be obtained with another style of embedding, depicted in Figure 5.4, which is more compact as it treats simultaneously the embedding of all $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ into Isar_HOL as a particular "sum type". In this case only one grey arrow occurs. This is particularly relevant if the grey arrow has a certain cost that cumulative executions would avoid. Both programming styles are nevertheless equivalent: any combinations of the form $\mathcal{L}_i \Rightarrow \mathcal{L}_j$ are possible to be defined in both cases (as long as the above naive strongly connecting condition is fulfilled). Otherwise said, we finally obtain at the end a similar session S_4 as in the sequential reproductive process.

The next sections are now devoted to reveal in more detail the implementation of the reproduction process behind the grey arrows and to present as well how to define the embedding functions.

5.2 Readability and Efficiency in Package Management

As observed in Figure 5.1, the management of dependencies among theories is completely carried out by writing the specific list of theories to import when starting a theory document (with the keyword imports). By maximizing the list of theories to import, one is typically tuning how parallel the Isabelle system is

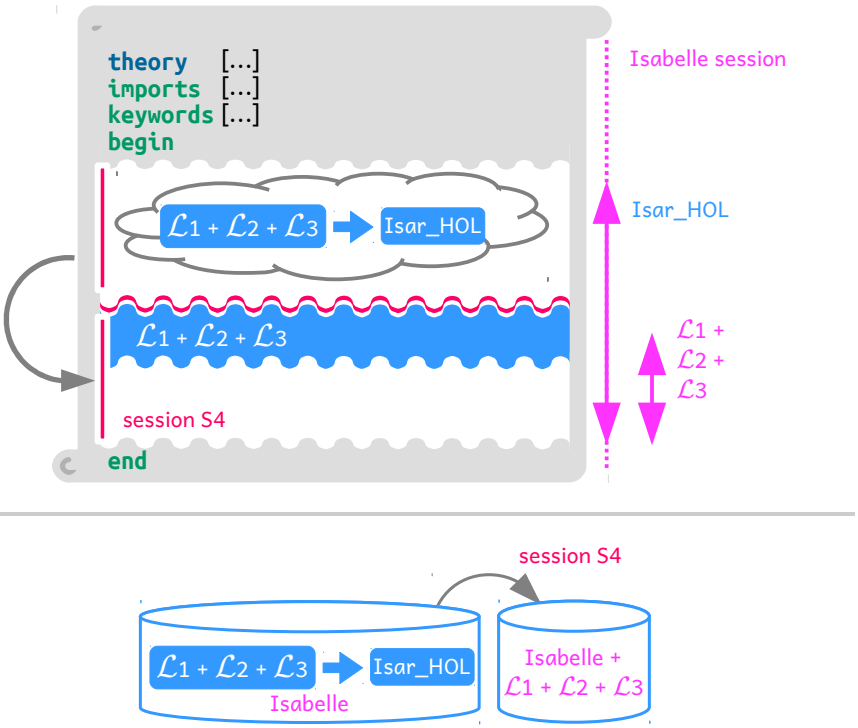


Figure 5.4: The evolution of the reproduction process (simultaneous embedding)

going to process the overall collection of documents. Obviously, a system with multi-core processors are better exploited when treating several unrelated theories in parallel.[2] On the other hand, separation of concerns generally contributes to reduce the effort of building a complex system or algorithm, by dividing a non-trivial task on smaller components easier to test and prove for example. In the present work, we will assimilate such components to packages [Mel91].

A package comes with a sequence of commands. Since commands are all defined in the respective Isabelle theory documents being imported[3], without loss of generality, we can approximate a package as a theory defining at least one command. So if one user misses to import the appropriate theory, errors naturally appear in front of all encountered unknown commands as usual. For the particular case of a theorem proving system like Isabelle, (most) commands in packages have all the more the property to generate a number of theorems. For example, whereas lemma usually produces one theorem, after writing datatype α LIST = NIL | CONS "α" "α LIST", we obtain 94 newly generated theorems in Isabelle 2016. However in Isabelle, commands are serving diverse purposes, for example:

- *(HOL item)* besides the possibility to generate theorems,

[2] In Isabelle, theories are forming a directed acyclic graph. [Wen16b]

[3] After bootstrapping Isabelle, it is always assumed that a theory imports at least one other theory.

- *(Isar item)* commands also appear used during the proof of a theorem, since they serve to instruct how far to advance a particular proof with specialized tactics. (In Figure 5.1, the command using advances to the middle of the proof, then the command by concludes the proof.)

The conjunction of these two facts suggests us to observe that the development of Isabelle packages to support a domain-specific language $\mathcal{L}$ can somehow be made generic by considering the whole type Isar + HOL (where _ + _ represents the sum type similar as Figure 5.4). More precisely, we estimate the function $\mathcal{L} \Rightarrow$ Isar_HOL enough abstract for covering at the same time:

- packaging functions of the form $\mathcal{L} \Rightarrow$ HOL for commands generating theorems, and
- packaging functions of the form $\mathcal{L} \Rightarrow$ Isar for commands solving proofs.

Thus "developing an $\mathcal{L}$-package" amounts to define a function of the form $\mathcal{L} \Rightarrow$ Isar_HOL. However in the present work,

- *(HOL item)* certain singular features of UML/OCL have motivated us to further generalize the concept of Isabelle packages (among others, the support of multiplicity outranging the expressivity scope of HOL, the incremental encoding of classes, etc., Chapter 6 will give further details), and
- *(Isar item)* the vast range of normalizing techniques (like normalization by evaluation) has incited us to determine how well UML/OCL formulae could be efficiently and automatically discharged in Isar proofs, for instance with automated theorem proving techniques like decision procedures (which are elaborated tactics, able to recognize theorems from a decidable theory).

So in order to uniformly satisfy both constraints, we are now asking if there could exist a "universal framework" unifying both at the same time the practices of developing packages on the one side (where the reasoning logic can be made arbitrarily large), and developing decision procedures on the other side.

At first sight, developing an $\mathcal{L}$-package could seem to be more general than developing a decision procedure for a particular logic, for example Presburger arithmetic (PA). This is because in decision procedure one has to write, at least, a function of type $fm \Rightarrow fm$ for a particular type fm representing formulas (e. g., PA). Then, provided a complex expression of type fm, the principle is to simplify it and obtain at the end an equal expression: a certain "normal form" easier to reason with (like in normalization by evaluation).

However, even if packages are usually presented as embedding functions of the form $A \Rightarrow B$, nothing prevents to introduce instead a slightly general type "$(A + B) \Rightarrow (A + B)$" (in reality only an expression of type A will be provided in input, and for the moment we only expect to obtain an expression in B). For the case of HOL-OCL 2.0 packages, "A = abstract syntax of UML/OCL", and "B = abstract syntax of Isar_HOL" the set of Isar_HOL definitions and lemmas automatically derived in output by the HOL-OCL 2.0 packages. The resulting objective is then to support (UML/OCL + Isar_HOL) $\Rightarrow$ (UML/OCL + Isar_HOL). Thus, similarly as a decision procedure, one can consider UML/OCL as a formal logical system, and as well Isar_HOL as a kind of superlogic (where a definition of superlogic can be found for example in the work of David A. Basin, Manuel

$\tau \vDash X.\texttt{oclAsType}(C_i) \triangleq X$

$\tau \vDash \texttt{invalid .oclAsType}(C_i) \triangleq \texttt{invalid}$

$\tau \vDash \texttt{null .oclAsType}(C_i) \triangleq \texttt{null}$

$\tau \vDash ((X :: C_i).\texttt{oclAsType}(C_j) \ .\texttt{oclAsType}(C_i) \triangleq X)$

$\tau \vDash (X :: \texttt{OclAny}).\texttt{oclAsType(OclAny)} \triangleq X$

$\tau \vDash \delta\, X \Longrightarrow \tau \vDash X.\texttt{oclAsType}(C_j) \ .\texttt{oclAsType}(C_i) \triangleq X$

$\tau \vDash X.\texttt{oclIsTypeOf}(C_j) \Longrightarrow \tau \vDash \delta\, X \Longrightarrow \tau \vDash \texttt{not}(\upsilon\, X.\texttt{oclAsType}(C_i))$

$\tau \vDash \texttt{invalid .oclIsTypeOf}(C_i) \triangleq \texttt{invalid}$

$\tau \vDash \texttt{null .oclIsTypeOf}(C_i) \triangleq \texttt{true}$

$\tau \vDash (X :: C_i).\texttt{oclIsTypeOf}(C_j) \Longrightarrow \tau \vDash (X :: C_i).\texttt{oclIsKindOf}(C_i)$

$(\tau \vDash (X :: C_j) \doteq X) = (\tau \vDash \texttt{if}\ \upsilon\, X\ \texttt{then true else invalid endif})$

$\tau \vDash (X :: C_j) \doteq Y \Longrightarrow \tau \vDash Y \doteq X$

$\tau \vDash (X :: C_j) \doteq Y \Longrightarrow \tau \vDash Y \doteq Z \Longrightarrow \tau \vDash X \doteq Z$

$\vdots$

Figure 5.5: Some generated and proved algebraic properties (here $C_i < C_j$)

Clavel and José Meseguer [BCM04]), which incidentally already includes HOL. Ultimately, the idea can be pursued further by extending the process into (PA' + UML/OCL + Isar_HOL) ⇒ (PA' + UML/OCL + Isar_HOL), for a logic PA' not already subsumed by HOL for example. By presenting packaging functions as decision procedures, we have now the required ingredients to implement a generic platform intending to ease both the implementation of decision procedures as well as packages in Isabelle/HOL, so to tend towards a kind of "Object-Logic Package Manager".

As an example of realistic domain-specific problems supported by HOL-OCL 2.0, we refer to the set of definitions, lemmas and corresponding proofs currently generated by our UML/OCL Class Model Package (analysed in Chapter 7, and listed in Appendix B and Appendix C). A UML *class model* underlying a given OCL invariant or operation contract produces several implicit operations which become accessible via appropriate OCL syntax. In more details, the fragment of UML/OCL class models contains:

- classes consisting of typed attributes and their inheritance relation,
- associations and aggregations between classes,
- class invariants (from the OCL contract language), and
- operations on classes (from the OCL contract language).

From a class model, the Class Model Package generates a set of Isar_HOL commands comprising:

- type definitions for each class names $C_1, \ldots, C_n$ sorted according to the inheritance relation,
- accessors `_.allInstances()` returning the set of all object instances existing at some time in the state of a system,
- definitions of accessors (destructors) for each attribute of a class, dereferenced in the pre-state (e. g., `_.age@pre`),
- definitions of accessors (destructors) for each attribute of a class, dereferenced in the post-state (e. g., `_.age`),
- for each class name C, tests of the form `_.oclIsTypeOf(`C`)` testing the *dynamic type* of an object, i. e., the type under which it was dynamically created,
- for each class name C, tests of the form `_.oclIsKindOf(`C`)` testing if the *dynamic type* of the given object belongs to one subtype of C,
- for each class name C, definitions of cast of the form `_.oclAsType(`C`)` always preserving the dynamic type of its argument (irrespective of C),
- for each class name C, there is an instance of the overloaded referential equality (written $\_ \doteq \_$),
- and finally all properties setting up the object-oriented datatype theory. A non-exhaustive overview is provided in Figure 5.5, and Chapter 7 is specially dedicated to the explanation of these properties.

These definitions refer to a typed denotational model, the *object universe*. In the algebraic layer, UML/OCL has an own type discipline providing *basic types* such as `Boolean`, `Integer` and `String` as well as *collection types* such as `Set(`X`)` and `Sequence(`X`)` (i. e., lists). While a one-to-one shallow mapping of the basic and collection types has been established in HOL (detailed in Chapter 4 [BTW14]), the part dealing with class types (via a denotational *object universe* $\mathfrak{A}$) requires another formalizing strategy. This is because notions like "sets of classes" make only sense on the syntactic level in HOL, where in this setting "classes" are considered as first-class citizen elements (so *constants* but not *types*). A *meta-level* construction is thus unavoidable to process class-models, for ideally obtaining an automated treatment as smooth as a regular one-to-one shallow mapping in HOL (hence the need of using higher expressive constructs like packages to implement such meta-level construction).

In previous work, given a particular example of class models in input, a formalization of the corresponding Isabelle definitions, lemmas and proofs has been performed by hand (namely, by manually writing by hand the examples of "Employee Analysis Model" and "Employee Design Model" in the associated formalization [BTW14]). In the present work, we generalize one step further: from an arbitrary class-model, definitions and above listed properties are automatically derived, like the usual deriving obtained when executing packages for datatypes, records or quotients in HOL systems.

Fortunately or unfortunately, after deriving such properties, the next step is to execute them: what happens if at run-time the execution of a given package does not seem to terminate? Can we precisely locate which tactic is being

performing the expensive computation? For the case of a simple lemma (which generates one theorem), Isabelle/jEdit is particularly suitable to experiment step by step which tactics to apply, undo some operations, as well as interchange the tactics being edited, since the editor has been optimized in many ways for a smooth prototyping of proofs, and accordingly treat tactics as atomic actions. On the other hand, the act of generating properties is different than a simple edition. Although the datatype package already generates hundreds of theorems (for some basic example like "LIST"), UML/OCL object-oriented semantics has a surprisingly rich theory: the Flight example of Figure 3.1 leads to more than 2000 theorems. In case an "apparent" non-terminating computation is arising, it becomes then quickly desirable to know if this non-terminating computation comes from the incapacity of the generator to generate some code or comes from the execution of what has been generated. This is particularly relevant whenever the code behind the generation is implementing non-trivial algorithms, is resembling to a realistic compiler (e. g., counting more than 10000 lines of code [Ler09], like in HOL-OCL 2.0), and whenever the generated tactics are resolving non-trivial theorems (e. g., the proofs of cast operations or lemmas related with `_.oclIsKindOf(_)` presented in Chapter 7).

Generally, in terms of readability, mathematical proofs are especially valuable, as soon as one becomes convinced that all assumptions and axiomatizations employed can indeed be ethically invoked. So having the possibility to read and study which lemmas was generated and how they are proved will permit for example to judge the pertinence of an object-logic theory with more conviction. Furthermore, in case a theory document is generated, erroneous introduced assumptions (if any) will have the possibility to regularly occur in several related theorems, so chances to detect such irregularities become multiple according to the number of bloc of related theorems.

To sum up, the next sections will focus the attention on the following points:

- **Efficiency:** How to maximize the maintainability and portability of (large) packaging functions of the form "$\mathcal{L} \Rightarrow$ Isar_HOL"? Can we benefit from substantial performance improvements similar as what one may get with decision procedures?

- **Readability:** How to readably inspect the contents of generated proofs and tactics being executed by the above point, so to potentially inherit from the readability of Isar (and HOL) [Wen99, BW01, WW02]?

- **Provability:** In terms of trusted computing base, how far can we mechanically relate the two above points, i. e., is *"the readable code that makes us convinced"* really equal to *"the efficient code that will be executed"*? Can we prove the termination of the generation process (including all type-checking stages for example) or establish properties related to the translation (like semantic preservation)? To which extent are we able to predict that a generated Isar_HOL content is well-typed (or well-proved), without actually the need to run the type-checker? Can we minimize its use, provided one has a reason to believe that the well-typing of a theory document can be incrementally preserved, like the preservation of a semantics implied by a reproductive partial ordering?

5.3 The Apparatus of the Reproduction Process

The standard solution for increasing the expressiveness of a supported type-system or object-logic in Isabelle (e. g. HOL, required by class-models or HOL-based decision procedures) is to implement the needed constructions inside a more expressive *meta-layer*. This is generally performed by first accessing the layer where this object-logic is being simulated or has been defined, in our case its source code. ML has the property to be a suitable Turing complete layer where HOL is implemented on top (the overall architecture follows the LCF-principle [GMW79]). As such, the framework offers the possibility to "drive" the core engine by user programmed ML code in a logically safe way. However, although it is unavoidable at the end to compute particular Turing complete expressions (including the parsing of arbitrary Turing complete languages), several reasons have incited the present work to not restrict the entire construction of the packages to the sole use of ML, but to take advantage of all sub-components made available by the framework.

Knowledge of the internals. More than ten years ago, Amine Chaieb and Tobias Nipkow observed that programming proof decision procedures in LCF-style in ML was disadvantageous compared to an HOL-based approach [CN05]. Despite noticeable improvements on communication technologies between the logical language HOL and the meta-language ML [WC07], they argued that *"it requires intimate knowledge of the internals of the underlying theorem prover (which makes it very unportable)"* and *"there is no way to check at compile type if the proofs will really compose (which easily leads to run time failure and thus incompleteness)."*

While this remark was done in the context of decision procedures and not packages (which are perhaps an easier task), we believe that the reproduction process to be presented in this section is applicable to both, as sketched in Section 5.2. Programming with the ML library is different than programming with the usual Isabelle commands one is entering in Isabelle/jEdit. For a person only familiar with the Isabelle commands, this requires a certain effort before being familiar with the organisation of the ML library and how it is functioning.

As shown in Figure 5.6, the Turing completeness of ML allows to simulate the execution of arbitrary commands, like lemma (occurring on top). So, for the particular case of commands, code of commands written in ML can be *equivalently* expressed in Isar_HOL (without the use of ML): this is one property coming from the architecture of the framework based on LCF. However reciprocally, starting from a set of Isar_HOL commands, writing an equivalent same counterpart in the sole use of ML becomes longer to achieve. This is because Isar_HOL already allows to concisely express what would express an expanded ML term: in the ML source, free or bound variables are not coloured distinguishably (e. g., instinctively, how many times is *l_apply* used in the picture?), functions receive additional arguments, the theory contextual environment is made explicit, monadic programming style [Mog91] becomes particularly involved... Additionally, these constraints must be multiply taken into account when the purpose is not only generating one lemma, but especially thousand proven ones. Moreover, inside one lemma, various combinations of tactics must again be multiply taken

```
lemma n: "l_spec ⟹ concl" proof l_apply qed o_by
```

```
ML {*
in_local (fn lthy => lthy
|> Specification.theorem_cmd Thm.theoremK NONE (K I) (To_sbinding n, []) []
      (List.map (fn (n, (b, e)) =>
        Element.Assumes [( ( To_sbinding n
              , if b then [[Token.make_string ("simp", Position.none)]] else [])
              , [(of_semi__term e, [])])])  l_spec)
      (Element.Shows [(((@{binding ""}, []),[(of_semi__term concl, [])])]) false
  |> fold semi__command_proof l_apply
  |> (case map_filter (fn META.Command_let _ => SOME []
                       | META.Command_have _ => SOME []
                       | META.Command_fix_let (_, _, _, l) => SOME l
                       | _ => NONE)  (rev l_apply) of
       [] => global_terminal_proof o_by
     | _ :: l => let val arg = (NONE, true) in fn st => st
      |> local_terminal_proof o_by
      |> fold (fn l => fold semi__command_state l o Proof.local_qed arg) l
      |> Proof.global_qed arg end))
*}
```

Figure 5.6: Knowledge of ML's library required

into account, where the same training is required to precisely locate how tactics can be implemented (as in Figure 5.6).

As remark, Isar_HOL has been designed to precisely write human-readable proof texts and enhance the presentation of theories. This is then perhaps one reason why the Archive of Formal Proofs (AFP)[4] contains minimal ML code compared to the code base involving only Isar_HOL (those without ML constructs). Generally, maintaining ML code base can require a certain effort (e. g., in HOL-OCL [BW08a]), even when Isar tactics are involved (e. g., in the seL4 project [KAE+10]), and generally in major domain-specific proof languages [MWM14].

Maintenance of the internals. Besides the need to acquaint some knowledge with the ML library organization, maintaining ML code depending on the library would become all the more easy if this task can be at most minimized. By regrouping together all the code that are depending on the library in a common place behind an abstract interface, then the maintaining task will be only restricted to the code behind this interface, and this task can in parallel be delegated to (potentially other) persons already familiar with the ML internals. Indeed in Isabelle, as in many actively developed interactive systems, updates concerning the content of its source code in ML have the possibility to happen more frequently than, for example, modifications of its own Isar_HOL grammar

[4] An Isabelle public repository with formalizations: `http://www.isa-afp.org/`

language. This is especially relevant when such updates do not affect the overall semantics of the language, or only a part occurring outside the subset where the implementor of $\mathcal{L}$ has a current interest.

At the same time, for the simple purpose of writing a packaging function of the form "$\mathcal{L} \Rightarrow$ Isar_HOL", one may be only interested on concentrating on one suitable Isar_HOL interface, without knowing in detail which internal language is implementing this interface, as long as certain requirements concerning this Isar_HOL interface is respected.

A Meta-Model for the Isabelle API.

Since the early inception of HOL-OCL 2.0, we have opted in this thesis to appropriately exploit several *technical* characteristics of the Isabelle framework in order to rule out the aforementioned issues of maintainability, portability and compositionality of proofs.

On the one hand, having an abstract API representing as *closely* as possible the Isar_HOL language would ease users already familiar with Isar_HOL to develop Isabelle packages, and to ideally incite experts of $\mathcal{L}$ to analyze consistencies of packages related with $\mathcal{L}$. On the other hand, the design of such abstract API has to be carefully performed. This is because both the constructions of the API and the packaging function $\mathcal{L} \Rightarrow$ Isar_HOL are fundamentally related with the requirements mentioned at the end of Section 5.2: namely, the properties of efficiency, readability and provability.

Fortunately, the flexibility to embed many languages in Isar_HOL is due to a combination of major features provided by the associated framework. These features notably comprise the editing engine and all surrounding technologies made available by the framework to practice programming activities and proving activities in a large sense. Following the Curry-Howard isomorphism (also called Curry-De Bruijn-Howard isomorphism) [CFC58, dB80, How80], we see the framework as a "meta" semantic container able to connect multiple logics with multiple languages together, as illustrated by the variety of object-logics in Isabelle, and the range of domain specific languages already formalized and submitted to the AFP.

In our novel approach, we are taking advantage of all sub-components of Isabelle (comprising Isar, HOL and ML). The approach is a particular *combination* of the following steps:

- We define an abstract syntax of our DSL in input in HOL (in the parlance of researchers in UML and Model-driven Architecture (MDA), this is a "meta-model" of UML). We shape our UML meta-model according to our first needs and refrain from completeness or full compatibility to existing standards.
- We define an abstract syntax of (an aspect) of the Isabelle kernel API in HOL [TW15]. Again, we deliberately privileged as a first modelling high-level abstractions over completeness.
- We define a translation between the former and the latter "UML/OCL $\Rightarrow$ Isar_HOL" (called "meta-translation"), still in HOL (to target provability requirements), which comprises the generation of declarations, definitions in terms of denotational constructions, and tactic proofs.

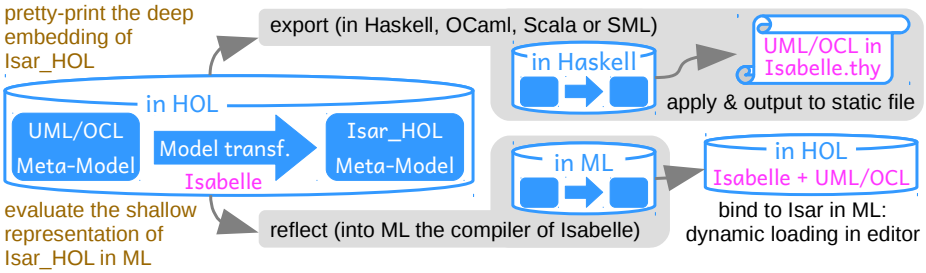


Figure 5.7: Building packages in Isabelle/HOL: targeting readability, efficiency and provability requirements

- We provide for the above Isabelle API a common infrastructure to generate Isabelle parsable text (targeting readability), as well as to generate Isabelle interpretable code (targeting efficiency). The interpretable code relies on a code-generation setup reflecting HOL terms to ML terms. Then at ML side, we can perform the parsing in input from UML/OCL syntax, and the binding in output to the own native Isabelle kernel in ML.

The overall schema we are following is depicted in Figure 5.7.

Source Syntax: The UML/OCL Meta-Model

We define meta-models of the compiler using the Isabelle datatype or record. We present as example just (the entry-point of) the UML/OCL class meta model (which may resemble to the Toy meta-model of our formalization [TW15], but this last contains lots of simplifications)[5]:

```
datatype uml_class = Class
           string                                  (* name of the class    *)
         (string (* name *) * uml_ty) list         (* attribute            *)
           string                                  (* link to superclasses *)
```

As an example, we take the first two class definitions shown in Figure 3.2 and present them in this abstract syntax datatype (the term command just type-checks it for presentation purposes):

```
term [Class "Flight"      [("seats", UmlTyInteger),
                           ("from", UmlTyString),
                           ("to", UmlTyString)      ] "OclAny",
      Class "Reservation"[("id", UmlTyInteger)      ] "OclAny", /.../]
```

Target Syntax: The Isabelle Meta-Model

Our abstract syntax of the Isabelle API supports the representation of

- types, terms (with syntax-declaration elements),
- elements for tactics and Isar high-level proof methods, and

[5] `OclAny` is added by the compiler as a super class inherited by all other classes.

- Isabelle outer commands (like datatype, lemma, locale, ...)

Here the manipulation of the monadic editing environment (like global context and proof context) becomes implicit: we aim to be as close as when one is editing in Isabelle/jEdit. This slight abstraction of the "real" internal interfaces might both enhance usability and portability. As an example of abstraction, we did not need polymorphic datatypes for the Class Model Package, so our current version of meta-model for datatype looks as follows:

```
datatype hol_datatype = Datatype
           string                                  (* name of the datatype *)
         (string (* name *) * hol_ty list) list (* constructor             *)
```

All commands are finally regrouped together in a general entry-point [TW15]:

```
datatype hol_theory = Theory_datatype    hol_datatype
                    | Theory_definition  hol_definition
                    | Theory_lemma       hol_lemma      | [...]
```

As remark, since these two datatypes are Isabelle datatypes, we can proceed as above and present them together in a general term:

```
term [Datatype "hol_datatype" [("Datatype",
   [  TyVar "string"
   ,  let list = λx. TyApp "list" [x] in
      list (TyPair (TyVar "string") (list (TyVar "hol_ty")))])],
 Datatype "hol_theory" (List.map (map_pair id (λa. [TyVar a]))
                [  ("Theory_datatype"      , "hol_datatype"      ),
                   ("Theory_definition"    , "hol_definition"    ),
                   [...]                                          ]), [...]]
```

Two Strategies of Code Generation

After having defined one *single* translation in HOL of some meta-model to this Isabelle model, we can choose at present *two* scenarios of exploitation. They are complementary from a certain perspective, if not equivalent: the result of this translation (so the generated Isar_HOL commands) can be either immediately executed (bottom of Figure 5.7), or converted to a string in concrete Isar_HOL syntax that can be stored in a file to be executed step by step for presentation purposes (top of Figure 5.7). Both scenarios use two different variants of code-generation inside Isabelle/HOL: namely, code-reflection and code-exportation.

The Reflection Scenario

The principle of compiling a formula with computational content to code, evaluating it, and re-introducing the result in derivations over the formula is called *reflection*. In the general domain of meta-reasoning, reflection has been a well-known concept ranging from the area of logic to programming languages, e. g., in 3-Lisp [Smi82, Smi84] — a pointer to a general survey can also be provided here [Cos02]. There is meanwhile a large body of publications on this technique often applied in interactive theorem proving systems (as non-exhaustive

list, we can cite some of them [BM79, Wey80, ACHA90, Bas93, Har95, Bou97, VGPA00, CN05]), where a universal axiomatizing approach has already been brought [CM96].

After reflecting the initial HOL translation function, we obtain an equivalent ML function which is automatically added in the ML environment of the running system. Thus this function can be used as any other ML function:

1. we bind to its input a parser reading tokens from the Isabelle/jEdit editing window. The parser is connected to the Isar_HOL syntax engine (e. g., to support UML/OCL syntax), so that one can write usual UML/OCL command names (Class, Association, Instance, etc.) in Isabelle/jEdit and trigger the execution of the ML reflected function in return.

2. then we map its output (i. e. the ML reflected API model of Isar_HOL) to the own Isabelle's ML interface of Isar_HOL.

Finally, the combination of both forms a way to implement new packages in Isabelle/HOL, as any other Isabelle packages, but here we are also relying on potential optimisations made by the code generation (like decision procedures implemented in HOL). The construction directly benefits from an implicit "shallow" integration in Isabelle/jEdit, with many associated functionalities: for example, syntax annotations (or "constant bindings") become available as usual. Then a click on an accessor in some OCL formula will let the Isabelle/jEdit interface "jumps" to the corresponding definition inside a class model definition. Obviously, the generated code is still checked by the Isabelle kernel, then assuring the correctness of the underlying constructions as in any other package. We call this scenario of execution, the "shallow (reflection) mode".

The Exportation Scenario

For readably present the generated content and debugging purposes (this partly addresses the aforementioned issue of "intimate knowledge of the internals"), it is convenient to observe the generated declarations by several means: e. g., having a file containing the generated HOL definitions and Isar proofs, and execute them on a step by step basis by hand. The exportation scenario resembles to the above reflection scenario, except that no bindings happen between the Isar_HOL meta-model (presented in this thesis [TW15]) and the Isabelle's ML interface of Isar_HOL. We write instead a pretty-printing function from this Isar_HOL meta-model to string, so that this string can be ultimately saved to file, called the "deep-certificate". Besides ML, the pretty-printing process and saving to file can actually be performed in Haskell, OCaml, or Scala since these processes do not use the Isabelle's ML interface of Isar_HOL. So in this scenario of execution, called the "deep (exportation) mode", the generated Isar_HOL content is not evaluated, but only represented as a string.

Unifying Both Scenarios

However a little work is still needed here for the exportation scenario to be used as smoothly as in the reflection scenario, i. e., in an interactive setting where the language in input is the UML/OCL language one is entering in the editor. Then Chapter 6 will pave the way for an automated treatment, by unifying

```
theory Scratch   imports  Main
                 keywords "Term" :: diag   begin

datatype LIST = NIL | CONS nat LIST

fun    height :: "LIST ⇒ nat"
where "height NIL        = 0                "
    | "height (CONS _ t) = Suc (height t)"

declare [[ML_source_trace]]

ML{* val NIL    = @{code NIL}
     val height = @{code height}
     val _      = height NIL     *}

ML{* Outer_Syntax.command @{command_keyword Term}
      " Term reads and prints an arbitrary HOL term "
      (Parse.term >> (Isar_Cmd.print_term o pair [])) *}

Term "height a + height b = height b + height a"

find_theorems

end
```

Figure 5.8: Defining new commands on the fly: the new Term command

with a special command "generation_syntax" the two presented scenarios, i. e., allowing to choose in Isabelle/jEdit between the reflection or exportation without changing the UML/OCL expressions provided in input.

ML Antiquotations (I): Static Embedding into System Runtime

We close the section by detailing certain noticeable functionalities related with ML commands, that have been used when performing reflection and the definition of new Isar_HOL commands.

As suggested in Figure 5.6, in modern Isabelle, ML code can be arbitrarily mixed with any other commands in the editor. Via code antiquotations [WC07, Wen16a], ML extensions can be programmed comfortably, since unlimited accesses to the own source of Isabelle are granted within Isabelle/jEdit (at run-time). Thus, by approximating with a certain "meta" perspective the code generation as an identity function, one can start some programming or proving activity in the full Isabelle framework (with any interleaving of Isar, HOL and ML), to later refine the same activity in ML.

In order to demonstrate the relevant technical features, we present a screenshot in Figure 5.8 showing a session based on Isabelle/HOL that consists of the only file `Scratch.thy`. As usual, we retrieve the header mentioning theory and the "imports *Main*" clause ("*Main*" is a synonym for HOL) and then a sequence of commands: datatype, fun, declare, ML... Isabelle sessions can be extended by

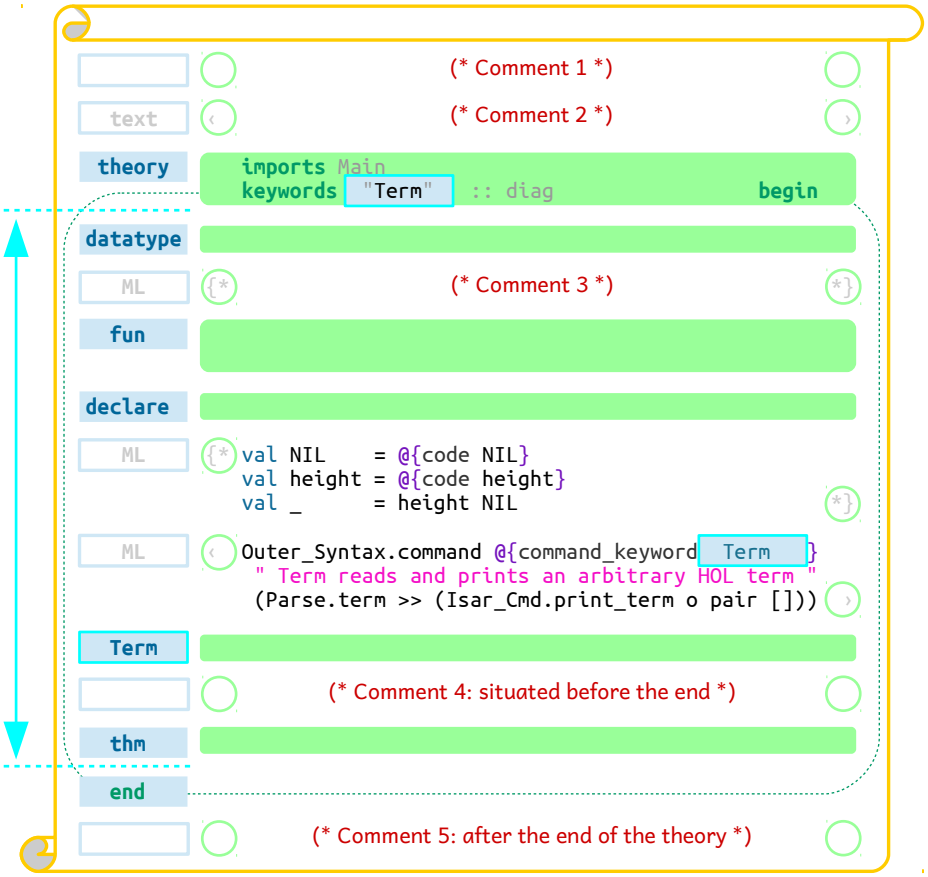


Figure 5.9: Isabelle session seen as a gigantic ML top-level

user-defined commands, a feature we use for defining on the fly (at run-time in Isabelle/jEdit) a new command called "Term".

By unusually inversing the color of this theory file, we can better explain the effect of the ML command. Figure 5.9 shows a content resembling to Figure 5.8, but tokens are here described as a list of repetitive blocks of two elements: one blue command, immediately followed by a green area (which can optionally be empty depending on the parsing policy of the blue command). As remark, only comments or informative messages can be written outside a theory: e. g. for comments, we typically use the command text, which can occur before the command theory and after the command **end** (although its color is green in the picture, **end** can be assimilated as a blue command).

In Figure 5.9 the largest white color in background has the meaning to specially denote ML programs. This is to accentuate that, at any positions, ML programs has the possibility to be interleaved with syntactic blue and green blocks. Indeed, Isabelle itself is built on top of an ML execution environment, and in fact, Isabelle *is* a collection of modules implemented in ML and added into the ML environment of execution. So blue and green blocks can be thought of as abbreviating internal ML code, so that the global file is basically nothing but an

entire ML top-level. Thus any blue block (together with its following green part) can be simulated with a corresponding piece of code appropriately written in ML. For example, it is possible to replace an entire theory file `Example.thy` with an *equivalent* one, mostly coded in ML, so that other files importing `Example.thy` (with "imports") would not know the percentage between 0 and 100% of ML fragments contained in `Example.thy` without a closer inspection inside the file `Example.thy`. Historically, ML itself was developed as a *meta-language* and an execution environment for theorem provers similar to Isabelle [GMW79, Gor00].

In principle, the ML command just gives access to the underlying ML execution environment: ML{∗`3 + 4`∗} compiles "`3 + 4`", executes it, and optionally displays the result in the output window. However, when so-called *code-antiquotations* [WC07] such as `@{code NIL}` are used, the process is more involved because ML antiquotations implicitly refer to values declared "at Isar_HOL side". Concretely, an additional processing step is needed to resolve the appropriate dependencies before the ML code can be compiled. declare`[[ML_source_trace]]` activates an option to inspect in detail the resulting ML code in the output window. For example, the two antiquotations `NIL` and `height` generate among other the following ML code (for clarity reasons, certain names of variables have been slightly renamed afterwards):

```
structure Generated_Code =
 struct
   datatype nat  = Zero_nat | Suc  of nat        ;
   datatype list = NIL      | CONS of nat * list ;
   fun height  NIL          = Zero_nat
     | height (CONS (x, t)) = Suc (height t)     ;
 end
```

This ML code looks close to the one we have defined at Isar_HOL side (where instead we used datatype and fun). Finally during the compilation, antiquotations are automatically replaced with their corresponding values:

```
val NIL    = Generated_Code.NIL
val height = Generated_Code.height
```

This makes "`height NIL`" efficiently executable in the context of the compiled code — no symbolic representation is any longer involved.

As remark, instead of using antiquotations, one can also invoke the command code_reflect to explicitly perform the process of reflection on some particular given constants [Haf09, HN10, Haf16].

ML Antiquotations (II): Defining New Isar_HOL Commands

The new command Term we are adding in Figure 5.8 and Figure 5.9 relies on the command ML to interact with the own source code of Isabelle, and to get access to `Outer_Syntax.command` further located in the source. The "Isar" component of Isabelle handling the blue commands, occurring in the "outer syntax" space [Wen16b, Wen16a], is in fact reconfigurable. `Outer_Syntax.command` takes a keyword as argument as well as one associated code to later execute whenever encountering the keyword. It ultimately binds both arguments so that the keyword can immediately be used afterwards like any other command (or any

function ready to be applied). The use of Term as a keyword is possible since we have priorly declared in the header "keywords *Term*". As pointed in Figure 5.9, the space where Term can be employed as a keyword is delimited between theory and **end**. Although the new command Term (with an uppercase "T") has been built based on the code of the existing command term (with a lowercase "t"), generally, any command from the Isar_HOL core API is accessible inside the ML scope. So it is as well possible to implement Term for it to have the same semantics as any other chosen command: namely datatype, fun, theory, or generally any existing Isar_HOL commands (including ML). However as a bootstrapping issue, while Term can be implemented with the code of theory (for it to have the same semantics as the command theory), after doing so, one will never have the possibility to call Term. This is because all user-defined commands (like Term) must be precisely called inside theory and **end** (and theory can not be called inside itself). This remark is not restricted to theory, but generally applies for all keywords having the possibility to occur outside theory, like text.

5.4 Properties of the Reproduction Process

Summing up, the construction presented in Figure 5.7 provides a generic principle to extend Isabelle with packages. To enable the prover to conceive its future object-logic, the reproduction of editing sessions basically requires three ingredients:

1. *formal meta-construction*,
2. *code reflection*,
3. and *own kernel binding*.

Then to perform the translation "UML/OCL ⇒ Isar_HOL", the implementation has capitalized on resources of the full Isabelle framework, i. e. Isar + HOL together, with some fragments in ML. This is to precisely benefit from a number of advantages.

Edition versus Generation

There is a subtle difference between the API presented in this thesis [TW15], and the native interface of ML signatures of Isar_HOL as implemented in the original source of Isabelle. While we see them as complementary, they are serving different objectives, and then they are differently optimized: the interface in ML optimally targets means to obtain reactive and asynchronous *editions* of Isar_HOL documents, whereas the presented API in this work optimally targets means to obtain correct and massive *generations* of Isar_HOL documents.

- *(editions of Isar_HOL documents)* For example in Figure 5.10, the four commands are differently instantiated: their parsers are all different, taking different arguments in input, and each command needs to call a precise piece of code, thus making all commands achieving different functionalities (at least the four presented in the figure). Then, modules responsible to set up datatypes are in "`~~/src/HOL/Tools/BNF`" (e. g., in Isabelle 2014), whereas for ML, functions implementing the evaluation

```
ML
val _ =
  Outer_Syntax.local_theory  @{command_spec "datatype" }
    "define inductive datatypes"
    (BNF_FP_Def_Sugar.parse_co_datatype_cmd
      BNF_Util.Least_FP BNF_LFP.construct_lfp)
ML
val _ =
  Outer_Syntax.local_theory' @{command_spec   "fun"    }
    "define general recursive functions (short version)"
    (Function_Common.function_parser Function_Fun.fun_config
      >> (fn ((config, fixes), statements) =>
        Function_Fun.add_fun_cmd fixes statements config))
ML
val _ =
  Outer_Syntax.command       @{command_spec   "ML"     }
    "ML text within theory or local theory"
    (Parse.ML_source >> (fn source =>
      Toplevel.generic_theory
        (ML_Context.exec (fn () =>
          ML_Context.eval_source
           (ML_Compiler.verbose true ML_Compiler.flags) source) #>
         Local_Theory.propagate_ml_env)))
ML
val _ =
  Outer_Syntax.command       @{command_spec   "end"    }
    "end context"
    (Scan.succeed
      (Toplevel.exit o Toplevel.end_local_theory o
        Toplevel.close_target o
         Toplevel.end_proof (K Proof.end_notepad)))
```

Figure 5.10: The genesis of commands

of ML sources are related (among other) with `ML_Context.eval_source`, `Local_Theory.propagate_ml_env`, etc. So the real source code of Isabelle is organised following a particular policy, containing well-structured collections of ML libraries where each module and each signature has a specific goal to achieve.

Thus, the Isar_HOL interface in ML is modularly organized and optimized to treat fast incoming commands from multiple incremental updates from Isabelle/jEdit. Commands are internally represented in low-level format as monadic combinators (i. e., with types of the form "$t \Rightarrow \alpha \Rightarrow t$"), taking for instance the editing context as additional parameter (i. e., proving context or global context). So at the end, these commands or monadic combinators will be linearly assembled like a stream data-structure, and potential errors are appropriately optimized to happen at run-time.

- *(generations of Isar_HOL documents)* On the other hand, the Isar_HOL API presented in this thesis is precisely designed to *abstract* the internal functioning of commands, and provide a kind of grammar indicating at prototyping time which commands can be called (or generated) after or inside which ones. Ultimately, the aim is to minimize grammatical errors: for

example datatype can not be called just after opening a proving scope like lemma. Similarly, after typing datatype, it is not expected to immediately type sledgehammer, or qed before beginning a proof. In the same spirit, when generating tactics, the Isar_HOL meta-model in HOL would treat the left parenthesis proof with the right parenthesis qed as a single constructor, so that one does not have to remember the number of left parenthesis opened until now, to be closed by one right "qed", and the *right* one. In addition, high-level constructs are modelled with *recursive* datatypes, e. g. for tactics $t := (t \text{ list}) + |(t \text{ list})?| \text{simp} | \text{rule} | \text{metis} | \text{auto} | [\ldots]$.

As summary, the Isar_HOL meta-model in HOL stands as an intermediate data-structure, designed to abstract and provide a *recursively typed* API, capturing the essence of a well-typed tree document (whose structure, as a tree, can be deeply and recursively folded).

Proving in Isabelle/HOL, in $\mathcal{L}_1$, ..., in $\mathcal{L}_n$

By defining the translation in HOL, and using Isabelle/HOL as "implementation language" itself, one immediately profits from a premium access to verified libraries. As pointed by the manual describing object-logics of Isabelle [Pau16]:

> *"HOL is currently the best developed Isabelle object-logic, including an extensive library of (concrete) mathematics, and various packages for advanced definitional concepts (like (co-)inductive sets and types, well-founded recursion etc.). The distribution also includes some large applications."*

Possible relevant libraries for the translation are among other: the formalised red black tree theory, infrastructures on list, pair, monad, or the one defining transitive closures for expressing inheritance relation, plus diverse libraries on λ-calculus from the AFP that can constitute sound foundations for both meta-models: both have to manipulate terms and types.

Furthermore, one can profit of the possibility to *prove* properties over the compiler within the native flexible Isar_HOL language, in a large sense: semantic preserving HOL-based compilations, or correctness properties in HOL-based decision procedures for instance.

Generally, the framework can serve to incrementally build constructive functions, i. e., irrespective of the notion of "a particular theorem to prove". Since the complete compiler has to be ultimately executed, its internal component aims to be built favouring the constructive subset of classical logic. Instead of writing a single block of definition, that same definition we are defining can in fact be incrementally constructed with lemma and a final intuitionistic extract.

For the case of HOL-OCL 2.0 packages, proofs are actually diversely covered ranging from the termination proofs of the compilation functions (which we provided alongside with our construction, they are mandatorily required when defining arbitrary Isabelle/HOL functions), or different studies concerning the implementations of the object-oriented data-structures (detailed with generation_semantics in Chapter 6).

The check of the non-emptiness of all datatypes being defined are then covered. From a syntactic point of view, defining a datatype in ML can be as concise as defining a datatype in Isabelle/HOL. From a semantic point of view,

both approaches follow different consistencies checking [Gun92]: number of lemmas are automatically derived in Isabelle to assure the well-formedness of the data-structures being defined [TPB12, BHL+14]. Thus, one can take advantage of this additional guarantee when defining the full meta-model of UML/OCL in Isabelle/HOL (and this meta-model has a certain size). In particular we will see in future chapters that a meta-model for UML/OCL must be rich enough to capture the description of classes, associations, instances, transitions, invariants... Not only are these additional lemmas proved by the datatype package of Isabelle, but associated folding recursive definitions are automatically provided in order to deeply fold the data-structures being defined. These folding definitions will be used to facilitate various pretty-printing operations to string (in Chapter 6).

Parallel Related Theorem Proving

Because writing a short sequence of tactics can be more rewarding than a long one, generating proofs solving a *class* of theorems can be as well more rewarding than generating proofs for solving only one. We present for instance a tactic function in HOL in Figure 5.11. For the moment, we can just note that this figure is well-typed in Isabelle (and only depending on *Main*), more detailed explanations about what this tactic is solving will be provided in Section 7.5. Although this tactic function might resemble as any usual definitions of tactics, e. g. in Coq's Ltac [Del00] or Isabelle's Eisbach [MWM14], here we are not solving just one theorem but a set of "related" theorems at the same time with this single function.[6] Otherwise said, one can for example use our approach to define an HOL function, solving a set of theorems, where each theorem is itself solved by some tactics in Eisbach (which has been designed to write short sequence of tactics). On the other hand, in our approach, solving a class of theorems is not mandatory: one can also generate a set of tactics for solving only one theorem.

Finally, even if Figure 5.11 seems to have been written in one shot (i. e. with no interactive theorem proving facilities), the debugging of this high-level construction by alternatively inspecting the deep-certificate turned out to be an extremely useful technique, especially when combined with the ability to type-check a set of related theorems at the same time in parallel, natively provided in Isabelle [Wen09, MW10, Wen14].

Meta Theoretical Properties

The presented construction allows to generate certain properties over syntactic and static sanity of the generated functions and models, such as: "if no context errors in the Class Package syntax occur, it can be assured that all generated names for accessors are distinct". In particular, we have taken advantage of the type system of HOL to do some extra type-checking and term rejections. For example, the checking of free or bound variables in the new command Instance

[6] The set contains theorems particularly related, because this function takes the full universe of UML/OCL classes in input and covers all situations: leaf nodes, root node...

```
theory Scratch4
imports Main
begin

locale T
begin
datatype ('a, 'b) tactic
  = simp_only 'a
  | erule 'b
  | simp_depth_1
  | simp_depth_2
  | simp_breadth
end

fun aux_depth
and aux_breadth where
   "aux_depth l_depth =
     (λ [] ⇒ []
      | (class, l_breadth) # l_depth ⇒
         T.simp_only class
         # aux_breadth class [] l_depth (rev l_breadth))
      l_depth"
 | "aux_breadth class tactic l_depth l_breadth =
     (λ [] ⇒ tactic
      | (class0, class0_path_inh) # l_breadth ⇒
         T.erule (class, class0 # map fst l_breadth)
         # (if l_breadth = [] then op # T.simp_breadth else id)
           (aux_breadth
              class
              ( (if class0_path_inh then
                   (if l_depth = [] then op # T.simp_depth_1 else id)
                   (aux_depth l_depth)
                 else [T.simp_depth_2])
                @ tactic)
              l_depth
              l_breadth))
      l_breadth"

end
```

Figure 5.11: Parameterizing which theorem to solve with a list of tactics as complete answer

can be (partly) subsumed to the generation of several definitions of the form:

definition $\text{typecheck}_{\text{Instance}}\_\text{extra}\_\text{variables}\_\text{on}\_\text{rhs} =$
$(\lambda\, F_2\ F_1\ R_{21}\ R_{11}\ C_2\ C_1\ S_1.\ (F_1, Mon, F_1, R_{21}, F_1, R_{11}, F_1, F_1))$

Here, whenever Mon has not been earlier defined in the code, we would automatically get an error, this error being raised by the definition command itself.

As remark, since we apply the code generator of Isabelle to generate code, which will again generate definitions and proofs (obtaining at the end some meta-level code whose results will be checked by the logical core engine), the general reproductive process can *not* be approximated in the precise sense of the word as a simple act of (syntactic) reflection or exportation, rather a tool construction by meta-level modelling not involving additional trust (except the understanding one might have on Isabelle generated theories, and associated arising trust).

Generally, the particular relation between rewriting logic [Mes92] and type theory [ML84] has already been deeply investigated, for example by Mark-Oliver Stehr [Ste02]. In parallel in the present work, a comparison can also be approached with Pure Type Systems (PTS) [Bar91] where inference rules (typically abstraction and application) are reused several times but differ on the nature of the folded (or quantified) sort (which can be a type or a kind). Similarly, having Isabelle/HOL as a back-end of itself shows that Isabelle/HOL (seen as a pure calculus system without considering potential non-terminating aspects from the ML layer) can be reused to fold itself through one deep embedding iteration. In term of expressivity, while one first iteration already allows to express types as first class citizen, comparatively to dependent types, no limitation on the number of iteration does actually occur as constraint.

While the typing of PTS crosses all sorts of hierarchy as a single entity, no particular assumption on well-formedness is initially performed when deeply embedding syntax trees (by default, additional proofs should be brought). However meta-considering Isabelle in itself does not restrict the calculus system to inner syntactic expressions (or object-logic expressions). Since the complete language is covered, HOL can as well be used for generating Isar_HOL tactics (at the meta level, the process of generating tactics is guaranteed to be terminating, whereas their execution may not).

Slightly more challenging, our technique can in principle be adapted to prove meta-theoretic properties such as: "if the class model is well-formed, the generated code will be well-typed with respect to HOL types". When complemented by a semantic model of the Isabelle/HOL API, it is even conceivable to extend our approach by true completeness proofs assuring that the evaluation of the various deep-certificates will not fail. However, this is a very ambitious task (not yet implemented) that appears feasible only for simple rewrite-oriented proofs or for the checking of simple proof-objects. We nevertheless consider the overall construction of the reproductive process as a major step into this direction.

CHAPTER 6

Meta Theorem Proving in HOL-OCL 2.0

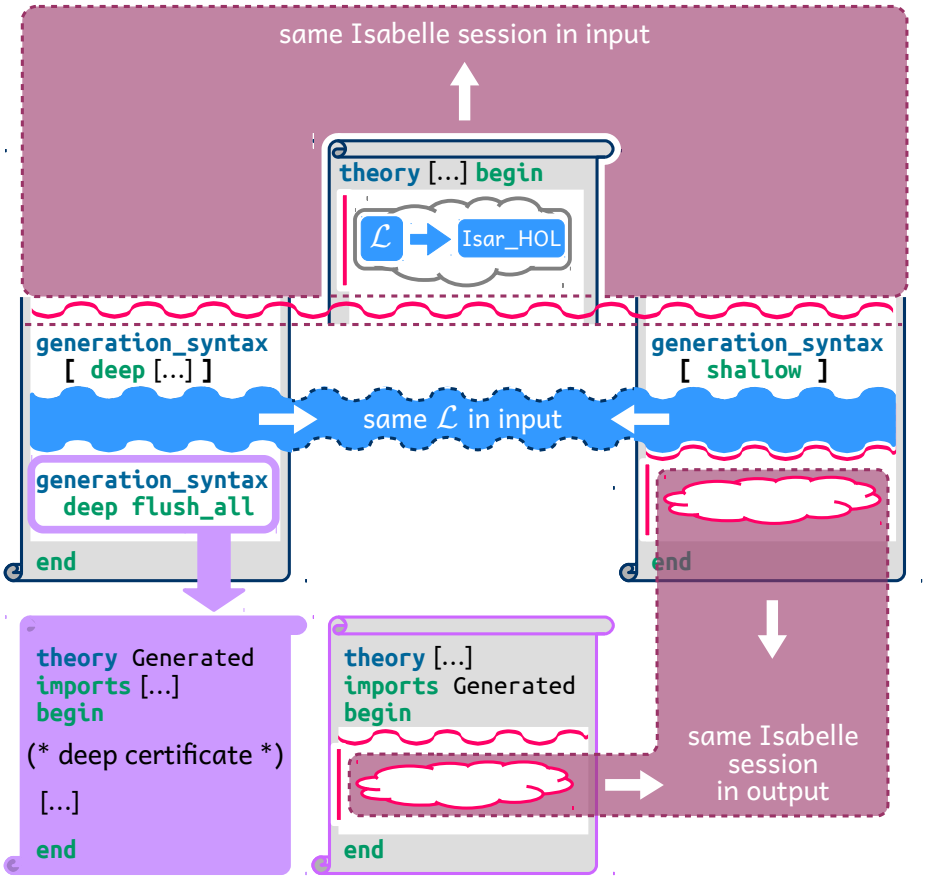
The embedding function $\mathcal{L} \Rightarrow$ Isar_HOL, described as a packaging function in the previous Chapter 5, becomes now interactively considered inside the editor Isabelle/jEdit. Due to the approach consisting to sequentially embed a chain of languages $\mathcal{L}_1 \cdots \mathcal{L}_n$, one could even obtain at the end an infrastructure supporting the modelling of Ouroboros programs [AJGL11] (which are mutually generating programs [Kle38, Cut80]: e. g., P_1 written in $\mathcal{L}_1$ which produces in output a program P_2 in $\mathcal{L}_2$ so that the execution of P_2 yields exactly P_1 in its turn[1]). The present work will nevertheless be regarded as an antagonist work for several reasons. The aim of successive embeddings presented here is to merely not form cycles, we imagine the reproduction process as a one way process, growing in many directions as a genealogical tree. In particular, several running modes of animation respectively illustrating Figure 5.7 will be presented for the construction to avoid cycles at run-time depending on the running mode. Whereas the deep exportation mode will delay the loading or load step-by-step the semantics of a given piece of code in $\mathcal{L}$, the shallow reflection mode will execute at full speed the semantics of this piece of code.

Precisions will also be provided on the limitation of such embedding and which symbols are needed or not to delimit the enclosing scope of the embedded languages. This is for programmers and computers to unambiguously know if a given piece of code has to be understood belonging to $\mathcal{L}_1$, $\mathcal{L}_2$, or somewhat else.

6.1 Modelling in deep and Executing in shallow

To animate the semantics of some piece of code written in $\mathcal{L}$, we integrate in the jEdit-based Prover IDE of Isabelle a special command, called generation_syntax, to fine-grained select which behaviour in Figure 5.7 to execute when encountering that piece of code. Furthermore, in order to determine if a given piece of code in Isabelle/jEdit has to be understood as a piece of code belonging to $\mathcal{L}$ or Isar_HOL, we introduce the terminology of *meta-commands*. By meta-commands, we precisely designate any Isabelle commands satisfying all the following conditions:

[1] https://en.wikipedia.org/w/index.php?title=Quine_(computing)



As remark about the two drawn Isabelle sessions in output, here the names of certain constants might need to be prefixed with "Generated" in one case, and prefixed with nothing in the other case. To be rigorous, the "exact" similarity only occurs when we compare the (end of the) deep-certificate with the (end of the) file in shallow-mode. Then we must also assume that both files have the same name and import similar ancestor theories.

Figure 6.1: Commutative diagram linking deep with shallow

1. commands manually defined with keywords in the header of a theory, e. g., like Term in Figure 5.8, and

2. commands C which are bound to the reflected ML translation function of Figure 5.7, so that finally one can parameterize the semantics of C at run-time to fluctuate between the exportation scenario or the reflection scenario.

For instance, the previously defined command Term will not be called as a meta-command, because it does not use the reflected translation of Figure 5.7. On the other hand, Class, Association, or Instance are examples of meta-commands defined in respective packages of HOL-OCL 2.0.

At any editing position, the special command generation_syntax can change the semantics of future incoming meta-commands representing $\mathcal{L}$: it takes a list

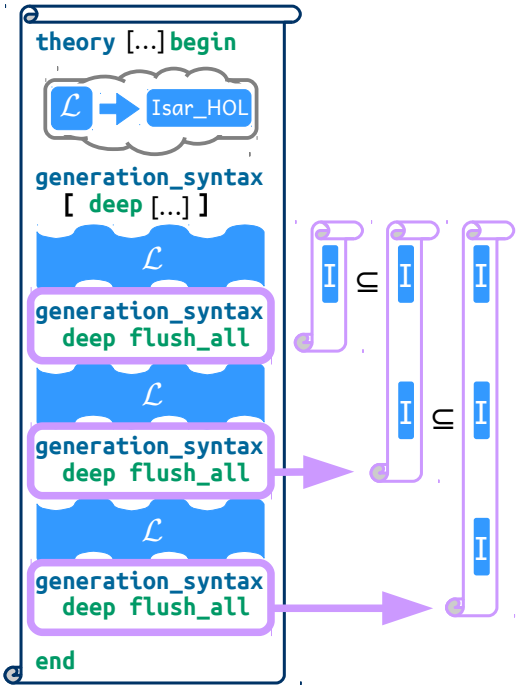


Figure 6.2: Incremental generations in deep

of animating mode, either deep, shallow or any elaborated combinations that can concur at the same time. In particular the fastest semantics is obtained with an empty list: when it is set, only minimal syntactic checks are supposed to occur afterwards. Figure 6.1 establishes as general idea or conjecture the equality relation of sessions between deep and shallow. Starting from a language $\mathcal{L}$ embedded into Isar_HOL, and a piece of code written in $\mathcal{L}$, the piece of code can exhibit two symmetric behaviours depending on if deep is given to generation_syntax or shallow. These two behaviours reflect exactly the two ways to export the meta-translation presented in Figure 5.7.

- In particular, deep can take additional parameters to specify which intermediate languages to use for generating the deep-certificate "`Generated.thy`" in the hard disk (among Haskell, OCaml, Scala, or SML). Then, at any time in deep-mode, one call of generation_syntax deep flush_all will perform as side effect the saving of the generated Isar_HOL commands associated to the piece of code written in $\mathcal{L}$, by invoking the respective compiler of the chosen intermediate language (several intermediate languages can also be chosen in parallel).

  Because all meta-commands are considered or possibly reconsidered again for the generation, we can obtain at the end several well-typed generated theories which are related by a particular relation of partial ordering. For example, Figure 6.2 shows an increasing ordering of three well-typed generated elements. As remark, the smallest element of this relation could be a not empty file: whenever we immediately call generation_syntax deep

flush_all, just after setting a file in deep-mode, the emptiness of the resulting generated theories actually depends on how the embedding from $\mathcal{L}$ to Isar_HOL has been defined.[2]

- In shallow-mode, all operations totally occur in RAM memory, including the operation simulating the execution of some generated deep-certificate. However in contrast with the deep-mode, no certificates are produced after or during generation_syntax [shallow]. Both the reflection step from HOL to ML, and the translation step from this reflected ML code to native ML interface of Isar_HOL happen before the definition of generation_syntax. So the execution cost of generation_syntax [shallow] is roughly $O(1)$, and after being correctly parsed, incoming $\mathcal{L}$ meta-commands have just to apply the reflected translation (that can already be more or less optimized into some native code, depending on the compiling strategy of the ML compiler used by Isabelle).

At the end, after loading both the deep-certificate and the file in shallow-mode in Isabelle/jEdit, one would finally obtain two similar sessions. It is as if they have executed similar definitions, theorems and proofs, except that:

- one needs to debug the RAM memory in shallow-mode to understand what Isar_HOL commands have been generated,
- whereas for the deep-certificate, it comes with a certain level of readability, but its creation costs an extra step of printing to string (normally to the hard disk), plus the cost associated to future parsing and type-checking (in intermediate languages, in Isar_HOL as well).

More generally, this commutative diagram can be understood as a monadic type of the form: $t \Rightarrow \alpha \Rightarrow t \times t$ or simply $t \Rightarrow \alpha \Rightarrow t$, where t represents the state of the Isabelle session and α the piece of code in $\mathcal{L}$ taken as experimentation. In particular, interleaving of modes can occur among any chains of embedding. For example, Figure 5.2 does not precise if the three grey arrows are similarly all executing in shallow-mode, or all executing some extracted deep-certificate each time (assuming we firstly forget that this figure represents a single file).

deep generally aims to complement shallow because the former can be assimilated as a process producing at the end a formal specification which can be visually inspected. Later, the formal specification can serve as a certificate to justify the execution of shallow. Given a deep-certificate and a theory file in exclusive shallow-mode, running both sessions in parallel (in separate Isabelle/jEdit processes) allows to inspect and potentially detect a non-terminating tactic, that task can be harder without having at hand a deep-mode (it would mean to only debug in shallow-mode).

Irrespective of the running mode (deep or shallow), generation_syntax takes further arguments to influence the semantics of generated contents, this is performed with the keyword generation_semantics. However this is just a slight influence, for example concerning optimizations we might have on the choice of data-structures used to model certain datatypes. So noticeable differences for

[2] Without loss of generality, we will see in Chapter 7 that class models considered in this thesis always have at least one class ("`OclAny`") automatically added by default.

```
(* before Isabelle 2014 *)
ML {*
  ((snd oo Datatype.add_datatype_cmd Datatype_Aux.default_config)
    [((To_sbinding n, [], NoSyn),
     List.map (fn (n, l) => (To_sbinding n, List.map s_of_rawty l, NoSyn)) l)])
*}
```

```
(* after Isabelle 2015 *)
ML {*
  (Isabelle_BNF_FP_Def_Sugar.co_datatype_cmd
    BNF_Util.Least_FP
    BNF_LFP.construct_lfp
    (Ctr_Sugar.default_ctr_options_cmd,
     [( ( ( (([], To_sbinding n), NoSyn)
         , List.map (fn (n, l) => ( ( (To_binding "", To_sbinding n)
                                    , List.map (fn s => (To_binding "", s_of_rawty s)) l)
                                  , NoSyn)) l)
       , (To_binding "", To_binding ""))
      , [])]))
*}
```

Figure 6.3: The implementation of datatype has meanwhile changed

end-users would only concern the global resources of the computer, time privileged over space, or vice versa. At the time of writing, design and analysis are such possible options to explicitly state that definitions of classes should be understood and compiled as "aggregations" or "associations" (these notions will be precisely detailed in Chapter 7).[3] Other options have been studied to fine-grain adjust the cost of operations related to accessors on objects and casts:

- one option to optimize the accessing of objects in $O(1)$, at the cost of performing casts in $O(n)$,
- and vice versa, one option to optimize casts in $O(1)$, at the cost of performing the accessing of objects in $O(n)$.

We have experimented several medium-sized samples by hand. In particular Chapter 7 will further mention the implementation details relating both data-structures.

Besides influencing the semantics of the embedded $\mathcal{L}$ with generation_semantics, generation_syntax can produce several variations of Isar_HOL theories, mostly to ease prototyping. For instance, we have added the keyword SORRY to explicitly disable the generation of all proofs, irrespective of the presence of sorry [Wen16b] or not in proofs initially intended to be generated. More conceptually, between Isabelle 2014 and Isabelle 2015, the algorithmic implementation of datatypes has fundamentally changed: datatype was renamed to old_datatype [BW99], whereas datatype_new renamed to datatype [BHL+14]. Since the Isar_HOL meta-model in HOL stands as an abstract interface of Isar_HOL commands (as presented in Section 5.3), changing

[3] As a third option, when nothing is specified, the meta-compiler will accordingly treat all respective notions, so both "aggregations" and "associations".

one supported implementation to another one is relatively transparent. Faster for deep than shallow, because for shallow a look at the source code of Isabelle is finally needed to find the respective Isar_HOL entry-point (located after the parsing expressions, as in Figure 6.3). For deep, a change in Isar_HOL syntax normally only implies a modification in the pretty-printer. To be rigorous, a long term project would prove or evaluate more formally the consequences of successive upgrades of Isar_HOL commands (like datatype). deep-certificates are intended to stand as static witness irrespective of Isabelle versions, while on the other hand modifications in deep-certificates are nevertheless necessary to be aligned with Isabelle and well-typed.

6.2 Testing deep-Certificates Before Checking Proofs

As enhancement, we further optimize the generation of deep-certificates. Figure 6.4 participates to the designing objective of Figure 6.1, by detailing a testing activity automatically performed when generating in several intermediate languages: we have programmed the system to check at run-time that all deep-certificates are similar (by performing syntactic comparisons). As soon as we give Haskell, OCaml, Scala, and SML in a list to deep as argument, generation_syntax will immediately proceed to the code exportation of the meta-translation function, without knowing yet which arguments will need to be translated. Then the first occurrence of generation_syntax creates four versions of the meta-translation, in four respective directories. The optimization consists to precompile these generated functions to object code so that they are all ready to be linked and applied with future incoming meta-commands. Future incoming meta-commands are supposed to heavily change during experimentations, whereas the main meta-translation function is exported once and for all. So it will just remain to compile the "tiny" set of meta-commands associated to each invocation of generation_syntax deep flush_all, then link the overall as last step before the ultimate execution.

However this optimization only works on languages allowing to break the typing inference mechanism. Indeed, the current Isabelle 2016 does not include commands to extract code to functors (only ground modules). This leads to two scenarios:

- For efficiency reasons, one call to `unsafeCoerce` in Haskell, and one call to `Obj.magic` in OCaml are executed to link and apply together "`Function.hs`" and "`Argument.hs`", respectively "`function.ml`" and "`argument.ml`". Similar optimizations are not yet implemented for Scala and are only half implemented for the SML target (which basically performs a step of marshalling to string in Isabelle/ML, the incremental compilation with object code is not yet implemented).

- For safety reasons, we can disable all optimizations: it suffices to extract all the meta-compiler together with the respective arguments in front of each incoming meta-commands every time, then the overall needs to be newly compiled every time. This is the current implemented behaviour for Scala. For Haskell, OCaml and SML, it was also the default behaviour in certain previous versions of the current project, so that functionality can be restored if needed.

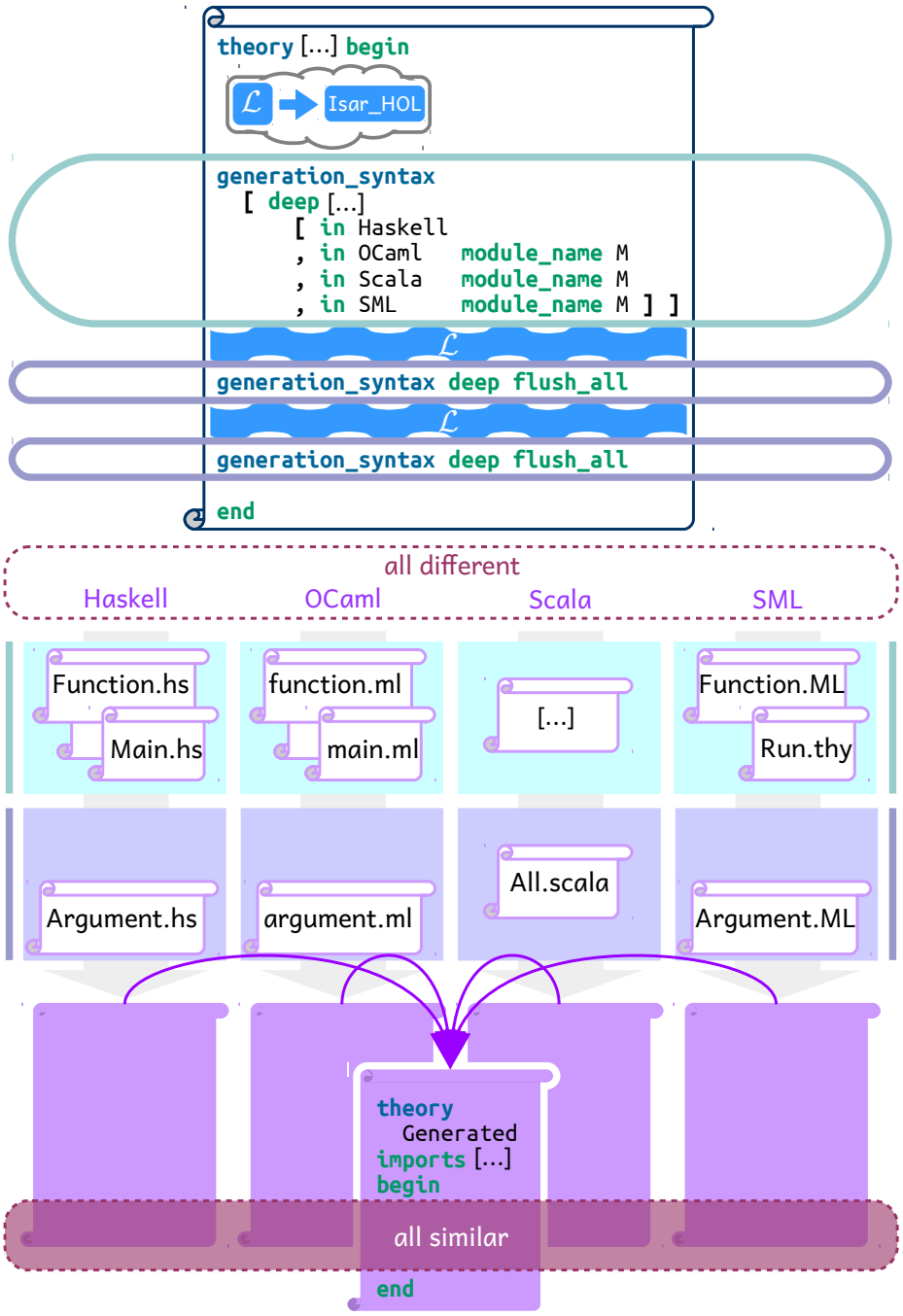


Figure 6.4: Multiple targets of generations in deep

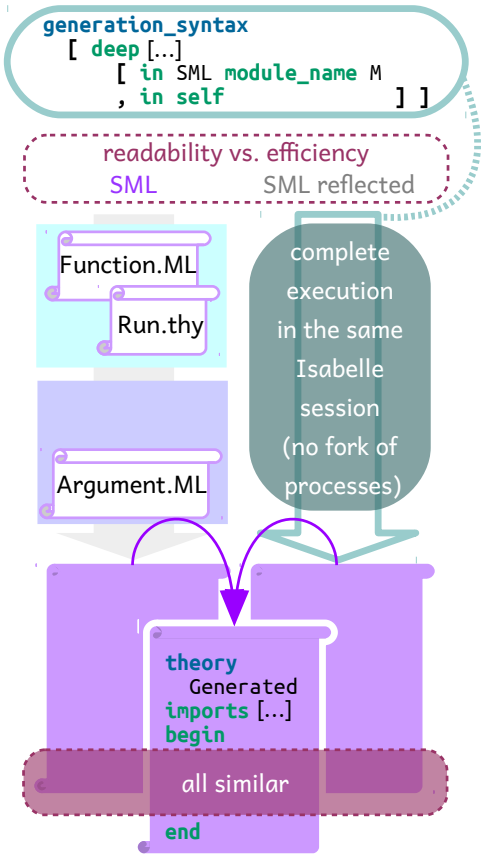


Figure 6.5: External target versus internal target in deep

As remark, a potential restoration of previous functionalities can simultaneously concur with the existing compiling schemes: for example we can have several active modes of compilation for Haskell, OCaml and SML, for compiling in byte-code and in native-code at the same time. This would all the more increase the testing activity for the benefit of not only respective compilers, but also code serializing function. In particular, we identified syntactic issues concerning the code generation of Isabelle to OCaml and Scala. These issues have been signalled, and fixed for the release of Isabelle 2016[4,5]. However on the other hand, Scala issues turned to be somehow useful for cleaning the meta-compiler: ghost (unused) functions was in certain conditions not correctly extracted in Scala (in this case, an error was explicitly raised).

In a determined attempt to combine efficiency and safety, we propose a third optimizing scenario. The new option self, represented in Figure 6.5, can be alternatively used in the list of target languages given to deep, besides the SML target for example. The target self resembles to the target SML: they ultimately perform the generation of the deep-certificate to the hard disk. Whereas for SML the

[4] http://isabelle.in.tum.de/repos/isabelle/rev/774752af4a1f
[5] http://isabelle.in.tum.de/repos/isabelle/rev/8e736ce4c6f4

meta-translation function (in "`Function.ML`") and incoming meta-commands (in "`Argument.ML`") have to be extracted after generation_syntax (and have to be repeatedly extracted for `Argument.ML`), for self nothing is extracted and all operations prior to the writing of the deep-certificate fully occur in RAM. Indeed, the environment of the Isabelle process running the implementation of generation_syntax already contains the reflected meta-translation function, since the reflection step occurs before the definition of generation_syntax. So for the case of the self target, it is enough to just execute the reflected meta-translation function from $\mathcal{L}$ to Isar_HOL, and pretty-print the resulting value to string, then to a file.[6] In particular, comparing with the target SML, we are saving here one fork of Isabelle process. However again, the arguments in favor or against an efficient execution particularly apply here as when we explained Figure 6.1: one possibility to justify what has been executed in RAM is to readably inspect the extracted function `Function.ML` (or `Argument.ML`), this is what the sole use of self as target can not provide. The code generator of Isabelle allowing to either export (with export_code) or reflect (with code_reflect) are nevertheless internally relying on a common algorithm, or same trusted computing base.

6.3 Higher-Order Meta-Commands

Besides meta-commands generating Isar_HOL commands, the collection of multiple HOL embedding presented in Figure 5.3, from one arbitrary language to another one, has implicitly suggested the notion of considering meta-commands as first-class citizen in HOL: so "meta-commands generating meta-commands". In deep-mode, this is particularly not a danger for meta-commands to generate themselves, whereas for shallow the recursion might not terminate. Indeed, the iterating process chaining the collection of multiple HOL embedding is defined recursively in ML just after the reflection step. However this does not mean that the chaining function itself, situated at ML side, can not be preliminary used in Isabelle/HOL before the reflection step, as the declaration of arbitrary constants is feasible with consts [Wen16b][7], associated with abstract instantiations in ML with code_printing [Haf16]. As a side note, this is how we have defined a pretty-printing process in HOL involving polymorphic cartouches [TW15].[8] Moreover, as the Turing completeness of ML has mainly been profited just for defining this chaining function (besides Isar_HOL binding and parsing from $\mathcal{L}$), we think it is the sole recursive function in the meta-compiler whose termination looks not straightforward to prove, but the setting seems already ready for such proof: to be potentially written in the same HOL level as the HOL level of the meta-compiler.

Generally, for meta-commands to generate themselves, the meta-tool must priorly support a form of automated call to generation_syntax beforehand, so

[6] Technically, we could have syntactically called this target "the target shallow" instead of "the target self". However for clarity reasons, we refrain to do so in this document.

[7] As remark, the type system can be weaken by mistake with "consts magic :: $\alpha \Rightarrow \beta$". As mentioned in the reference manual of Isabelle, for nearly ten years now [Wen16b, NFWP15]: *"It is at the discretion of the user to avoid malformed theory specifications!"*

[8] In the implementation, the translation on meta-models makes use of optional portions of ML code (and abbreviations) that can all be removed. The translation does not depend on the printing process which happens afterwards, as shown in Figure D.3

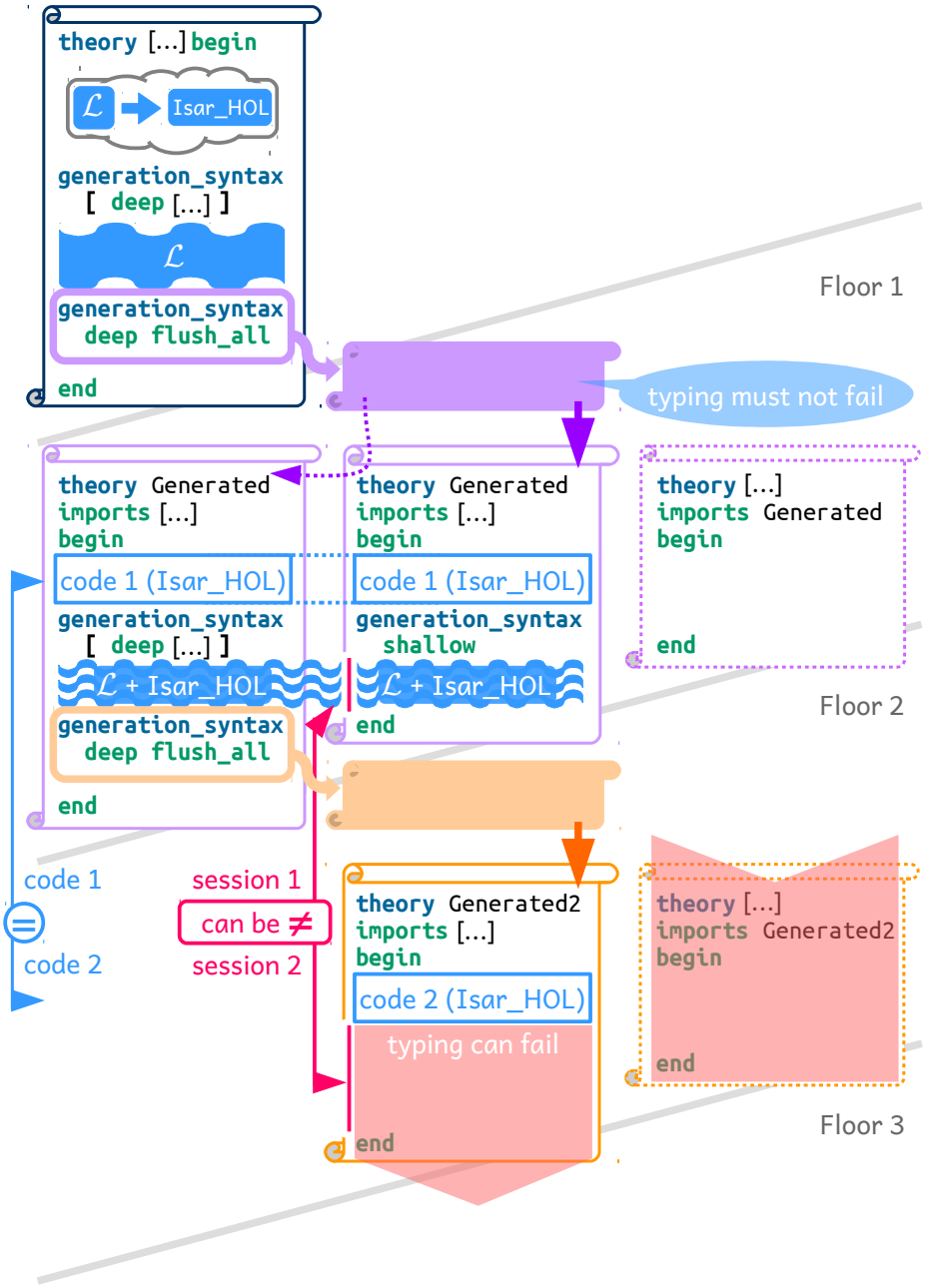


Figure 6.6: Multiple floors of generations in deep

that one can know which semantics to give to the newly created meta-commands. However this is not enough, the general compiling environment of Isabelle (behind the notion of session, and comprising the history of meta-commands) are changing throughout the interactive evaluations, so in certain situations the environment must also be taken into account and propagated when meta-commands are generating themselves. For example, Figure 6.6 shows an example where the environment is propagated across many levels of meta-generations.

Figure 6.6 divides the universe, seen as a *semantic tower*, into a set of disjoint partition, or *semantic floor*. (To our knowledge, the terminology "infinite (reflective) tower" in the domain of reflection came from the works of Brian Cantwell Smith [Smi82, Smi84]. Moreover, some noticeable characteristics of the "ground floor" of such towers have been presented for example by Bas R. Steunebrink and Jürgen Schmidhuber [SS12].) In the picture, the floor 3 is empty but the process can in principle be further continued. By reading from top to bottom, we start with a normal file in deep-mode, such as the one presented in Figure 6.1. Then after a step of deep-generation, the file `Generated.thy` finally appears in floor 1, with particularly inside a set of meta-commands. However to respect the commutative property of Figure 6.1, not only has this generated file the property to be well-typed, but we have automatically set it to be generated in shallow-mode (by default). Having an option in generation_syntax to force the generation towards a deep-file would be feasible as well. In the picture, after manually changing the mode of this file to deep, we extract a new theory `Generated2.thy` in floor 2. Let's assume this time it has inside zero meta-command.

As a design decision, and contrarily to `Generated.thy`, it is perfectly fine for the theory `Generated2.thy` to be not well-typed, precisely if `Generated.thy` contains Isar_HOL commands interleaved with $\mathcal{L}$, just after generation_syntax. This is because these generated Isar_HOL commands are *intentionally* not copied (or not produced any more) when the generation occurs from `Generated.thy` to `Generated2.thy`. As another possible choice, we could have chosen to explicitly do the copy but this assumes to transmit along particular information for `Generated.thy` to know what to copy. So it means to generate a code to dynamically modify the Isabelle environment and editing session, so that the environment of the meta-compiler can dynamically be modified as well. Generally, this modification must occur not only when jumping from floor i to floor $i + 1$, but an arbitrary floor n would generally need to know which Isar_HOL commands were generated from floor 1 to floor $n - 1$, so ultimately speaking, the knowledge of all generated Isar_HOL commands irrespective of floors. As remark and optimization privileging space than time, instead of propagating and remembering a set of (generated) Isar_HOL commands potentially large, the shorter list of meta-commands generating these Isar_HOL commands can be considered for the transmission.

- However, even if transmitting such information can have a certain cost, this kind of propagation of the environment across floors has been implemented, but not for all meta-commands. In practice in UML/OCL, we have not encountered serious situations where the (potential) failure of `Generated2.thy` would be an issue. Currently, the propagation has been implemented for only few meta-commands, i. e. those generating Isar_HOL commands (no meta-commands), that are particularly all situ-

ated before the first call to generation_syntax. For instance this includes meta-commands related with the Class Model Package: Class, Association, Composition, Aggregation; but this can also include meta-commands from the Instance Package, because in certain circumstances generation_syntax does not have to be immediately triggered after the Class Model Package. In Figure 6.6, code 1 (respectively code 2) represents the position where such generated code would occur. In particular code 1 is here equal to code 2, and generally, the generation of code 1 is automatically planned to be continued and repeated (at the beginning of generated files) following the creation of new semantic floors.

Generally, the transmission has been implemented for at least this particular case because in `Generated.thy` the generated content appearing after generation_syntax could sometimes contain zero Isar_HOL command. So this implies in this case that `Generated2.thy` would always be fully well-typed, thus we maximize situations where files are respecting Figure 6.1.

- Besides forcing the transmission for all meta-commands (particularly including those generating Isar_HOL commands situated after generation_syntax), there is another solution to overcome the limitation of the design decision. Instead of generating Isar_HOL commands, the solution would be to generate Isar_HOL' meta-commands, where Isar_HOL' has been bijectively mapped from all Isar_HOL command, by adding in their name at least one arbitrary symbol somewhere, so that all Isar_HOL' are syntactically all different from any regular Isar_HOL command. For example, we can introduce the following meta-commands which do not conflict with existing commands: datatype', definition', lemma', ML', etc... (Section 6.5 will particularly detail how to find suitable new names)

  Consequently, by using this technique, all deep-generated theories in all floors would be well-typed. However in the last floor, it would just remain to explicitly perform once more an additional step of generation from Isar_HOL' meta-commands to retrieve their associated Isar_HOL forms (to not say normal forms).

- As remark, while `Generated2.thy` could be not well-typed, it does not mean that all `Generated.thy`, having Isar_HOL commands after generation_syntax, will generate not well-typed file! In particular such Isar_HOL commands could have been written by hand or could be completely unrelated with the success or failure of `Generated2.thy`. Lemmas and proofs can most of the time be qualified as having such unrelated profile, in case their content are mostly involving pure computation not affecting the global context of Isabelle.

To effectively transmit our contextual information (with particularly the list of meta-commands generating Isar_HOL commands), we have used the Isar_HOL command setup [Wen16b], so that the global environment of Isabelle can be modified on the fly. However in certain circumstances, the command setup must be explicitly forced between some particular interleaving of two meta-commands C1 and C2, especially when C1 only generates Isar_HOL commands, so zero meta-command, and when C2 generates at least one meta-command (among potential Isar_HOL commands). Without an explicit use of setup, after C1,

the code generated by C2 would normally have no way to detect that some Isar_HOL code has been generated or not, precisely by C1. Consequently, one solution for C2, before generating its *first* meta-command, is to generate setup. In particular, this setup will increase the knowledge of C2 by instructing it of the existence of all Isar_HOL commands generated by C1.

Generally, generating meta-commands allows to perform various extensions on the language $\mathcal{L}$ being embedded, without altering the semantics of a particular command in $\mathcal{L}$. This is the picture we had when imagining a stack of semantic floors as a set of layers of a PTS in Section 5.3. For example, the UML/OCL meta-command Transition usually only takes "bound variables" as parameters (not arbitrary λ-terms), so something like "Transition σ_1 σ_2". However the semantics of Transition was extended to mimic the support of some particular terms not restricted to variables. This extension was implemented by executing some steps of "ζ-rewriting rules" [Coq16] operating on an upper meta-layer of semantic floor abstracting the floor where the semantics of Transition should usually be held accountable for. As an example of execution trace, we present a sequence of steps rewriting until normal form:

$$\text{Transition } [\bullet\,\bullet\,\bullet]\ \sigma_2 \rightsquigarrow \begin{array}{l}\text{State } \sigma_1 = [\bullet\,\bullet\,\bullet]\\ \text{Transition } \sigma_1\ \sigma_2\end{array} \rightsquigarrow \begin{array}{l}\text{Instance } X = \bullet\,\bullet\,\bullet\\ \text{State } \sigma_1 = [X]\\ \text{Transition } \sigma_1\ \sigma_2\end{array}$$

where "$\bullet\,\bullet\,\bullet$" represents a complex expression, normally only understood by Instance, and where σ_1 and X are fresh invented names. The particularity of the construction is that "$\bullet\,\bullet\,\bullet$" becomes implicitly supported by State and Transition as well, without having to program it, modulo some steps of meta-commands generating meta-commands. In the same spirit, "[$\bullet\bullet\bullet$]" becomes also supported by State.

As optimization, one can also implement a new keyword "nf" for meta-commands to know that they are acting as normal form meta-commands, and raise an error whenever one or several "$\bullet\,\bullet\,\bullet$" are given as arguments. So it means in fact to consider the following rewriting steps:[9]

$$\text{Transition } [\bullet\,\bullet\,\bullet]\ \sigma_2 \rightsquigarrow \begin{array}{l}\text{State } \sigma_1 = [\bullet\,\bullet\,\bullet]\\ \text{Transition (nf) } \sigma_1\ \sigma_2\end{array} \rightsquigarrow \begin{array}{l}\text{Instance } X = \bullet\,\bullet\,\bullet\\ \text{State (nf) } \sigma_1 = [X]\\ \text{Transition (nf) } \sigma_1\ \sigma_2\end{array}$$

6.4 Lazy Meta-Commands

In Isabelle, the responsivity of the editing engine globally participates to the interactive animation of the framework. Events occurring during the edition are continuously happening under various forms: pop-ups, diagnostic messages in the output window, asynchronous underlining of warning and errors. These suggest additional ideas to further align the meta-tool with the reactivity of Isabelle. In this part, we are going to all the more refine the animating aspect of the commutative diagram of Figure 6.1. Instead of visualizing $\mathcal{L}$ as a continuous text, we will describe the semantical effect of deep to the atomic level of meta-commands (thus similar properties will also hold for shallow).

[9] In HOL-OCL 2.0, we syntactically use the option "[shallow]" instead of "(nf)".

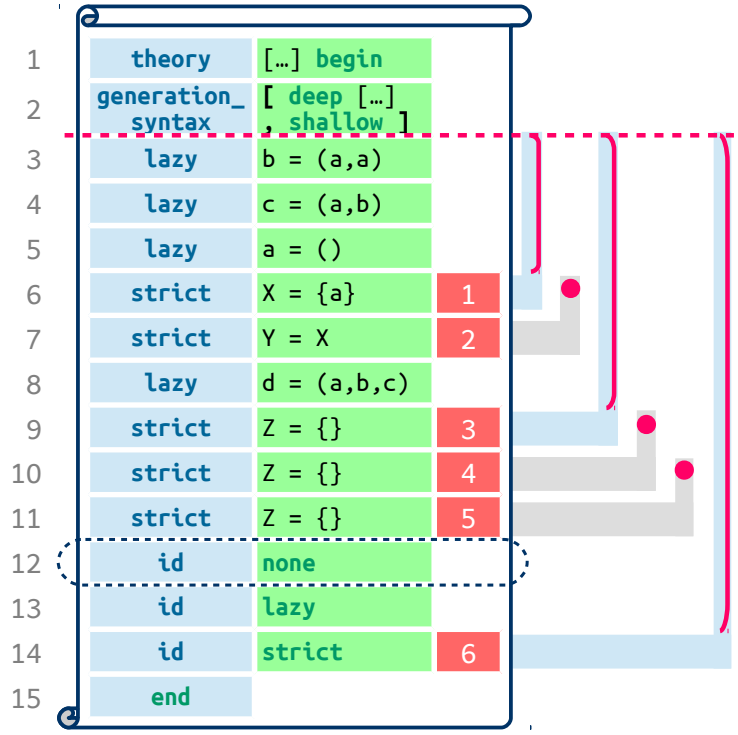


Figure 6.7: Rearranging the control flow of the prover

In Figure 6.2 we briefly saw that generation_syntax deep flush_all can be alternated among any meta-commands when experimenting a file in deep-mode. More precisely, for any generation_syntax deep flush_all being able to fold all meta-commands since the beginning, we needed to globally store all encountered meta-commands in the contextual environment of the meta-compiler. By generalizing the possibility to access this data-structure for any meta-commands (apart for generation_syntax), we obtain a new dimensional aspect in theorem proving, involving dynamic recomputation of meta-commands: namely "lazy meta-commands". Meta-commands are then getting grouped into two categories, depending on if they should be understood as supporting laziness or not. To emphasize that laziness is a dimensional feature independent of the default animating mode (deep or shallow), Figure 6.7 considers the activation of both deep and shallow at the same time. To simplify, we can restrict our presentation to only three meta-commands: lazy representing lazy meta-commands, strict standing as non lazy meta-commands, and id a form of exception meta-command to be described later.

Native Isar_HOL commands resemble to the family of strict because their side-effects are immediately visible and rendered, as soon as the asynchronous engine of the prover has a reason to require an effective evaluation. On the other hand, lazy meta-commands are specially skipped and their semantics are always getting frozen, irrespective of the prover. However they are not meaningless since lazy meta-commands impact incoming strict and id meta-commands.

- For example in Figure 6.7, lines 3-5 have been randomly permuted, because they are lazily declaring the variables a, b and c. So they are all ignored until we reach the next non lazy meta-command, like strict.
- At the position of cursor 1, the evaluation of strict $X = \{a\}$ will automatically force the evaluation of previous encountered lazy meta-commands.

  As remark, if we suppose line 3 removed, one could obtain either an error at cursor 1 or no particular errors: this is a simple design decision, depending on how the semantics of strict has been implemented during the embedding of $\mathcal{L}$ into Isar_HOL. In particular, the implementor can explicitly choose to raise an error, warning or nothing.
- Intuitively, line 7 gets evaluated as usual, since the previous meta-command was also strict.
- After another switching to lazy mode, cursor 3 needs to reconsider the evaluation of the entire set of meta-commands, as when we were at cursor 1. However as another design decision, strict could first consider the declarations of a, b, c and d together, *before* treating X and Y. Generally for any meta-command C, any permutating scenario happening before C can be considered, as long as the partial ordering of Figure 6.2 is respected by the implementation at any editing position, hence always producing an ordered increasing theories of well-typed elements.
- Because as any Isar_HOL command, ML can be generated, one can generate Isabelle/ML warnings or errors at cursors 4 and 5, since for example Z has already been defined at cursor 3. However, while in principle such errors can be directly raised in deep-mode, one design decision can delay the incoming of errors in deep-mode to the next semantic floor, i. e. generating errors to be triggered only when evaluated. By comparing with shallow, this would tend to consider the deep-mode as an experimenting framework where errors are minimized, thus inciting to do there arbitrary prototyping.

Besides the meta-commands lazy and strict, id is an example of meta-command where laziness can dynamically be parameterized with options situated in green areas, so "id lazy" stands for laziness, "id strict" as non-lazy. "id none" is an identity function combinator, where no effects are produced irrespective of the status of the previous command.

To better examine in deep-mode the list of Isar_HOL commands generated by a particular meta-command, we integrate in the meta-compiler a functionality to display in the output window the generated code associated to a meta-command, as illustrated in Figure 6.8. What to display in the output window is dynamically computed since for instance syntactically similar instructions, like lines 9-11, can actually generate different Isar_HOL commands (contrarily to line 10 and 11, line 9 has to take into account the history of all previous meta-commands, then includes itself). So the output window varies differently depending on the movement of the cursor when browsing the entire theory document. In addition, we augment the interactivity of the overall by mimicking the proof reconstruction tool sledgehammer [PS07, MP08, PB10, Bla16], so generated definitions and lemmas associated to a particular meta-command can be selectively inserted

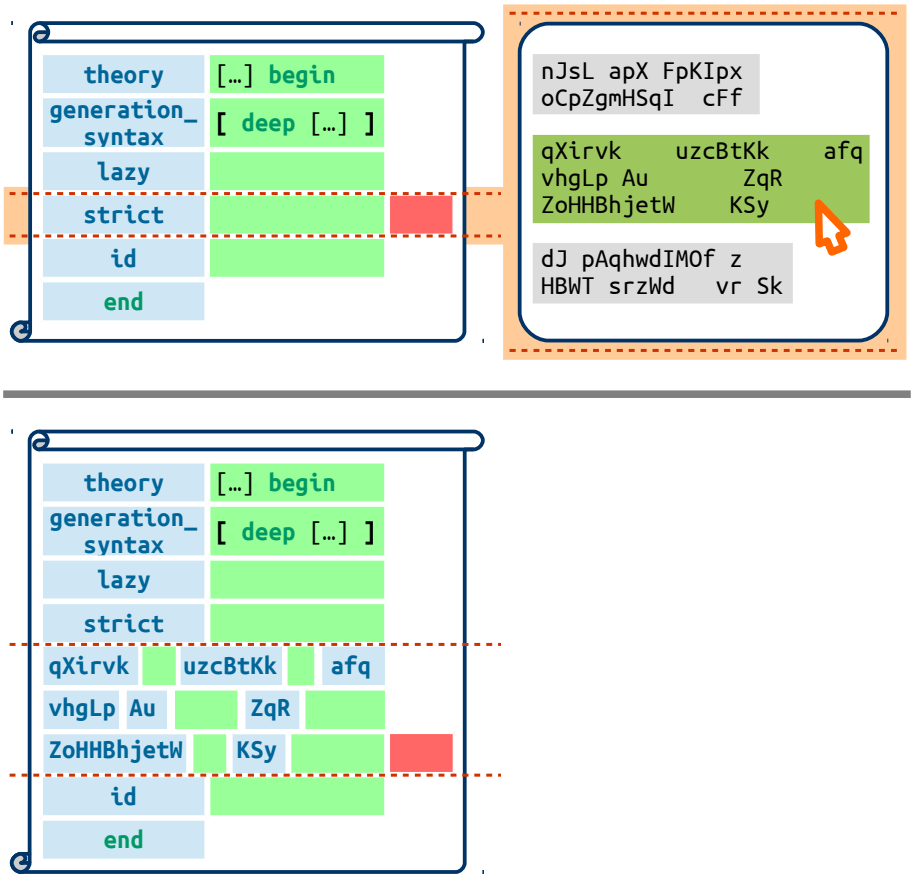


Figure 6.8: Inserting meta-commands from the output window

from the output window, by clicking on a chosen piece of content. The integration of this click-insert behaviour actually leads to various kind of programming scenarios.

- Besides a simple insertion, we imagine concurrently feasible (with a particular combination of keyboard) for a click to modify and replace the command where the cursor is situated. This would in a certain sense close the bootstrapping reproduction process, allowing to do in place, in full HOL, an upgrade of arbitrary code by means of meta-programming, and serve to complement the edition in Isabelle/jEdit. For example, one can manually type a huge λ-term and work on it by performing particular automatic editing operations. These operations are intended to be "fully" programmed in HOL (e. g. the renaming operation on all occurrences of a bound variable).

- Generally, in case the generated piece of content, chosen to be clicked in the output window, is embedding a setup command, then the current running global context of the editor would be transparently swapped (assuming the update function given to setup disregards its argument, containing the

state of the editing context). This is one way to travel across semantic floors using a depth-first exploration strategy, compared to the breadth-first strategy natively offered by generation_syntax deep flush_all. For the depth-first jump from one floor to another floor to not cause hazardous errors, we recommend to begin the experimentations with the deep-mode alone as side-effects would start minimized.

Finally, we come to the generalization of the laziness property to higher-order meta-commands. For example for each lazy meta-command C, we can define a new meta-command C', where C' is C but with an explicit flag to set the laziness or force the execution. So if the flag is meant to force the generation, C' would generate C and just after id strict; otherwise C' would only generate C.

- On the one hand, it suffices to natively force the execution of all C', for the output window to completely behave as fully animated in front of all meta-commands. So inside a theory file, the laziness property can be made imperceptible, i. e. each meta-command of a given file can be transformed into a non-lazy meta-command.

- On the other hand, in practice the laziness property can be implemented, for certain higher-order meta-commands, but without the need to cross the barrier of meta-commands generating meta-commands.

  For example in UML/OCL, laziness was required for Associationclass which aims to lazily generate what is lazily generated by both Class and Association. In the meta-tool, the code of Associationclass has been turned into an *implicit* lazy construction: a construction purely implemented in one level of HOL, without involving meta-constructions. Thus in deep-mode, the resulting effect of Associationclass (when encountering a next strict meta-command) is not to syntactically display both Class and Association in the output window, but the output window will show the code generated by both Class and Association. On the other hand, the termination is immediately guaranteed since it is a construction in pure HOL.

  Finally, we estimate our monadic construction enough abstract for it to be generalized to any lazy meta-commands requiring to generate meta-commands which are all non-lazy.

To conclude, laziness can be particularly useful to abstract end-users from certain characteristics of the languages being embedded: in UML/OCL, this allows us to incrementally declare classes at any editing position in the prover (declarations of classes are lazy) [TW15]. End-users would not know if the underlying logic is following an open-world or closed-world assumption, unless by monitoring the space and time consumed by resources at run-time. This is one drawback, the evaluation of a complete theory might have a certain cost, when laziness frequently occurs among non lazy meta-commands. Nevertheless, in USE all classes must normally be declared at the beginning of the file, before other expressions.

```
syntax "_OclForallSeq" :: "[('𝔄,'α::null) Sequence, _, _] ⇒ _"
       ("(_)->forAll'(_|_')")

syntax "_OclForallSet" :: "[('𝔄,'α::null) Set     , _, _] ⇒ _"
       ("(_)->forAll'(_|_')")

term "X->forAll(x|P x)"   (* ERROR: X is ambiguously
                                  parsed as a set
                                  and as a sequence
                                  at the same time    *)

term "(X :: ('𝔄,'α::null) Sequence)->forAll(x|P x)"
                          (*        X is a sequence     *)
term "(X :: ('𝔄,'α::null) Set     )->forAll(x|P x)"
                          (*        X is a set          *)
```

Figure 6.9: Syntactic ambiguities because of similar notations

6.5 Obfuscated Meta-Commands

For supporting the rather rich concrete syntax of OCL in a flexible and standard conform manner, the parser-combinator-based infrastructure of Isabelle prior to version 2014 is not powerful enough. For example, one needs to write `self .x` (note the space in front of the accessor) instead of `self.x`. Moreover, the operation definitions of the library [BTW14] need to make a compromise between readability and logical precision. For example, to facilitate type checking and avoid spurious errors during typing, the overloaded collection type OCL operation X`->forAll(`$x|P(x)$`)` has to be represented more precisely by the concrete instance X`->forAll`$_{\texttt{Set}}$`(`$x|P(x)$`)`, and usually implicit type coersions between sub- and supertype have to be written explicitly. Similarly, notations of OCL number and certain data-structures can happen to be slightly differently represented in HOL, depending on the range of symbols already used or available in Isabelle/jEdit. As illustration, we show a situation where an ambiguity error is expected to be raised in Figure 6.9, especially when the typing information is omitted.

To enable the writing of OCL expressions with a simpler and standard conform concrete syntax, we integrate a specific parser and type checker for OCL that was developed as part of su4sml [BDW06a] (and which is also used by HOL-OCL). The su4sml type inference injects type casts automatically and is implemented in SML using standard parser generator tools (i. e., ml-lex and ml-yacc). As su4sml is implemented in SML, it can directly be called from within the Isabelle/ML layer. Moreover, since version 2014, Isabelle supports a mechanism for defining dedicated parsers for domain specific languages, called *cartouches* [Wen16b][10], which we can plug su4sml into. Thus, even within logical HOL formulae, standard

[10] `https://en.wikipedia.org/w/index.php?title=Cartouche`

OCL syntax becomes possible, for example:

term "(λ *one*. ‹`self.clients->forAll(x|x.age>25)->size()`› ≐ *one*) ‹1›"

where the text between the ‹···›-markers (i. e. "U+2039"[11] and "U+203A"[12]) is handled by the su4sml parser and type inference. More generally, besides OCL, this mechanism can be used to nest arbitrary languages, provided the symbols "‹" and "›" are themselves not lexically present in the language being nested (balanced blocks of "‹" and "›" symbols are nevertheless permitted inside cartouches). The same remark holds for the quote symbol "`"`" (i. e. "U+0022"[13]), which is impossible to write in certain circumstances: assuming one has to delimit a string with this symbol, the writing becomes not possible inside cartouches particularly if the outermost enclosing delimiters of the overall expression are two "`"`" (as it is the case in the example, which is of the form term "···"). To overcome this limitation, Isabelle supports an alternative writing, i. e. where the outermost expression is of the form term ‹···›, so for example term ‹(λ *s*. ‹···› ≐ *s*) ‹"string"››. However this alternative writing is not enabled by default in Isabelle 2016: one has to redefine the command term in order to modify its parser to accept a cartouche as argument. In Isabelle 2014 and previous versions, such redefinitions of commands were permitted: e. g., definition could declare datatypes and datatype could declare definitions. However starting from Isabelle 2015 we obtain an error instead of a warning, so one solution is to manually introduce a not existing name, like the fresh name term'. Using this technique, we get the desired expression: term' ‹(λ *s*. ‹···› ≐ *s*) ‹"string"››.

At this point, the flexibility to nest arbitrary languages with cartouches appears enough for supporting OCL expressions, assuming one is using a recent version of Isabelle. Still, the presented technique does not mention how to proceed whenever, beyond OCL, the enclosing language $\mathcal{L}$ in cartouches has already reserved "‹" or "›" in its own syntax, or some escaping symbols which happen to be in potential conflict with Isabelle syntax (which can comprise for instance "`"`" and any symbols listed in the manual [Wen16b]). In particular, the meta-translation process of Chapter 5 explicitly manipulates the Isar_HOL meta-model, and concurrently we estimate feasible to take advantage of cartouches to enhance the readability of the translation and its presentation (by syntactically embedding the manipulated Isar_HOL language itself in special delimiting cartouches). Further investigations become then necessary to determine if conflicting symbols and delimiters of cartouches can syntactically be substituted with other symbols inside HOL expressions, or be made temporarily invisible whenever this is a relevant solution. To this end, we are now examining in more detail the range of symbols natively present in Isabelle.

As pointed in the manual of reference, symbols supported in Isabelle is potentially infinite [Wen16b], but their rendering are left to front-end tools (e. g., Isabelle/jEdit). Still, the final rendering is affected by external constraints. For example, although Isabelle/jEdit natively supports Unicode, the rendering of Unicode characters does not generally depend on the editing software drawing fonts, but on the original font specification where several characters could look identi-

[11] `http://unicode.org/cldr/utility/character.jsp?a=2039`
[12] `http://unicode.org/cldr/utility/character.jsp?a=203A`
[13] `http://unicode.org/cldr/utility/character.jsp?a=0022`

```
ML
val _ =
  Outer_Syntax.local_theory  @{command_keyword "datatype" }
    "define inductive datatypes"
    (BNF_FP_Def_Sugar.parse_co_datatype_cmd
      BNF_Util.Least_FP BNF_LFP.construct_lfp)
ML
val _ =
  Outer_Syntax.local_theory' @{command_keyword   "fun"    }
    "define general recursive functions (short version)"
    (Function_Common.function_parser Function_Fun.fun_config
      >> (fn ((config, fixes), statements) =>
            Function_Fun.add_fun_cmd fixes statements config))
ML
val _ =
  Outer_Syntax.command       @{command_keyword   "ML"     }
    "ML text within theory or local theory"
    (Parse.ML_source >> (fn source =>
      Toplevel.generic_theory
        (ML_Context.exec (fn () =>
          ML_Context.eval_source
           (ML_Compiler.verbose true ML_Compiler.flags) source) #>
         Local_Theory.propagate_ml_env)))
ML
val _ =
  Outer_Syntax.command       @{command_keyword   "end"    }
    "end context"
    (Scan.succeed
      (Toplevel.exit o Toplevel.end_local_theory o
        Toplevel.close_target o
         Toplevel.end_proof (K Proof.end_notepad)))
```

Figure 6.10: The genesis of commands as a half well-typed file

cal. As example, the Unicode characters “U+0430”[14] and “a” (i. e. “U+0061”[15]) are classified in the same set of confusing characters by `http://unicode.org`, this is the same for the character “U+041C”[16] and “M” (i. e. “U+004D”[17]). So the code shown in Figure 6.10 is only half well-typed in Isabelle 2016 because fun and **end** have already been defined earlier — in the real source code of Isabelle. On the other hand, datatype and ML have also already been defined in the source of Isabelle, but no errors are raised here because in this example the words *datatype* and *ML* are actually masking several symbols from the Cyrillic alphabet instead of the Latin alphabet, and their associated glyphs look similar as both originating from the Greek alphabet.

However, since equal glyphs might only be available in limited occurrences for a particular symbol, we have to get into a more uniform solution to cover all situations where it is desirable to have an “unlimited” number of abbreviations (and hence which are all looking close). This would permit to uniformly represent on the one hand UML/OCL collection operations on sets, sequences and bags, and on

[14] `http://unicode.org/cldr/utility/character.jsp?a=0430`
[15] `http://unicode.org/cldr/utility/character.jsp?a=0061`
[16] `http://unicode.org/cldr/utility/character.jsp?a=041C`
[17] `http://unicode.org/cldr/utility/character.jsp?a=004D`

```
syntax "_OclForallSeq" :: "[('𝔄,'α::null) Sequence, _, _] ⇒ _"
       ("(_)->forAll'(_|_')")

syntax "_OclForallSet" :: "[('𝔄,'α::null) Set     , _, _] ⇒ _"
       ("(_)->forAll'(_|_')")

term "X->forAll(x|P x)"   (*        X is a sequence     *)
term "X->forAll(x|P x)"   (*        X is a set          *)
term "[X->forAll(x|P x),
       X->forAll(x|P x)]" (* ERROR: X can not be a set
                                    and a sequence
                                    at the same time    *)
```

```
notation StrongEq    (infixl "=" 30) (* ≜ *)
notation StrictRefEq (infixl "=" 30) (* ≐ *)

term "(X = Y) = (X ≜ Y)"
term "(X = Y) = (X ≐ Y)"

definition "BOT = (invalid :: '𝔄 Integer)"
lemmas [simp] = BOT_def

lemma l1: "let X = BOT in X = Y ⟶ (τ ⊨ X = Y) ≠ (τ ⊨ X = Y)"
 apply(simp add: StrictRefEq_Integer StrongEq_def OclValid_def)
by (simp add: bot_option_def invalid_def true_def)

lemma l2: "let X = BOT in X = Y ⟶ (τ ⊨ X = Y) = (τ ⊨ X = Y)"
oops (* this is like proving False, the negation of l1 above *)

notation not_equal (infixl "=" 50) (* not_equal is defined in
                                      Isabelle in HOL.thy *)

lemma l3: "let X = BOT in X = Y ⟶ (τ ⊨ X = Y) = (τ ⊨ X = Y)"
by (rule l1)
```

Figure 6.11: `null` is null (invisible)

the other hand, regroup together arithmetic operations on integers and reals for instance. By examining the range of Unicode characters available in Isabelle, we have naturally retained our attention on two exception elements of the Unicode characters: `invalid` and `null`. In the domain of fonts, whereas "the meaning" of `invalid` will be detailed in Appendix K, `null` has generally the property to be an invisible symbol having a length of zero. Since at least 2009, this symbol can be used in Isabelle/jEdit like any whitespace at any string positions, thus also in the name of any commands and meta-commands. Then it becomes straightforward to add this invisible symbol to overload particular OCL operations, without having instead to search for particular representatives of equal glyphs. As example, in Figure 6.11, the two notations "`_->forAll`$_{\texttt{Set}}$`(_|_)`" and "`_->forAll`$_{\texttt{Seq}}$`(_|_)`" have been renamed, by replacing the string "$_{\texttt{Set}}$", respectively "$_{\texttt{Seq}}$", with two different `null`-strings. If we have written similar notations (or similar `null`-strings), errors are only expected to be raised when ambiguities are detected, as mentioned in Figure 6.9. However we are here in a different situation. In Figure 6.11, all abbreviations introduced with syntax (and notation) [Wen16b] are all different because they contain all different numbers of `null` symbol concatenated with itself. In particular, this is why the three introduced notations for `StrongEq`, `StrictRefEq` and `not_equal` are all different (besides having in common one extra symbol "=" of equality). As illustrated in the figure, invisible symbols do not mean weakening in the typing inference: the third term is not well typed because we are trying to consider at the same time X as a set and as a sequence.

As summary, we point certain similarities and differences between the introduction of abbreviations with Unicode characters and the use of cartouches. On the one hand, because ML is used to set up cartouches, one can program cartouches to parse and support Turing complete languages. This assumes nevertheless to know where the entry-points of the commands being redefined are located, as it is required for example to implement term' (such entry-points resemble to Figure 5.6). On the other hand, independently of cartouches, there are several commands in Isabelle to attach particular concrete syntax or notations to any manipulated constants or types (for the purpose of defining basic abbreviations), namely notation, syntax, translations, and also similar counterpart for abbreviations denoting types: type_notation, etc. Not significant lines of ML code are required, e. g., it is quite easy to introduce `null` in an abbreviation, as shown in Figure 6.11. Such so-introduced notations are immediately taking effect, so with this technique one can dynamically change the name of any constants, types or any meta-commands. Still, the arbitrary reconfiguration of Isar_HOL commands require some cares [NFWP15].

WARNING: In this document, both symbols `invalid` and `null` have not been used in this thesis and in the associated source code (only as example in Figure 6.11 and in Appendix K). Furthermore, unless it is *explicitly* explained in the text, all characters are understood as plain ASCII letters. This holds for Isar_HOL commands, and more generally for the entire source code of the project accompanying this thesis.[18] Generally, the inspection of the source (in Isabelle/jEdit) can already reveal the presence of characters outside the ASCII

[18] ACKNOWLEDGMENTS: The inspection of the range of Unicode symbols performed here was partly motivated by certain discussions arisen in the "*Isabelle Club*". *Isabelle Club* is a group meeting biweekly held at the LRI: `https://modhel.lri.fr/IsabelleClub/`.

range: when defining keywords, the use of quotes around keywords are normally mandatory, except for particular combinations of ASCII where this is optional. In Figure 6.10, we nevertheless chose to quote all the four keywords. So quotes can sometimes signal the presence of non-ASCII characters.

More generally, besides the use of cartouches in the meta-translation itself from $\mathcal{L}$ to Isar_HOL, one can also examine the possibility to propagate the editing environment context when editing the meta-translation, so that for example the colouring could be propagated inside cartouches in real-time, even if Isar_HOL is embedded in cartouches. Although the termination of higher-order meta-commands has to be manually brought, we would be nevertheless closer to the practice of "multi-stage programming in Isabelle".

CHAPTER 7

Object-Oriented Datatype Theories

In the following, we will refine the concepts of a user-defined data-model implied by a *class-model* (*visualized* by a class-*diagram*) as well as the notion of "($\mathfrak{A}$ :: object) state" used in Chapter 4 to much more detail (and also detail the concept of object). UML class models represent in a compact and visual manner quite complex, object-oriented datatypes with a surprisingly rich theory. In this chapter, this theory is made explicit and corner cases are pointed out.

7.1 Class Models

Our abstract syntax—called a *class model* following UML terminology—represents complex, object-oriented datatypes in a compact and viewable manner. Over such a class model, OCL invariants for states and OCL operation contracts for state transitions can be defined.

HOL

Definition "**Class model (user interface)**":
A class model is a four-tuple $(C, <, \mathit{PreAttrib}, \mathit{PreAssoc}, \mathit{mode})$ where:

- C is a set of class names written as $\{C_1, \ldots, C_n\}$. To each class name a datatype in OCL is associated,
- "_ < _" is a non-reflexive partial inheritance relation on classes, "_ $<^+$ _" its transitive closure, and "_ $<^*$ _" its reflexive transitive closure. "_ > _", "_ $>^+$ _", and "_ $>^*$ _" are their respective associated symmetric relations. As additional abbreviation, we introduce "X $<\!/\!>^*$ Y" for "$X \not<^* Y$ and $X \not>^* Y$".
- $\mathit{PreAttrib}(C_i)$ is a set of attributes associated to class C_i. Each attribute $a \in \mathit{PreAttrib}(C_i)$ declares two families of accessors, denoted by $X.a :: C_i \Rightarrow A$ and $X.a\,\texttt{@pre} :: C_i \Rightarrow A$ where $A \in \mathrm{TYPES}_0$,
- $\mathit{PreAssoc}(C_i, C_j)_{(S_i,S_j)}$ is a set of binary relations of the form $(n, rn_{\texttt{from}}, rn_{\texttt{to}}, ty)$ between two classes C_i and C_j where $S_i, S_j \in \{\texttt{Sequence}_m, \texttt{Set}_m\}$. The tuple consists of a (unique) association name n, two role names $rn_{\texttt{to}}$ and $rn_{\texttt{from}}$ and finally a command $ty \in$

{Association, Aggregation} that indicates if the pair $(rn_{\texttt{from}}, rn_{\texttt{to}})$ is initially intended by the user to be processed as an aggregation or not (in this case as an association). However this indication is not final: the option *mode* described below will also affect how the pair $(rn_{\texttt{from}}, rn_{\texttt{to}})$ will be processed, i. e., in conjunction with the value of *ty*. Like attributes a above, all role names declare two families of accessors, denoted by $X.a :: C_i \Rightarrow S_j(C_j)$ and $X.a\,\texttt{@pre} :: C_i \Rightarrow S_j(C_j)$ if $a = rn_{\texttt{to}}$. If the multiplicity associated to a evaluates to 1, both function types are $C_i \Rightarrow C_j$. For $a = rn_{\texttt{from}}$ we exchange i and j: both function types are $C_j \Rightarrow S_i(C_i)$.

- *mode* is an option to explicitly override how certain attributes and role names will be considered, to force them to be treated as associations or aggregations. Three values are possible for *mode*: design, analysis, or a default behaviour left to the meta-tool when nothing is provided. The conditions where these values have their effects are now detailed in the next definition.

Throughout the document, we will only consider finite class models with at least one element, called `OclAny`, the superclass of all classes: $C_i <^* \texttt{OclAny}$ (for all C_i).

Internally, the meta-tool will consider another intermediate version of the definition of class models. This version serves to further optimize the previous one and prepare for the separate treatments of associations and aggregations. In particular, one can represent class model in this intermediate version as a tree to enable a convenient folding of the global data-structure of classes. This is also where we will decide once for all how attributes and role names will be processed, i. e., as associations or aggregations. Indeed, a key idea of defining the semantics of UML and extensions like SecureUML [BDW06b] is to translate certain diagrammatic UML features into a combination of more elementary features of UML and OCL expressions [GR02]. For example, associations (i. e., relations on objects) can be implemented in specifications at the design level by aggregations, i. e., collection-valued class attributes together with OCL constraints expressing the multiplicity; and conversely, certain forms of aggregations can be simulated with associations.

HOL

Definition "**Class model (internal representation)**":
Besides the previous definition, class models can be simplified as $(C, <, \mathit{Attrib} \cup \mathit{Assoc})$, where the contents of *Attrib* and *Assoc* will depend on *mode* (described earlier). *Attrib* represents all attributes and role names intending to be considered by the meta-tool as aggregation relations, and *Assoc* represents all attributes and role names intending to be considered by the meta-tool as association relations. The definition proceeds by case analysis on *mode*.

If $\mathit{mode} = \text{design}$, then $\mathit{Attrib}(C_i)$ is defined as the union of the following items:

- attributes a in $\mathit{PreAttrib}(C_i)$
- role names a in $\{(\_, \_, a, \_) \longleftarrow \mathit{PreAssoc}(C_i, \_)_{(\_,\_)}\}$

- role names a in $\{(\_, a, \_, \_) \longleftarrow \mathit{PreAssoc}(\_, C_i)_{(\_,\_)}\}$

and $\mathit{Assoc}(C_i) = \{\}$.

Otherwise if $\mathit{mode} \neq$ design, we introduce a subset $A(C_i) \subseteq \mathit{PreAttrib}(C_i)$ such that $\mathit{Attrib}(C_i)$ is defined as the union of:

- attributes a in $\mathit{PreAttrib}(C_i)$ which are not in $A(C_i)$
- role names a in $\{(\_, \_, a, \text{Aggregation}) \longleftarrow \mathit{PreAssoc}(C_i, \_)_{(\_,\_)}\}$
- role names a in $\{(\_, a, \_, \text{Aggregation}) \longleftarrow \mathit{PreAssoc}(\_, C_i)_{(\_,\_)}\}$

and $\mathit{Assoc}(C_i)$ the union of:

- $A(C_i)$
- role names a in $\{(\_, \_, a, \text{Association}) \longleftarrow \mathit{PreAssoc}(C_i, \_)_{(\_,\_)}\}$
- role names a in $\{(\_, a, \_, \text{Association}) \longleftarrow \mathit{PreAssoc}(\_, C_i)_{(\_,\_)}\}$

If $\mathit{mode} =$ analysis, then $A(C_i)$ is defined as the union of:

- attributes a of the form $X.\, a :: C_i \Rightarrow C_j$ in $\mathit{PreAttrib}(C_i)$
- attributes a of the form $X.\, a :: C_i \Rightarrow S_j(C_j)$ in $\mathit{PreAttrib}(C_i)$

otherwise, if nothing is set for mode, we take by default $A(C_i) = \{\}$.

Finally, even if we have regrouped Attrib and Assoc together as $\mathit{Attrib} \cup \mathit{Assoc}$, the meta-tool will still keep the types of all attributes and role names stored next to attributes. This is to later decide, when having to process on an arbitrary $a \in \mathit{Attrib} \cup \mathit{Assoc}$, if a is actually an attribute or a role name (so to know its type, either $X.\, a :: C_i \Rightarrow A$ or $X.\, a :: C_i \Rightarrow S_j(C_j)$).

The definition of a class model gives rise to a number of induced operations which constitute the class model signature.

meta

Definition "**Class model signature**":
The signature associated to a class model $(C, <, \_)$ is the following:

- for all attributes a, role names a and class name $C_j \in C$ such that $X.\, a :: C_j \Rightarrow A$ and $X.\, a\,\texttt{@pre} :: C_j \Rightarrow A$ are well-formed for $A \in \text{TYPES}_0$, the two families (mentioned in the definition of class models) are exactly all expressions of the form:

  1. $X.\, a :: C_i \Rightarrow A$
  2. $X.\, a\,\texttt{@pre} :: C_i \Rightarrow A$

  for all $C_i \in C$ such that $C_i <^* C_j$,
- each class name $C_i \in C$ declares two projector functions to the set of all objects in a state: $C_i\texttt{.allInstances()}$ and $C_i\texttt{.allInstances@pre()}$,
- for each pair $C_i, C_j \in C$, there is a cast operation of type $C_i \Rightarrow C_j$ that can change the static type of an object of type C_i: $(X :: C_i).\,\texttt{oclAsType(}C_j\texttt{)}$,

- for each pair $C_i, C_j \in C$, there are two dynamic type tests:
  - $(X :: C_i).\,\texttt{oclIsTypeOf}(C_j)$ testing the dynamic type and
  - $(X :: C_i).\,\texttt{oclIsKindOf}(C_j)$ testing one subtype of the dynamic type,
- for each class name $C_i \in C$ there is an instance of the overloaded referential equality (written $\_ \doteq \_$).

Note on n-ary associations Given the fact that there is at present no consensus on the semantics of n-ary associations (for particularly $n \geq 3$), the following will mainly focus on situations where $n = 2$, on binary associations. A definition of class models supporting arbitrary n-ary associations can nevertheless be given by changing the above *PreAssoc* item with a more general version:

HOL

Definition "**Class model (user interface, generalized form)**":
A class model is a five-tuple $(C, <, \mathit{PreAttrib}, \mathit{PreAssoc}_n, \mathit{mode})$ where:

- C is [... same line as "Class model" ...]
- $\_ < \_$ is [... same line as "Class model" ...]
- $\mathit{PreAttrib}(C_i)$ is [... same line as "Class model" ...]
- mode is [... same line as "Class model" ...]
- For $n \geq 2$, $\mathit{PreAssoc}_n$ is a set of n-ary relations of the form (n', l_{rn}, ty).

The tuple consists of a (unique) association name n', a set of role names l_{rn} of cardinal n and a tag $ty \in \{\text{Association}, \text{Aggregation}\}$.

1. Each pair of different role names $(C_i, S_i, rn_{\texttt{from}})$ and $(C_j, S_j, rn_{\texttt{to}})$ in l_{rn} declare two families of accessors, denoted by $X.a :: C_i \Rightarrow S_j(C_j)$ and $X.a\,\texttt{@pre} :: C_i \Rightarrow S_j(C_j)$ if $a = rn_{\texttt{to}}$ where $S_i, S_j \in \{\texttt{Sequence}_m, \texttt{Set}_m\}$, and where the pair $(rn_{\texttt{from}}, rn_{\texttt{to}})$ will be processed by the meta-tool as an aggregation depending on ty. If the multiplicity associated to a evaluates to 1, both function types are $C_i \Rightarrow C_j$. For $a = rn_{\texttt{from}}$ we exchange i and j: both function types are $C_j \Rightarrow S_i(C_i)$.
2. More generally, for all (proper) non-empty subsets $l_{rn_{from}} \subset l_{rn}$ and $\{(C_j, S_j, rn_{\texttt{to}})\} \subseteq (l_{rn} \backslash l_{rn_{from}})$, if $l_{rn_{from}}$ of cardinal k has at least two elements, we declare two families of accessors of the form

   $(X_1, \cdots, X_k).\,a :: S_{i_1}(C_{i_1}) \Rightarrow \cdots \Rightarrow S_{i_k}(C_{i_k}) \Rightarrow S_j(C_j)$
   $(X_1, \cdots, X_k).\,a\,\texttt{@pre} :: S_{i_1}(C_{i_1}) \Rightarrow \cdots \Rightarrow S_{i_k}(C_{i_k}) \Rightarrow S_j(C_j)$

   for all k-permutations $[(C_{i_1}, S_{i_1}, rn_{\texttt{from}_1}), \cdots, (C_{i_k}, S_{i_k}, rn_{\texttt{from}_k})]$ of $l_{rn_{from}}$.

For $n \geq 3$, the type inference becomes no more decidable for expressions of the form $X.a :: \_ \Rightarrow \cdots \Rightarrow \_ \Rightarrow S_j(C_j)$ (for example an ambiguity relies in deciding if the type of an expression would be $C_{i_1} \Rightarrow S_j(C_j)$ or $C_{i_2} \Rightarrow S_j(C_j)$,

and particularly whenever $C_{i_1} = C_{i_2}$). One solution is to manually explicitly indicate which role names are involved, in particular these new syntaxes are used in HOL-OCL 2.0:

"$X.\, a$ /* $rn_{\texttt{from}_1} \cdots rn_{\texttt{to}}$ */ :: _ ⇒ $S_j(C_j)$" or
"$X.\, a$ /* $rn_{\texttt{from}_2} \cdots rn_{\texttt{to}}$ */ :: _ ⇒ $S_j(C_j)$".

The semantical problem for n-ary associations is to determine if we should only restrict to item "*1.*" (this is what is currently implemented in HOL-OCL 2.0) or also include item "*2.*" in the definition.

Running Example

The class model of the flight reservation example is the following tuple (C, $<$, *PreAttrib*, *PreAssoc*, *mode*) where:

- $C = \{\texttt{Person}, \texttt{Client}, \texttt{Staff}, \texttt{Flight}, \texttt{Reservation}, \texttt{OclAny}\}$
- $< = \{(\texttt{Staff}, \texttt{Person}), (\texttt{Client}, \texttt{Person}),$
  $(\texttt{Person}, \texttt{OclAny}), (\texttt{Reservation}, \texttt{OclAny}),$
  $(\texttt{Flight}, \texttt{OclAny})\}$
- $\begin{array}{lcl} \mathit{PreAttrib}(\texttt{Person}) & = & \{\texttt{name}\} \\ \mathit{PreAttrib}(\texttt{Flight}) & = & \{\texttt{seats}, \texttt{from}, \texttt{to}\} \\ \mathit{PreAttrib}(\texttt{Reservation}) & = & \{\texttt{id}\} \\ \mathit{PreAttrib}(\texttt{Client}) & = & \{\texttt{address}\} \\ \mathit{PreAttrib}(\texttt{Staff}) & = & \{\} \\ \mathit{PreAttrib}(\texttt{OclAny}) & = & \{\} \end{array}$

  In particular, we have $\mathit{PreAttrib}(\texttt{Client}) \neq \{\texttt{name}, \texttt{address}\}$ and $\mathit{PreAttrib}(\texttt{Staff}) \neq \{\texttt{name}\}$, because all inherited attributes (including `name`) are not yet processed at this moment, they will be accordingly computed later by the meta-tool.
- $\begin{array}{l} \mathit{PreAssoc}(\texttt{Person}, \texttt{Flight})_{(\texttt{Set},\texttt{Set})} \\ \quad = \{(i_1, \texttt{passengers}, \texttt{flights}, \text{Association})\} \\ \mathit{PreAssoc}(\texttt{Client}, \texttt{Reservation})_{(\texttt{Set},\texttt{Set})} \\ \quad = \{(i_2, \texttt{client}, \texttt{cl\_res}, \text{Association})\} \\ \mathit{PreAssoc}(\texttt{Flight}, \texttt{Reservation})_{(\texttt{Set},\texttt{Sequence})} \\ \quad = \{(i_3, \texttt{flight}, \texttt{fl\_res}, \text{Aggregation})\} \\ \mathit{PreAssoc}(\texttt{Reservation}, \texttt{Reservation})_{(\texttt{Set},\texttt{Set})} \\ \quad = \{(i_4, \texttt{next}, \texttt{prev}, \text{Association})\} \end{array}$

  where i_1, i_2, i_3, i_4 can be arbitrarily chosen, but have to be all different integers, e. g., respectively 0, 1, 2 and 3.
- *mode* can be arbitrarily chosen, e. g., we set it to be *None*.

For the attribute `seats` of `Flight` we have the two operations _.`seats` :: `Flight` ⇒ `Integer` and _.`seats@pre` :: `Flight` ⇒ `Integer`. For the association between `Client` and `Reservation`, we have the two operations _.`client` :: `Reservation` ⇒ `Client` and _.`cl_res` :: `Client` ⇒ `Set(Reservation)` (and

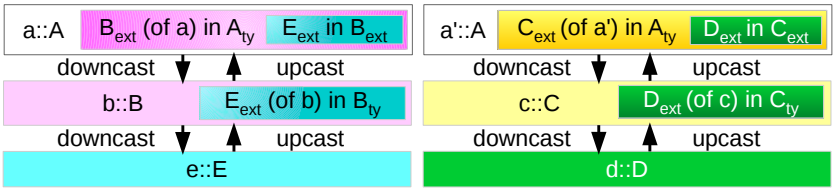


Figure 7.1: Casting in Universes.

the corresponding operations in the pre-state). As remark, `OclAny` has not been manually defined in Figure 3.2 because the meta-tool will implicitly include it.

As mentioned earlier, Featherweight OCL as semantic theory is organized as a "shallow embedding," which means that operators of the library and the datatype theory are represented by operators in Isabelle/HOL, such type representation of OCL types is one-to-one[1].

Inheriting from Isabelle/HOL a strong static type discipline in the sense of Hindley-Milner types, Featherweight OCL has no "syntactic subtyping." In contrast, subtyping can be expressed *semantically* in Featherweight OCL; by adding suitable casts which do have a formal semantics, subtyping becomes an issue of the front-end that can make implicit type coercions explicit. Our perspective on subtyping shifts the emphasis on the semantic properties of casting, and the necessary universe of object representations (induced by a class model) that allows to establish them.

7.2 A Denotational Space for Class Models

As a pre-requisite of a denotational semantics for operations induced by a class model, we need an *object universe* $\mathfrak{A}$ in which these operations can be defined denotationally and from which the necessary properties can be derived.

We represent objects with a type class object, they are identified with an *object id* (oid) under which it is referenced in the state, and OidOf ::(α::object) $\Rightarrow$ oid particularly returns the oid of any object. Objects are *statically typed* with class types, and under some additional conditions it is possible to approximate the equality on object representations by the equality of their references, i. e. by the referential equality. This section details now the cast of objects along the inheritance relation $\_ <^* \_$, in particular how to cast an object X of static type D up and down again in a semantically lossless way, whenever $D <^* C$:

$$(X :: D)\texttt{.oclAsType(}C\texttt{).oclAsType(}D\texttt{)} = X$$

Figure 7.1 presents the situation and sketches a solution: object representations need optional *object extensions* which remember the necessary information for consecutive up-down-casts to be idempotent. In addition, since object representations are designed to "live in a state", the type oid will also be included in the definition of class types. This gives rise to the following inductive definitions of *class types* C_i and *class type extensions* $C_{i\text{ext}}$.

[1] By slight abuse of language, arguments in parenthesis of the test and cast operations are always *class names* not types, e. g.: $((X :: \text{Staff})\texttt{.oclAsType(Person).oclAsType(Staff)}) = X$

meta

Definition "**Class type extensions (privileging accessors over casts)**":
Let C_i be a class with a possibly empty set of immediate subclasses $C_{j_1}, \ldots, C_{j_m}$ ($C_{j_l} < C_i$). Then

- the *class type extension* $C_{i\text{ext}}$ associated to C_i is $a_{i_1\underline{\perp}} \times \cdots \times a_{i_h\underline{\perp}} \times (C_{j_1\text{ext}} + \cdots + C_{j_m\text{ext}})_{\underline{\perp}}$ where a_{i_k} ranges over the local attribute types of C_i (not inherited ones) and $C_{j_l\text{ext}}$ ranges over all class type extensions of immediate subclasses C_{j_l} of C_i.

Here, $A + B$ denotes the sum type for the types A and B, such that $(C_{j_1\text{ext}} + \cdots + C_{j_m\text{ext}})_{\underline{\perp}}$ constructs the "potential alternative of one of the type extensions $C_{j_1\text{ext}}$ to $C_{j_m\text{ext}}$." As a consequence of the definition of class type extensions, we can now define class types (which depend on class type extensions):

meta

Definition "**Class types (privileging accessors over casts)**":
Let C_i be a class with a possibly empty set of immediate subclasses $C_{j_1}, \ldots, C_{j_m}$ ($C_{j_l} < C_i$). Then

- the *class type* $C_{i\text{ty}}$ for C_i is $\text{oid} \times a_{i_1\underline{\perp}} \times \cdots \times a_{i_n\underline{\perp}} \times (C_{j_1\text{ext}} + \cdots + C_{j_m\text{ext}})_{\underline{\perp}}$ where a_{i_k} ranges over the inherited *and* local attribute types of C_i and $C_{j_l\text{ext}}$ ranges over all class type extensions of immediate subclasses C_{j_l} of C_i.

Recall that this construction *cannot* be done in Featherweight OCL itself since it involves quantifications and iterations over the "set of class types". Then one can precisely use here the meta-tool detailed in Chapter 5 to overcome this limitation.

With respect to our semantic construction, which above all is intended to be type-safe, this has the following consequences:

- there is a generic theory of states, which must be formulated independently from a concrete object universe,
- there is a principle of translation (captured by the inductive scheme for class type extensions and class types above) that converts a given class model into a concrete object universe,
- there are fixed principles to derive the semantic theory of any concrete object universe, called the *object-oriented datatype theory.*

For *class type* and *class type extension*, this means to generate $\mathfrak{A}$ by the following scheme of Isabelle datatype definitions:

meta

$$
\begin{array}{lll}
\text{datatype } C_{i\text{ext'}} = \text{mk}_{C_i\_C_{j_1}} & C_{j_1\text{ext}} & \text{where } C_{j_1} < C_i \\
\quad \mid \cdots & \cdots & \cdots \\
\quad \mid \text{mk}_{C_i\_C_{j_m}} & C_{j_m\text{ext}} & \text{where } C_{j_m} < C_i \\
\text{datatype } C_{i\text{ext}} = \text{mk}_{C_i} \quad a_{i_1\perp} \cdots a_{i_h\perp} & C_{i\text{ext'}\perp} & \text{where } a_{i_1} \cdots a_{i_h} \text{ are owned} \\
\text{datatype } C_{i\text{ty}} = \text{mk'}_{C_i} \quad \text{oid } a_{i_{h+1}\perp} \cdots a_{i_n\perp} & C_{i\text{ext}} & \text{where } a_{i_{h+1}} \cdots a_{i_n} \text{ are inherited}
\end{array}
$$

$$\text{datatype } \mathfrak{A} = \text{in}_{C_k}\, C_{k\text{ty}} \mid \ldots \mid \text{in}_{C_l}\, C_{l\text{ty}}$$

The presented definitions of class types extensions and class types form the fundamental basis for some more involved operations on objects, such as accessor operations and cast operations. However as a design decision and due to our particular encoding, it becomes more efficient to apply attribute operations $(X :: C_i).a$ than cast operations $(X :: C_i).\texttt{oclAsType}(C_j)$. Alternatively, we can nevertheless provide an equivalent definitions of class types extensions and class types so that casts become conversely privileged over accessing operations, this is performed as follows:

meta

Definition "**Class types extensions and Class types (privileging casts over accessors)**":
Let C_i be a class with a possibly empty set of arbitrary subclasses $C_{j_1}, \ldots, C_{j_m}$ $(C_{j_l} <^+ C_i)$.

- Then the *class type extension* $C_{i\text{ext2}}$ associated to C_i is $oid \times a_{i_1\perp} \times \cdots \times a_{i_k\perp} + C_{j_1\text{ty2}} + \cdots + C_{j_m\text{ty2}}$ where a_{i_h} ranges over the inherited attribute types of C_i (not local ones) and $C_{j_l\text{ty2}}$ ranges over all *class types* of arbitrary subclasses C_{j_l} of C_i.
- Then the *class type* $C_{i\text{ty2}}$ for C_i is $C_{i\text{ext2}} \times a_{i_1\perp} \times \cdots \times a_{i_l\perp}$ where a_{i_n} ranges over the local attributes of C_i (not inherited ones) and $C_{i\text{ext2}}$ is the *class type extension* associated to C_i.

These definitions of class types extensions and class types look as being mutually recursive, however they are not actually: the meta-tool will implement these definitions by following a particular order of generation: from leaves first to the root as last node (which is `OclAny`). For example, the definition of class types of C_{j_l} will be generated before the definition of class types extensions of C_i.

As implementation remark, in the meta-tool all datatype encoding of class type extensions (i. e., $C_{i\text{ext}}$ and $C_{i\text{ext2}}$) and class types (i. e., $C_{i\text{ty}}$ and $C_{i\text{ty2}}$) have been formalized.[2] However to simplify the rest of the document, we will only take $C_{i\text{ext}}$ and $C_{i\text{ty}}$ as main definitions of class type extensions and class types (this choice is arbitrary). On the other hand, the formalization will consider $C_{i\text{ext2}}$ and $C_{i\text{ty2}}$ as definitions (e. g., in the generated code shown in Appendix B, $C_{i\text{ext2}}$ and

[2]For the moment, this comprises the generation of the respective datatype definitions, and all conversion functions between $C_{i\text{ty}}$ and $C_{i\text{ty2}}$. It remains to furthermore generate the equivalence proof between $C_{i\text{ty}}$ and $C_{i\text{ty2}}$ for all C_i.

$C_{i\text{ty}2}$ are respectively named $ty\mathcal{EXT}_{C_i}$ and ty_{C_i}, the generated code associated to $C_{i\text{ext}}$ and $C_{i\text{ty}}$ are present but not displayed in this present document).

Running Example

We show the definitions of class types and class type extensions of `Client` and `Person` from Figure 3.2. The construction of the universe comprises the following datatype definitions:[3]

HOL (generated)

$$
\begin{array}{llllllll}
\textcolor{blue}{\text{datatype}} & \text{Client}_{\text{ext}} & = & \text{mk}_{\text{Client}} & & \text{string}_{\lfloor} & & \\
\textcolor{blue}{\text{datatype}} & \text{Client}_{\text{ty}} & = & \text{mk'}_{\text{Client}} & \text{oid} & \text{string}_{\lfloor} & \text{Client}_{\text{ext}} & \\
\textcolor{blue}{\text{datatype}} & \text{Person}_{\text{ext'}} & = & \text{mk}_{\text{Person\_Staff}} & & & \text{Staff}_{\text{ext}} & \\
 & & | & \text{mk}_{\text{Person\_Client}} & & & \text{Client}_{\text{ext}} & \\
\textcolor{blue}{\text{datatype}} & \text{Person}_{\text{ext}} & = & \text{mk}_{\text{Person}} & & \text{string}_{\lfloor} & \text{Person}_{\text{ext'}\lfloor} & \\
\textcolor{blue}{\text{datatype}} & \text{Person}_{\text{ty}} & = & \text{mk'}_{\text{Person}} & \text{oid} & & \text{Person}_{\text{ext}} &
\end{array}
$$

$$
\begin{array}{llllllll}
\textcolor{blue}{\text{datatype}} & \mathfrak{A} & = & \text{in}_{\text{Flight}} & \text{Flight}_{\text{ty}} & | \text{in}_{\text{Client}} & \text{Client}_{\text{ty}} \\
 & & | & \text{in}_{\text{Staff}} & \text{Staff}_{\text{ty}} & | \text{in}_{\text{Person}} & \text{Person}_{\text{ty}} \\
 & & | & \text{in}_{\text{Reservation}} & \text{Reservation}_{\text{ty}} & | \text{in}_{\text{OclAny}} & \text{OclAny}_{\text{ty}}
\end{array}
$$

Here, $\text{oid} \times \text{string}_{\lfloor} \times \text{string}_{\lfloor\lfloor}$ is (the only) optional extension that represents `Client` objects cast to `Person`:

$$
\begin{aligned}
\text{Person}_{\text{ty}} &= \text{oid} \times \text{Person}_{\text{ext}} \\
&= \text{oid} \times \text{string}_{\lfloor} \times \text{Person}_{\text{ext'}\lfloor} \\
&= \text{oid} \times \text{string}_{\lfloor} \times \text{Client}_{\text{ext}\lfloor} \\
&= \text{oid} \times \text{string}_{\lfloor} \times \text{string}_{\lfloor\lfloor}
\end{aligned}
$$

In UML terminology (resp. in Java terminology), these are objects with dynamic type (resp. actual type) $\text{Client}_{\text{ty}}$ and static type (resp. apparent type) $\text{Person}_{\text{ty}}$.

7.3 Denotational Semantics of Accessors on Objects and Associations

Our choice to use a shallow embedding of OCL in HOL, thus having an injective mapping from OCL types to HOL types, results in type-safety of Featherweight OCL. Arguments and results of accessors are based on type-safe object representations and *not* oids. This implies the following scheme for an accessor:

1. The *evaluation and extraction* phase: the oid is extracted from the object representation.
2. The *dereferencing* phase. The oid is interpreted in the pre- or post-state.
3. The *selection* phase. The corresponding attribute is extracted from the object representation.
4. The *re-construction* phase. Converting oids or value representations containing oids back to object representations and values (sets, sequences...) containing object representations.

[3] In this chapter, the complete detail of each "Running Example" associated to the construction we are generating can be fully inspected in Appendix B and Appendix C.

The evaluation and extraction phase. If the argument evaluation results in an object representation, the oid is extracted; if not, `invalid` is reported.

```
HOL
definition eval_extract X f = (λτ. case X τ of ⊥      ⇒ invalid τ
                                                 propagating the exception
                                             | ⌊⊥⌋    ⇒ invalid τ
                                                 dereferencing a null value
                                             | ⌊⌊obj⌋⌋ ⇒ f (OidOf obj) τ)
```

The de-referencing phase ("heap" case or $Attrib(C_i)$ case). The oid is interpreted in the pre- or post-state, the resulting object is converted to the expected format. The exceptional case of nonexistence in the state yields `invalid`. For each class C_i, we have:

```
meta
definition deref_oid_{C_i} fst_snd f oid = (λτ. case heap (fst_snd τ) oid of
                                                 ⌊in_{C_i} obj⌋ ⇒ f obj τ
                                               | _              ⇒ invalid τ)
```

The operation yields undefined if *oid* is not interpretable in the state or referencing an object representation not conforming to the expected type.

The de-referencing phase ("assocs" case or $Assoc(C_i)$ case). In complement to general HOL notations, like for instance $f \circ g \equiv \lambda x.\ f\ (g\ x)$, we first introduce several shorthands for readability. Each association $(n, rn_{\texttt{from}}, rn_{\texttt{to}}) \in Assoc(C_i, C_j)_{(S_i,S_j)}$ can refer to the association name n from a particular role name $rn_{\texttt{from}}$ and $rn_{\texttt{to}}$ in input:

```
meta
definition n_assoc_{rn_from} = n
definition n_assoc_{rn_to} = n
```

As additional aliases, we define definition in_pre_state = fst (for first component), definition in_post_state = snd (for second component) and definition reconst_basetype = id (for identity function).

Following Section 4.5, we now encode binary associations as a set of pairs of the form $(rn_{\texttt{from}}, rn_{\texttt{to}})$. Given a particular role name rn, the retrieval of the associated $rn_{\texttt{from}}$ or $rn_{\texttt{to}}$ is performed symmetrically (either: first component to second or second component to first). The following definitions describe the accessing of such role names:

```
HOL
definition deref_assocs_list to_from oid S =
  concat (map (in_post_state ∘ to_from)
              (filter (λp. List.member (in_pre_state (to_from p)) oid) S))
```

```
definition deref_assocs_base pre_post to_from assoc_oid f oid =
           (λ τ.  case assocs (pre_post τ) assoc_oid of
                    ⌊S⌋  ⇒ f (deref_assocs_list to_from oid S)  τ
                  | _    ⇒ invalid                              τ)
```

```
meta
definition deref_assocs_rn fst_snd f =
           (deref_assocs_base fst_snd switch2_X n_assoc_rn f) ∘ OidOf
```

We provide for every pair all possible permutation functions: switch_2_01 and switch_2_10. While switch_2_01 is basically the identity function; switch_2_10 swaps the first component with the second one: as a consequence, if rn occurs at an $rn_{\texttt{to}}$ position, we set as convention $X = 01$; otherwise $X = 10$.

The selection phase. The corresponding attribute is extracted from the object representation. For each class C_i in the class model with at least one attribute, and each attribute a in this class, the selection phase is of this form:

- for inherited attributes a returning a base type:

```
meta
definition select_{C_i}_a f = (λ mk'_{C_i} oid ⋯ ⊥ ⋯ _ ⇒ null
                             | mk'_{C_i} oid ⋯ ⌊a⌋ ⋯ _ ⇒ f (λ x _. ⌊⌊x⌋⌋) a)
```

- for owned attributes a returning a base type:

```
meta
definition select_{C_i}_a f = (λ mk'_{C_i} _ ⋯ (mk_{C_i} ⋯ ⊥ ⋯ ) ⇒ null
                             | mk'_{C_i} _ ⋯ (mk_{C_i} ⋯ ⌊a⌋ ⋯ ) ⇒
                                                  f (λ x _. ⌊⌊x⌋⌋) a)
```

- for attributes a returning a "set" of object type (for "sequence" it is similar):

```
meta
definition select_a^set f =
                  X_a ∘ foldl OclIncluding^set mt^set ∘ map (f (λ x _. ⌊⌊x⌋⌋))
```

If the multiplicity of a allows to return at least two elements, $X_a = \text{id}$; otherwise we optimise by picking the only element with $X_a = \text{OclANY}^{\text{set}}$, which is the Hilbert's ϵ-operator. In particular, `null` is returned whenever the "set" is empty.

The re-construction phase. The resulting value has to be embedded in the adequate HOL type. If an attribute has the type of an object (not value), it is represented by an optional (set of) oid, which must be converted via dereferencing in one of the states to produce an object representation again. The exceptional case of nonexistence in this state must be treated.

Let _.getB be an owned accessor of class C_j yielding a value of base type $A \in \mathrm{T}_{base}$. Then its definition for every class $C_i <^* C_j$ is of the form[4]:

meta
overloading _.getB :: $C_i \Rightarrow A$
begin
definition X.getB = eval_extract X (deref_oid$_{C_i}$ in_post_state
(select$_{C_i_\mathrm{getB}}$ reconst_basetype))
end

Let _.getO be an owned accessor of class C_j yielding a value of object type C_k (or Set(C_k) depending on the returned type of select$^{\mathrm{set}}_{\mathrm{getO}}$). Then its definition for every class $C_i <^* C_j$ is of the form:

meta
overloading _.getO :: $C_i \Rightarrow C_k$ (or Set(C_k) depending on select$^{\mathrm{set}}_{\mathrm{getO}}$)
begin
definition X.getO = eval_extract X (deref_oid$_{C_i}$ in_post_state
(deref_assocs$_{\mathrm{getO}}$ in_post_state
(select$^{\mathrm{set}}_{\mathrm{getO}}$ (deref_oid$_{C_k}$ in_post_state))))
end

The variant for an accessor yielding a TYPES_0 is omitted here; its construction follows by the application of the principles of the former two. The respective variants _.a @pre are produced when in_post_state is replaced by in_pre_state.

Note on Multiplicities For an accessor returning a value of object type, situations of wrong multiplicities can statically be detected by a type-checking process (performed once). So no further checks are required during the access here, but only when object instances will be built (particularly with Instance in Section 8.2). Otherwise, the classical rules to convert multiplicities to invariants bounding the size of the collection types normally apply [BKW09].

Running Example

The de-referencing operation instantiated for the class Person is clear and will not be given here. We focus on the select functions:

HOL (generated)
definition
select$_{\mathrm{Person_name}}$ f = (λ mk'$_{\mathrm{Person}}$ _ (mk$_{\mathrm{Person}}$ $\bot$ _) $\Rightarrow$ null
| mk'$_{\mathrm{Person}}$ _ (mk$_{\mathrm{Person}}$ $\llcorner s \lrcorner$ _) $\Rightarrow$ f (λx _. $\llcorner\!\llcorner x \lrcorner\!\lrcorner$) s)
definition select$^{\mathrm{set}}_{\mathrm{flights}}$ f = id $\circ$ foldl OclIncluding$^{\mathrm{set}}$ mt$^{\mathrm{set}}$ $\circ$ map (f (λx _. $\llcorner\!\llcorner x \lrcorner\!\lrcorner$))

which gives the top-level definitions:

HOL (generated)
overloading _.name :: Person $\Rightarrow$ Integer
begin
definition X.name = eval_extract X (deref_oid$_{\mathrm{Person}}$ in_post_state
(select$_{\mathrm{Person_name}}$ reconst_basetype))
end

[4] We use an ad-hoc overloading mechanism for defining a family of functions, parameterised over C_i.

```
overloading _.flights :: Person ⇒ Set(Flight)
begin
definition  X.flights = eval_extract X (deref_oid_Person in_post_state
                                        (deref_assocs_flights in_post_state
                                        (select^set_flights (deref_oid_Flight in_post_state))))
end
```

7.4 Tests for Types and Casts

As a consequence of our decision to consider subtyping an issue to be solved by a static type checker, the semantic treatment of casts and dynamic types lie in the heart of the concept of object-orientedness of Featherweight OCL. We reduce subtyping to castability, and type tests allow for specifying exactly the semantics of operation calls. Although OCL has no constructors inside the language, objects can be constructed in HOL and can be specified via OCL operation contracts. The problem needs therefore to be solved that objects have an implicit dynamic ("actual") type, which is invariant under cast; whereas the returned static type (statically inferable, "apparent") of an object can differ from its type before cast.

First, let us consider dynamic type tests of the form X.`oclIsTypeOf(`C_j`)`. To implement a similar syntax in Featherweight OCL, we declare for each class C_j of the class model a constant X.`oclIsTypeOf(`C_j`)` of a too large type $\alpha \Rightarrow$ `Boolean`. These constants will be defined by a family of concrete instances for class pairs C_i, C_j.

```
meta
overloading
begin
definition (X :: C_i).oclIsTypeOf(C_j) ≡ (λτ. case X τ of
              ⊥                                          ⇒ invalid τ
            | ⌊⊥⌋                                        ⇒ true    τ
            | ⌊⌊mk'_Ci _ ⋯ (mk_Ci _ ⋯  ⊥  )⌋⌋            ⇒ true    τ  if C_i = C_j
            | ⌊⌊mk'_Ci _ ⋯ (mk_Ci _ ⋯ ⌊mk_Ci_Cj _⌋)⌋⌋     ⇒ true    τ  if C_i > C_j
            |   _      ⇒ X.oclAsType(C_i').oclIsTypeOf(C_j)  τ  if (1)
            | ⌊⌊_⌋⌋                                      ⇒ false   τ ) if (*)
end
```

where

(1) stands for "$C_i \not> C_j$ and $C_i >^+ C_j$", in this case we are computing $C_{i'}$ such that $C_i >^+ C_{i'} > C_j$ (like the definition of `oclAsType` below);

(*) stands for "$C_i \neq C_j$ and not (1), or there exists C_h such that $C_h <^+ C_i$."[5]

We now define a family of casts for any pairs C_i, C_j.

[5] Isabelle does not accept definitions where redundant clauses in the pattern matching are written (and already covered by preceding clauses).

```
meta
overloading
begin
definition (X :: C_i).oclAsType(C_j) ≡ (λ τ. case X τ of
              ⊥    ⇒ invalid                                          τ
            | ⌞⊥⌟   ⇒ null                                             τ
            | X    ⇒ X                                                          if (1)
            | ⌞⌞mk'_{C_i} oid a_1 ··· a_n X⌟⌟
                   ⇒ ⌞⌞mk'_{C_j} oid A_inh (mk_{C_j} A_own⌞mk_{C_j_C_i} X⌟)⌟⌟   if (2)
            | _    ⇒ X.oclAsType(C_j').oclAsType(C_j)                 τ    if (3)
            | ⌞⌞mk'_{C_i} oid A_inh(mk_{C_i} A_own⌞mk_{C_i_C_j} X⌟)⌟⌟
                   ⇒ ⌞⌞mk'_{C_j} oid a_1 ··· a_n X⌟⌟                          if (4)
            | _    ⇒ X.oclAsType(C_i').oclAsType(C_j)                 τ    if (5)
            | ⌞⌞_⌟⌟ ⇒ invalid                                         τ )  if (*)
end
```

(1) if $C_i = C_j$, we are returning the same object. As optimisation, the pattern matching is not required for behaving as an identity function.

(2) if $C_i < C_j$, we are up casting. Then we compute the set of attributes A_{own} (owned) and A_{inh} (inherited) such that $\{a_1, \cdots, a_n\} = A_{own} \uplus A_{inh}$ (disjoint union).

(3) if $C_i \not< C_j$ and $C_i <^+ C_j$, we are up casting. Then we compute $C_{j'}$ such that $C_i <^+ C_{j'} < C_j$.

(4) if $C_i > C_j$, we are down casting. Then we compute the merging $\{a_1, \cdots, a_n\}$ such that $A_{own} \uplus A_{inh} = \{a_1, \cdots, a_n\}$.

(5) if $C_i \not> C_j$ and $C_i >^+ C_j$, we are down casting. Then we compute $C_{i'}$ such that $C_i >^+ C_{i'} > C_j$.

(*) if $C_i > C_j$ or $C_i <\!\!>^* C_j$, we are raising an exception when the down cast of (4) operates on an unexpected type, or if we have a situation of incomparability.

While conditions from (1) to (5) are all disjoint, the last condition (*) applying for (4) is not redundant: whenever we have $C_i > C_j$, several patterns always exist for C_i. Finally as completeness, condition (*) is also needed whenever all conditions from (1) to (5) are not satisfied. As remark, conditions (*) and (5) are not present together at the same time but since (5) calls (4) internally, then (5) eventually reaches (*). Although clauses (3) and (5) seem recursive, they are not actually, we are calling other overloaded definitions. As a consequence, when generating these definitions with the meta-tool, this implies a particular order of generation to follow. For example, to execute (3), we must have priorly defined both $(\_ :: C_i)$`.oclAsType(`$C_{j'}$`)` and $(\_ :: C_{j'})$`.oclAsType(`C_j`)`. Here the intermediate class $C_{j'}$ is arbitrary, we could have chosen as in (5) one $C_{j''}$ such that $C_i < C_{j''} <^+ C_j$ (as long as at least one decrementing step of $\_ < \_$ is involved and corresponding overloadings already defined).

As one key-property of the object universe construction, the preservation of up down casting is directly implied by the definition, for all $C_i <^* C_j$:

meta
lemma $((X :: C_i).\texttt{oclAsType}(C_j).\texttt{oclAsType}(C_i)) = X$

Both definitions make tests and casts strict and neutral or idempotent on `null`:

meta
lemma $(\texttt{invalid} :: C_i).\texttt{oclIsTypeOf}(C_j) = \texttt{invalid}$
lemma $(\texttt{null} :: C_i).\texttt{oclAsType}(C_j) = \texttt{null}$
lemma $(\texttt{invalid} :: C_i).\texttt{oclAsType}(C_j) = \texttt{invalid}$
lemma $(\texttt{null} :: C_i).\texttt{oclIsTypeOf}(C_j) = \texttt{true}$

This is a slight deviation from the standard: `null` as argument should in general yield `invalid`. Since `null` is usually considered as one unique constant appearing in all types, we have technically one polymorphic constant `null`. To mimic the desired effect, the last equation is required. Another issue is that casts yield `null` for a `null`- argument (with the right static type). Since casts can appear everywhere, this is to avoid non intuitive effects. Consider the case that X and Y have a distinct class type C_i and C_j. Then the OCL term

HOL
term $X \doteq \texttt{null and } Y \doteq \texttt{null and } X \doteq Y$

is either `false` or `invalid`, since $X \doteq Y$ is translated to $X.\texttt{oclAsType}(C_j) \doteq Y$ or $X \doteq Y.\texttt{oclAsType}(C_i)$ and thus to `invalid` if we apply, as required by the OCL standard, the rule `null.oclAsType(_)` = `invalid`.

Besides the lemmas on strictness and `null`-preservation, the relative position of C_i and C_j (in $C_i.\texttt{oclIsTypeOf}(C_j)$) reveals opposite consequences:

1. The type testing from a class C_i to a larger class C_j is always `false`. More precisely, for all classes $C_i <^+ C_j$ or $C_i <\!\!>^* C_j$:

   meta
   lemma $\tau \models \delta\ X \Longrightarrow \tau \models((X :: C_i).\texttt{oclIsTypeOf}(C_j)) \triangleq \texttt{false}$

2. When reversing the inheritance relation between C_i and C_j, as soon as a large class C_i *does* belong to the type of a small class C_j, the casting to C_j fails for all its subclasses. For all $C_i >^* C_j >^+ C_k$ (or whenever $C_i <\!\!>^* C_j$):

   meta
   lemma $\tau \models \delta\ X \Longrightarrow \tau \models(X :: C_i).\texttt{oclIsTypeOf}(C_j)$
   $\Longrightarrow \tau \not\models \upsilon\ (X.\texttt{oclAsType}(C_k))$

Altogether, these lemmas of type tests, casts, and their corner cases to definedness and `null` constitute the key properties of the object-universe construction, part of the object-oriented datatype theory.

Running Example

We instantiate the generic definitions for our example. For dynamic type tests, this leads to this concrete instance of the definition:

```
HOL (generated)
overloading
begin
definition (X :: OclAny).oclIsTypeOf(Person)                    ≡ (λτ. case X τ of
             ⊥                                                  ⇒ invalid τ
           | ⌊⊥⌋                                                ⇒ true    τ
           | ⌊⌊mk'OclAny _ (mkOclAny⌊mkOclAny_Person _⌋)⌋⌋      ⇒ true    τ
           | ⌊⌊_⌋⌋                                              ⇒ false   τ       )
end
```

For type casting, we similarly illustrate on a down casting example:

```
HOL (generated)
overloading
begin
definition (X :: OclAny).oclAsType(Person)                      ≡ (λτ. case X τ of
             ⊥                                                    ⇒ invalid τ
           | ⌊⊥⌋                                                  ⇒ null    τ
           | ⌊⌊mk'OclAny oid (mkOclAny⌊mkOclAny_Person X⌋)⌋⌋     ⇒
                                                                  ⌊⌊mk'Person oid X⌋⌋
           | ⌊⌊_⌋⌋                                                ⇒ invalid τ     )
end
```

In particular, we obtain the required casting properties:

```
HOL (generated)
lemma      τ ⊨ δ X
       ⟹ τ ⊨(X :: OclAny).oclIsTypeOf(OclAny)
       ⟹ τ ⊭ υ (X.oclAsType(Person))
lemma ((X :: Person).oclAsType(OclAny).oclAsType(Person)) = X
```

7.5 Tests for Kinds and Casts

While `oclIsTypeOf(`D`)` precisely checks if the dynamic type of an object is D, the operator `oclIsKindOf(`D`)` relaxes the query by only checking if that dynamic type belongs to one subtype of D. Given the fact that we assume closed-world semantics, a simple way to define the overloaded `oclIsKindOf` operation is by the disjunction:

```
meta
overloading
begin
definition (X :: C_i).oclIsKindOf(C_j) ≡ X.oclIsTypeOf(C_j) or
                    X.oclIsKindOf(C_k1) or ⋯ or X.oclIsKindOf(C_kn)
end
```

where $C_{k_1}, \ldots, C_{k_n}$ are all the immediate subclasses of C_j ($C_{k_l} < C_j$).

This leads to the usual rules of definedness and validity: for all classes C_i and C_j,

meta
lemma $\tau \models \upsilon\, X \Longrightarrow \tau \models \delta\, (X :: C_i).\texttt{oclIsKindOf}(C_j)$
lemma $\tau \models \delta\, X \Longrightarrow \tau \models \delta\, (X :: C_i).\texttt{oclIsKindOf}(C_j)$

1. Contrasting with the similar lemma of the previous section for `oclIsTypeOf`, the kind checking from a class C_i to a larger class C_j is always `true`. More precisely, for all classes $C_i <^* C_j$:

   meta
   lemma $\tau \models \delta\, X \Longrightarrow \tau \models (X :: C_i).\texttt{oclIsKindOf}(C_j)$

   We separate the proof of this lemma in two cases, depending on if $C_i = C_j$ or $C_i <^+ C_j$ because the proof of the latter will use the proof of the former in its own proof.

   meta
   proof If $C_i = C_j$, we begin by unfolding the definition of $(X :: C_i).\texttt{oclIsKindOf}(C_i)$. Then we obtain an expression of the form $A_1 \texttt{or} A_2 \texttt{or} \ldots \texttt{or} A_n$. Due to the abbreviation priority of $X \texttt{or} Y$, the expression becomes actually understood as $((A_1 \texttt{or} A_2) \texttt{or} \ldots \texttt{or}\, A_n)$. This has an importance since we are going to unfold all expressions A_k, with A_k of the form $(\_ :: \_).\texttt{oclIsKindOf}(\_)$. However the unfolding of A_k can only be performed in front of a state τ. So we apply at the same time the rule *cp-OclOr* everywhere, i. e. $(A_{k1} \texttt{or}\, A_{k2})\,\tau = ((\lambda \_.\, A_{k1}\,\tau) \texttt{or}\, (\lambda \_.\, A_{k2}\,\tau))\tau$ in front of all expressions A_k. Since we are (arbitrarily) proceeding from left to right, it means to generate a list of substitution of the form:

$$\text{apply}\left(\begin{array}{r} \mathit{subst}\ (1)\ \mathit{cp\text{-}OclOr}, [\ldots], \\ \mathit{subst}\ (2\ 1)\ \mathit{cp\text{-}OclOr}, [\ldots], \\ \mathit{subst}\ (3\ 2\ 1)\ \mathit{cp\text{-}OclOr}, [\ldots], \\ \vdots \\ \mathit{subst}\ (n-1\ n-2\ \ldots\ 1)\ \mathit{cp\text{-}OclOr}, [\ldots] \end{array}\right)$$

   where $[\ldots]$ corresponds to the piece of tactics unfolding the corresponding $(\_ :: \_).\texttt{oclIsKindOf}(\_)$ expression. At the end, we only obtain a general expression of the form $(((B_1 \texttt{or}\, B_2) \texttt{or}\, \ldots) \texttt{or}\, B_m)$ with all B_l of the form $(\_ :: \_).\texttt{oclIsTypeOf}(\_)$. Thus the proof terminates with

$$\text{apply}\left(\begin{array}{lr} \mathit{auto\ simp}: & \mathit{cp\text{-}OclOr}[\mathit{symmetric}]\ \mathit{foundation16} \\ & \mathit{bot\text{-}option\text{-}def} \\ & \mathit{OclIsTypeOf\text{-}C_{m_1}\text{-}C_i}\ \ldots \\ & \qquad\qquad \mathit{OclIsTypeOf\text{-}C_{m_N}\text{-}C_i} \\ \mathit{split}: & \mathit{option.split} \\ & \mathit{ty^{ext}\text{-}C_{m_1}.split}\ \ldots\ \mathit{ty^{ext}\text{-}C_{m_N}.split} \\ & \mathit{ty\text{-}C_{m_1}.split}\ \ldots\ \mathit{ty\text{-}C_{m_N}.split} \end{array}\right)$$

where $OclIsTypeOf\text{-}C_j\text{-}C_i$ is the definition of $(\_ :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{)}$, $ty\text{-}C_i.split$ and $ty^{ext}\text{-}C_i.split$ are respectively the splitting rules of class types and class type extensions of C_i, and the set of all C_{m_N} represents the subtree of C_i. At the end, whenever *auto* leaves some pending goals, the following simplification rule will ultimately terminate the proof:

$$\text{apply}\left(\left(\textit{simp-all add: false-def true-def } \textit{OclOr-def OclAnd-def OclNot-def}\right)?\right)$$

qed

meta

proof If $C_i <^+ C_j$, we begin as above, by unfolding the definition of $(X :: C_i)\texttt{.oclIsKindOf(}C_j\texttt{)}$. So by definition, we exactly obtain

$$X\texttt{.oclIsTypeOf(}C_j\texttt{) or } X\texttt{.oclIsKindOf(}C_{k_1}\texttt{) or } \cdots \texttt{ or } X\texttt{.oclIsKindOf(}C_{k_n}\texttt{)}$$

However, since we are *generically* generating this "meta"-proof from bottom to top for an increasing set of C_i and C_j, then it means we have already proved at some time in the past that $(X :: C_i)\texttt{.oclIsKindOf(}C_{k_l}\texttt{)}$ for exactly one C_{k_l} among $C_{k_1} \ldots C_{k_n}$. So it suffices to retrieve the name of this previously proved lemma. In particular, that name depends on if $C_i = C_{k_n}$ (in this case, we refer to the proof above) or not.

qed

2. When reversing the inheritance relation between C_i and C_j, we obtain the following property characterising an "unfolding" definition of `oclIsKindOf`. For all $C_i >^+ C_j$ and $\{C_{k_n} \mid C_j >^* C_{k_n}\}$:

meta

lemma 1: $$\tau \models \delta\, X \Longrightarrow \tau \models (X :: C_i)\texttt{.oclIsKindOf(}C_j\texttt{)} \Longrightarrow \tau \models X\texttt{.oclIsTypeOf(}C_{k_1}\texttt{)} \vee \cdots \vee \tau \models X\texttt{.oclIsTypeOf(}C_{k_n}\texttt{)}$$

On the other hand, as soon as a large class C_i *does not* belong to the kind of a small class C_j, the casting to C_j fails for all its subclasses. For all $C_i >^+ C_j >^* C_k$:

meta

lemma 2: $$\tau \models \delta\, X \Longrightarrow \tau \not\models (X :: C_i)\texttt{.oclIsKindOf(}C_j\texttt{)} \Longrightarrow \tau \not\models \upsilon\, X\texttt{.oclAsType(}C_k\texttt{)}$$

We prove this theorem by introducing an intermediate lemma performing an exhaustive case distinction, as illustrated in Figure 7.2: for all $C_i >^+ C_j$, let $K = \{C_{k_n} \mid C_i >^* C_{k_n} >^+ C_j\}$ such that we can construct $\{C_{l_m} \mid (C_k \in K) > (C_{l_m} \notin (K \cup \{C_j\}))\}$, then:

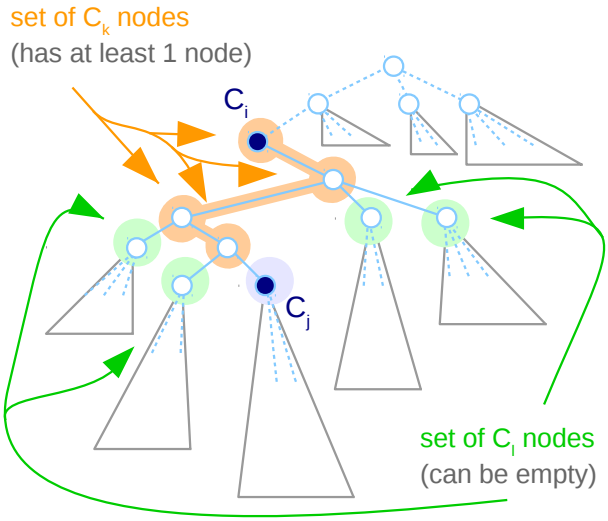


Figure 7.2: Example of nodes C_i, C_j, set of C_k and set of C_l following the hypothesis of the lemma

meta

lemma 2':
$$\begin{aligned} & \tau \models \delta\, X \Longrightarrow \tau \not\models (X :: C_i)\texttt{.oclIsKindOf(}C_j\texttt{)} \\ \Longrightarrow\ & \tau \models X\texttt{.oclIsTypeOf(}C_{k_1}\texttt{)} \vee \cdots \\ & \qquad \vee \tau \models X\texttt{.oclIsTypeOf(}C_{k_n}\texttt{)} \\ & \vee \tau \models X\texttt{.oclIsKindOf(}C_{l_0}\texttt{)} \vee \cdots \\ & \qquad \vee \tau \models X\texttt{.oclIsKindOf(}C_{l_m}\texttt{)} \end{aligned}$$

meta

proof We prove lemma 1 by first unfolding the expression $(X :: C_i)\texttt{.oclIsKindOf(}C_j\texttt{)}$: this leads to two cases depending on the number of immediate subclasses of C_j.

If C_j does not have immediate subclasses, then the proof is immediately finished since the unfolding gives exactly $(X :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{)}$.

Otherwise, if C_j has $m \geq 1$ immediate subclasses, we refine the so-obtained goal by m steps of elim-resolution rule because by definition of $(X :: C_i)\texttt{.oclIsKindOf(}C_j\texttt{)}$, we have an expression of the form $((((X :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{)}\ \texttt{or}\ A_1)\ \texttt{or}\ \ldots)\ \texttt{or}\ A_m)$. In particular, we repetitively apply m sequences of *erule* (*foundation26*[...]), where *foundation26* represents the rule

$$\begin{aligned} & \tau \models \delta\, P \Longrightarrow \tau \models \delta\, Q \Longrightarrow \\ & \tau \models P\ \texttt{or}\ Q \Longrightarrow (\tau \models P \Longrightarrow R) \Longrightarrow (\tau \models Q \Longrightarrow R) \Longrightarrow R \end{aligned}$$

and "[...]" designates the expression providing the proof of $\tau \models \delta P$ and $\tau \models \delta Q$ for *foundation26*[...] to be of the form

$$\tau \models P\ \texttt{or}\ Q \Longrightarrow (\tau \models P \Longrightarrow R) \Longrightarrow (\tau \models Q \Longrightarrow R) \Longrightarrow R$$

However, following the binary structure of $(((\ldots\ \texttt{or}\ A_1)\ \texttt{or}\ \ldots)\ \texttt{or}\ A_m)$, the proof expression "[...]" needs to be assembled as a binary tree as well. Then, we use at

each node the operator $defined\text{-}or\text{-}I : \tau \models \delta X \Longrightarrow \tau \models \delta Y \Longrightarrow \tau \models \delta(X \texttt{ or } Y)$ to chain the corresponding branches. We finally repeat this construction m times since we have m sequences of *erule* (*foundation26*[...]) to build:

- Step number m: we build the proof of $\tau \models \delta P$ and $\tau \models \delta Q$, where $P = ((((X :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{) or }A_1)\texttt{ or } \ldots)\texttt{ or }A_{m-1})$ and $Q = A_m$.
- Step number $m-1$: we build the proof of $\tau \models \delta P$ and $\tau \models \delta Q$, where $P = ((((X :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{) or }A_1)\texttt{ or } \ldots)\texttt{ or }A_{m-2})$ and $Q = A_{m-1}$.
- ...
- Step number 1: we build the proof of $\tau \models \delta P$ and $\tau \models \delta Q$, where $P = (X :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{)}$ and $Q = A_1$.

After applying all these sequences of *erule*, we finally obtain $m+1$ subgoals where the first is fast discharged since we have $(X :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{)}$ in both the assumption and the conclusion.

The remaining m subgoals uses the fact that we are *generically* generating this "meta"-proof from bottom to top for an increasing set of C_i and C_j. So we terminate by calling m times *drule* with the name of the adequate meta-proof (each *drule* becomes followed by a *blast*).
qed

Because the proof of lemma 2 uses the proof of lemma 2', we are first showing how to resolve this last.

meta
proof We prove lemma 2' by generating a list of tactics to sequentially apply: Figure 5.11 displays a recursive function aux_{depth} in HOL which precisely returns this list of tactics. At the beginning of the figure, we have included a minimal datatype modelling tactics, it has specially been simplified for this presentation. Then comes the mutually recursive functions aux_{depth} and $aux_{breadth}$. In input, the function aux_{depth} takes a tree data-structure, like the one shown in Figure 7.2. By convention, we assume that the tree initially given to aux_{depth} is truncated, where its root will represent C_i. This is without loss of generality since the result of type testing and kind testing only depends on nodes occurring in the subtree of C_i. More precisely, the subtree given to aux_{depth} is represented as an ordered list and contains all elements of K sorted according to the relation $\_ < \_$ (where $K = \{C_{k_n} \mid C_i >^* C_{k_n} >^+ C_j\}$, and where the first element of the list is C_i). In particular, the type of aux_{depth} is: $(\alpha \times (\beta \times bool)\, list)\, list \Rightarrow (\alpha, \alpha \times \beta\, list)\, T.tactic\, list$, and each element of the list in input is a pair where

- the first component α represents one node n of K,
- and the second component $(\beta \times bool)\, list$ contains as a list the collection of (immediate) subtrees β of n. In addition, the special boolean *bool* indicates if the root of the subtree β is in $K \cup \{C_j\}$ or not. So there is always only one element in the list which has an associated boolean equals to $True$.

As convention, we assume that this list of subtrees is sorted with the same order we have used when declaring class type and class type extensions. This is important since the order of tactics we will generate depends on the structure of declarations of class type and class type extensions.

Before detailing the tactics generated by aux_{depth}, we begin the proof of lemma 2' by adding as hypothesis the rule $\tau \models (X :: C_i)\texttt{.oclIsKindOf(}C_i\texttt{)}$ which has just been proved earlier in a previous lemma, and will name this rule H_i. This rule H_i has a central role here, as each recursive call of aux_{depth} is going to unfold the definition of $(\_ :: \_)\texttt{.oclIsKindOf(\_)}$. Consequently, we will cross during the overall proof a family of rules H_{class} of the form $H_{class} : \tau \models X\texttt{.oclIsKindOf(}class\texttt{)}$.

At the beginning, aux_{depth} first proceeds with a case distinction on the given list (representing K). Initially, this list is not empty since K is initially supposed to be not empty.

The other case in aux_{depth} concerns the unfolding of $H_{class} : X\texttt{.oclIsKindOf(}class\texttt{)}$, this is precisely the purpose of $T.simp\_only$, which takes as argument the name $class$ of the current class to unfold. After the unfolding, we obtain by definition

$$X\texttt{.oclIsTypeOf(}class\texttt{) or } X\texttt{.oclIsKindOf(}class_1\texttt{) or } \cdots \texttt{ or } X\texttt{.oclIsKindOf(}class_n\texttt{)}$$

where $class_1, \cdots, class_n$ are all the immediate subclasses of $class$ ($class_l < class$). The next step of the proof suspends the treatment of aux_{depth} by letting $aux_{breadth}$ continue the generation of the proof. $aux_{breadth}$ will particularly recursively fold the list of immediate subtrees $l_{breadth}$ of $class$. The new version of l_{depth} (in green) is also given as argument to $aux_{breadth}$, for aux_{depth} to resume the processing later. As remark, we reverse $l_{breadth}$ before calling $aux_{breadth}$ since this last will generate tactics in reverse order.

In the recursive body of $aux_{breadth}$, we retrieve the generation of a repetitive sequence of elim-resolution rule, this is similar as the proof of lemma 1 above. In particular the arguments given to $T.erule$ will be the information needed to generate the appropriate sequences of $erule$ ($foundation26[\dots]$).

After having generated the consecutive list of $T.erule$, the next tactic $T.simp_{breadth}$ (i. e. $simp$ or $blast$) will discharge the case where $(X :: C_i)\texttt{.oclIsTypeOf(}C_j\texttt{)}$ appears in both the assumption and the conclusion, similarly as the same situation in lemma 1 above.

Ultimately, we have two cases depending on the situation of $class0$.

- If $class0$ is in $K \cup \{C_j\}$, we continue to generate the list of tactics with aux_{depth} whenever $l_{depth} \neq \{\}$. After reaching the end of l_{depth}, we will have $class0$ equal to C_j. So it suffices to call $T.simp_{depth\_1}$, which will use the rule $\tau \not\models (X :: C_i)\texttt{.oclIsKindOf(}C_j\texttt{)}$, from the initial hypothesis of this lemma, to contradict with the rule H_j also present in the hypothesis.
- If $class0$ is not in $K \cup \{C_j\}$, we call $T.simp_{depth\_2}$ (i. e. $simp$ or $blast$) to explicitly use the current H_{class0}, which is present in both the assumption and the conclusion.

qed

meta
proof To prove lemma 2, we use lemma 2' by repeatedly eliminating all its disjunction after inserting it.

For each new subgoal, whenever we have as assumption a rule of the form $\tau \models (X :: C_i).\texttt{oclIsKindOf}(C_{l_m})$ (where C_{l_m} comes from the lemma 2'), we use *drule* applied with lemma 1.

At the end, we obtain as hypothesis a rule of the form $A_1 \vee A_2 \vee \ldots \vee A_j$ where all A_k are of the form $\tau \models (X :: C_i).\texttt{oclIsTypeOf}(C_k)$, with particularly $C_k <^+ C_i$. So we retrieve for each A_k the corresponding solving lemma from Section 7.4 (namely $\tau \models \delta\ X \implies \tau \models (X :: C_i).\texttt{oclIsTypeOf}(C_k) \implies \tau \not\models v\ (X.\texttt{oclAsType}(C_k))$), and terminate with "*auto simp* :" applied with this list of lemmas.
qed

Running Example

For our example, we obtain the following definition:

HOL (generated)
overloading
begin
definition $X.\texttt{oclIsKindOf(Person)} \equiv X.\texttt{oclIsTypeOf(Person)}\ \texttt{or}\ X.\texttt{oclIsKindOf(Staff)}\ \texttt{or}\ X.\texttt{oclIsKindOf(Client)}$
end

7.6 Access to the Global State

The operation `allInstances` builds the collection of all the instances of a given class in a state. With a little trick it is possible to define the global accessor on the state in a universal, generic (class model independent) way:

HOL
definition $\_.\texttt{allInstances()} :: V_{\mathfrak{A}}(\mathfrak{A} :: \underline{\text{object}} \Rightarrow (C_{\text{ty}})_\perp) \Rightarrow \texttt{Set}(C)$
where $(H.\texttt{allInstances()})\,\tau = \text{Abs\_Set}_{\text{base}}\ \llcorner\text{Some}$
$\quad ‘(\ (H\,\tau\,‘\,\text{ran}(\text{heap}(\text{in\_post\_state}\,\tau)))\ -\ \{\perp\}\ )\ \lrcorner\lrcorner$

meta
definition $C_{i\mathfrak{A}} = (\lambda\,\text{in}_{C_j}\,X \Rightarrow ((\lambda\_.\ \llcorner X \lrcorner\lrcorner) :: C_j).\texttt{oclAsType}(C_i))$
definition $C_i\ \tau\ X = (\text{case}\ C_{i\mathfrak{A}}\ X\ \tau\ \text{of}\ \llcorner X \lrcorner\lrcorner \Rightarrow \llcorner X \lrcorner \mid \_ \Rightarrow \perp)$

In our running example, this boils down to the following definitions:

HOL (generated)
definition $\text{Person}_{\mathfrak{A}} =$
$\quad (\lambda\ \text{in}_{\text{Person}}\ X \Rightarrow ((\lambda\_.\ \llcorner X \lrcorner\lrcorner) :: \texttt{Person}).\texttt{oclAsType(Person)}$
$\quad\ \mid\ \text{in}_{\text{Flight}}\ X \Rightarrow ((\lambda\_.\ \llcorner X \lrcorner\lrcorner) :: \texttt{Flight}).\texttt{oclAsType(Person)}$
$\quad\ \mid\ \ldots\ X \Rightarrow ((\lambda\_.\ \llcorner X \lrcorner\lrcorner) :: \ldots\ ).\texttt{oclAsType(Person)}$
$\quad\ \mid\ \text{in}_{\text{OclAny}}\ X \Rightarrow ((\lambda\_.\ \llcorner X \lrcorner\lrcorner) :: \texttt{OclAny}).\texttt{oclAsType(Person)})$
definition $\texttt{Person}\,\tau\ X = (\text{case}\ \text{Person}_{\mathfrak{A}}\ X\ \tau\ \text{of}\ \llcorner X \lrcorner\lrcorner \Rightarrow \llcorner X \lrcorner \mid \_ \Rightarrow \perp)$

It is easy to prove on the basis of these definitions, that our global accessors have "isKindOf"-semantics for any $C_i <^* C_j$:

meta
lemma $\tau \models C_i$`.allInstances()->forAll(`$X|X$`.oclIsKindOf(`C_j`))`

whereas the equivalent lemma for "isTypeOf"-semantics is only verified for C_i such that $\nexists C_h.\ C_h <^+ C_i$:

meta
lemma $\tau \models C_i$`.allInstances()->forAll(`$X|X$`.oclIsTypeOf(`C_i`))`

since we also prove for the others C_i (such that $\exists C_h.\ C_h <^+ C_i$):

meta
lemma $\exists \tau_1.\ \tau_1 \not\models C_i$`.allInstances()->forAll(`$X|X$`.oclIsTypeOf(`$C_i$`))`
lemma $\exists \tau_2.\ \tau_2 \models C_i$`.allInstances()->forAll(`$X|X$`.oclIsTypeOf(`$C_i$`))`

We found out that the current Annex A of the OMG standard actually defines the latter, while the mandatory part of the standard apparently favours the former. This inconsistency of the most recent standard (i. e., 2.4) is still to be resolved in a future version of the standard. We strongly suggest the `oclIsKind`-variant as it easily allows to use an additional type-selector construct in cases where the exact type set is required; this is not possible the other way round.

7.7 A Comparison to Related Work

Type and Kind Tests

Our formal semantics of OCL defines `null.oclIsKindOf(`C`)` to be true for all types C. This is on contrast to programming languages such as Java (and thus JML) or C# (and thus Spec#) which defines this to be false. While this decision is reasonable for a programming language as it avoids additional null-checks in case distinctions, it complicates verification as (X :: `Set(`D`))->forAll(`$X|X$`.oclIsKindOf(`D`))` would no longer be universally true.

Associations

Besides OCL, none of the mainstream object-oriented modelling languages supports associations (as relations) between objects and navigation over them as a first-class concept. This paves the way to a particular modelling methodology that is appealing to users. We consider it an advantage to have mathematical relations as an important concept both in real world scenarios as well as in the formal verification presented in this "user-friendly" way.

Equality

ACSL is the only language that has strong equality, simply since it has explicit pointers and no exception elements ("deep equality" has to be defined by hand). All other languages of our comparison know some form of strict equality. Both JML and Spec# have `null = null`, while `null <= null` interestingly yields $false$ in Spec#... Since JML and Spec# have explicit exception objects, they have a more concrete, more programming-oriented treatment of exceptional behaviour compared to `invalid` in OCL; this is also reflected in its equality.

Global Access

The operation `allInstances()` allowing for the access to the collection of all instances in the current state is a fairly original concept in UML/OCL reflecting its heritage from database modelling. In principle, its effect can be modelled in JML, Spec# in ghost-fields, which can also be used to model "sets of reachable objects" in a recursive data structure. However, if they cannot be constructed incrementally together with "ghost-code", this approach comes to a limit, since the use of recursive predicates is typically discouraged for methodological reasons (automated verification typically breaks down, being unable to provide some form of induction proofs).

Framing Conditions

All languages considered here have provided solutions to the problem, that the content of the post-state must be constrained to be equal to the pre-state in most cases; just the small portion of the memory that is updated by the function (to be specified) can be altered. A vast literature has been developed to address this problem ranging from region-like approaches as in ACSL, ownership approaches (one object or thread "owns" a set of other objects) as in Spec# to separation logics. Featherweight OCL proposes a `oclIsModifiedOnly()` predicate that states the set of objects that can change; all objects not in this set are identical between pre-and post-state. There is currently no solution and consensus in the community how to tackle associations.

Exceptional Behaviour

There is the possibility not to treat exceptions, let them occur as a consequence of illegal divisions 1/0, de-referencing `null` (as in `null.name`) or illegal oids (in pointers). Then, operations on them are underspecified. To exclude this, invariants and pre-conditions must be strengthened to permit reasoning only on specified behaviour (ACSL). The other extreme is specification and reasoning over explicit exceptions (Spec#, JML). OCL is in between these two extremes, having basically one exception `invalid` reflected in the logic. If we do not mention them in a pre-condition or an invariant, this leads typically to implicitly assume that they are excluded.

CHAPTER 8

Case Study

8.1 Corner Cases of Path Expressions

In this section, we illustrate the definitions of the previous section on a concrete example. Figure 8.1 shows two states (two object diagrams) instantiating Figure 3.1: before and after a reservation made by Arthur for a flight between Miami and Ottawa. Two reservations link clients Arthur and Bertha to flight `F1` from Valencia to Miami, with a staff Merlin onboard. After Arthur's reservation for flight `F2`, a new reservation links him to this flight. Moreover, his two reservations are part of the same journey therefore they are linked in order. Both states satisfy the invariants stated in Figure 3.3.

Corner Cases of Objects and Accessors

By loading Figure 8.1 in HOL-OCL 2.0, we can check arbitrary OCL path expressions. For instance, we have $(\sigma, \sigma') \models$ `C1.address` $\doteq$ "Saint-Malo", since Bertha will not be `invalid` nor `null` in "Saint-Malo" in the post-state (moreover the type of "Saint-Malo" belongs to T_{base}, so it is defined every time, independently of states). Before her move, we also have $(\sigma, \sigma') \models$ `C1.address@pre` $\doteq$ "Miami" since Bertha was defined at "Miami" in the pre-state. Similarly for Arthur, we have $(\sigma, \sigma') \models$ `C2.cl_res` $\doteq$ `Set{R21,R22}` since `R21` and `R22` will be all the reservations of the defined Arthur in the post-state, while $(\sigma, \sigma') \models$ `C2.cl_res@pre` $\doteq$ `Set{R21}` since Arthur was defined in the pre-state and he ordered only one reservation `R21` at that time.

We have a particular case with `R22` which will have no following reservation in the post-state: $(\sigma, \sigma') \models$ `R22.next` $\doteq$ `null`. Trying to de-reference a `null` association end yields an `invalid` value at any time, so $(\sigma, \sigma') \not\models \upsilon$ `R22.next.id`. As another `invalid` error, since `R22` did not occur in the pre-state, its de-referencing in this state necessarily fails: $(\sigma, \sigma') \not\models \upsilon$ `R22.id@pre`, and $(\sigma, \sigma') \not\models \upsilon$ `R22.flight@pre`. Let us point out that any empty association end yields `null`, even when the multiplicity is `*`. For instance `F2` had no reservation in the pre-state, therefore $(\sigma, \sigma') \models$ `F2.fl_res@pre` $\doteq$ `null`. In the USE tool for instance, `F2.fl_res@pre` is the empty set of reservations.

More complex expressions lead to other cases that are well-defined although not always intuitive. When an expression refers to only one state, the semantics

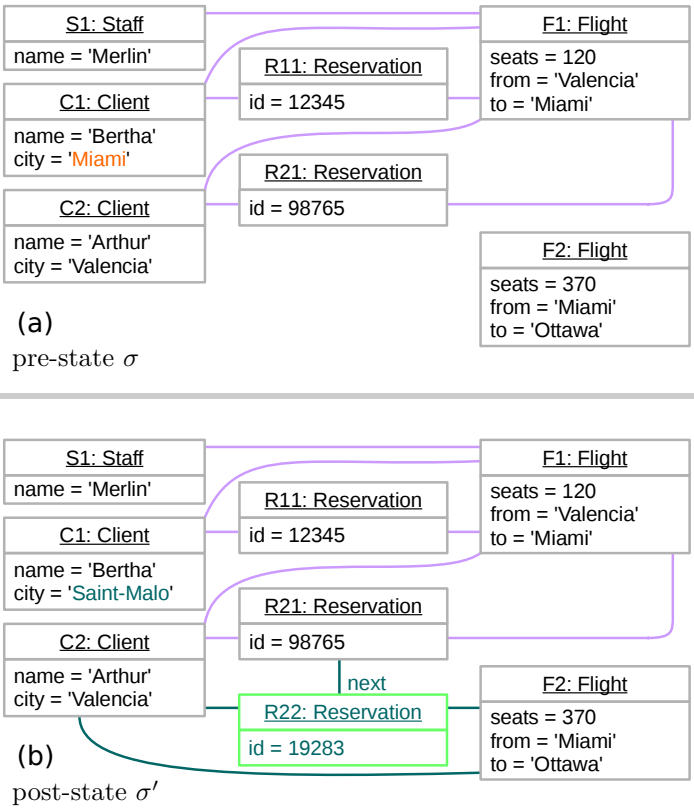


Figure 8.1: Two system states for the model of Figure 3.1.

remains easily comprehensible.

For instance, the following formulas are evaluated in the post-state only:

$\forall\, \sigma \quad . \quad (\sigma, \sigma') \models$ `R11.client.address` $\doteq$ "Saint-Malo"

$\forall\, \sigma \quad . \quad (\sigma, \sigma') \models$ `R21.next.flight` $\doteq$ `F2`

while those are evaluated in the pre-state only:

$\forall\ \ \sigma'. \quad (\sigma, \sigma') \models$ `R11.client@pre.address@pre` $\doteq$ "Miami"

$\forall\ \ \sigma'. \quad (\sigma, \sigma') \models$ `R21.flight@pre.passengers@pre` $\doteq$ `Set{S1,C1,C2}`

$\forall\ \ \sigma'. \quad (\sigma, \sigma') \models$ `R21.next@pre` $\doteq$ `null`

$\forall\ \ \sigma'. \quad (\sigma, \sigma') \not\models \upsilon$ `R21.next@pre.flight@pre`

A path expression involving both the pre and the post-state is for instance `R11.client@pre.address`. The client which reserved `R11` in the pre-state was Bertha, but her address will be "Saint-Malo" in the post-state:

$(\sigma, \sigma') \models$ `R11.client@pre.address` $\doteq$ "Saint-Malo"

Similarly for the path expression `R22.prev.client@pre`: in the post-state, the preceding flight of `R22` will be `R21`, but its client in the pre-state was Arthur:

$(\sigma, \sigma') \models$ `R22.prev.client@pre` $\doteq$ `C2`

Since `R22` did not exist in the pre-state, we also have that $(\sigma, \sigma') \models$ `R22.prev.next@pre` $\doteq$ `null` and $\forall \sigma'.\ (\sigma, \sigma') \not\models \upsilon$ `R22.prev@pre.next`.

Corner Cases of Types, Kinds and Casts

Now we consider an arbitrary state τ, since objects in states are not consulted for performing membership operations on type and kind, as well as (pure) casts. We also suppose having an object `P` of dynamic type `Person` (with `P` defined).

As demonstrated in Section 7.4, casting an instance of `Client` up to `Person`, then down to `Client` again returns the original object:

$\tau \models$ `C1.oclAsType(Person).oclAsType(Client)` $\doteq$ `C1`

However, casting an instance of `Person` down to `Client` is not possible if this instance is not a cast up of an instance of `Client`: $\tau \not\models v$ `P.oclAsType(Client)`.

We also saw in Section 7.4 that the `oclIsTypeOf` operator checks the dynamic type of an object while `oclIsKindOf` performs a weak form of dynamic check. This leads to the following properties (where $P =$ `Person` and $C =$ `Client`):

$\tau \models$ `P .oclIsTypeOf(`P`)` $\doteq$ `true`	$\tau \models$ `P .oclIsKindOf(`P`)` $\doteq$ `true`
$\tau \models$ `P .oclIsTypeOf(`C`)` $\doteq$ `false`	$\tau \models$ `P .oclIsKindOf(`C`)` $\doteq$ `false`
$\tau \models$ `C1.oclIsTypeOf(`P`)` $\doteq$ `false`	$\tau \models$ `C1.oclIsKindOf(`P`)` $\doteq$ `true`
$\tau \models$ `C1.oclIsTypeOf(`C`)` $\doteq$ `true`	$\tau \models$ `C1.oclIsKindOf(`C`)` $\doteq$ `true`

As expected, casting an instance of `Client` up to `Person` does not return an object of dynamic type `Person`:

$\tau \models$ `C1.oclAsType(Person).oclIsTypeOf(Person)` $\doteq$ `false`

In Section 7.6, the definition of `allInstances()` explicitly manipulates the post-state in τ given as parameter. By including the object `P` in σ' (the right-hand state of Figure 8.1), we obtain the following property for class `Client`: $\forall\sigma.\ (\sigma, \sigma') \models$ `Client.allInstances()` $\doteq$ `Set{C1,C2}`. For class `Person`, `allInstances()` returns all the instances of `Person` and of its child classes, while casting the latter up to `Person`, so that the result is a set of instances of `Person`:

$\forall\sigma.\ (\sigma, \sigma') \models$ `Person.allInstances()` $\doteq$ `Set{P,`
`C1.oclAsType(Person),C2.oclAsType(Person),S1.oclAsType(Person)}`

8.2 Specification Analysis of the Flight Model

In this section, we implement in Isabelle and HOL-OCL 2.0 the methodology of consistency analysis of specifications [BW09], instantiated here to the Flight Model example. All the code presented in the following has been generated, proofs are moreover not shown: the extended version with proofs can be inspected in Appendix A.

```
theory
  Flight-Model
imports
  ../src/UML-OCL

begin
```

Class Model

This part corresponds to the writing in Isabelle of the code shown in Figure 3.2.

```
Class Flight
  Attributes
    seats : Integer
    from : String
    to : String
End

lemma id = (λx. x)
⟨proof⟩
```

As remark, we are checking for example that the constant *id* already exists, and that one can also use this name in the following attribute: no conflict will happen.

```
Class Reservation
  Attributes
    id : Integer
    date : Week
End

Class Person
  Attributes
    name : String
End

Class Client < Person
  Attributes
    address : String
End

Class Staff < Person
End

Association passengers
  Between Person      [*]
            Role passengers
          Flight      [*]
            Role flights
End

Aggregation flights
  Between Flight      [1]
            Role flight
          Reservation [*]
            Role fl-res Sequence-
End
```

```
Association reservations
  Between Client      [1]
          Role client
        Reservation [*]
          Role cl-res
End

Association connection
  Between Reservation [0 . . 1]
          Role next
        Reservation [0 . . 1]
          Role prev
End
```

In complement to Figure 3.2, we define an enumeration type.

```
Enum Week
  [ Mon, Tue, Wed, Thu, Fri, Sat, Sun ]
End!
```

Two State Instances of the Class Model

The creation of (typed) object instances is performed in HOL-OCL 2.0 with the command Instance:

```
Instance S1 :: Staff = [ name = Merlin , flights = F1 ]
    and C1 :: Client = [ name = Bertha , address = Miami , flights = F1 ,
cl-res = R11 ]
    and C2 :: Client = [ name = Arthur , address = Valencia , flights = F1 ,
cl-res = R21 ]
    and R11 :: Reservation = [ id = 12345 , flight = F1 , date = Mon ]
    and R21 :: Reservation = [ id = 98765 , flight = F1 ]
    and F1 :: Flight = [ seats = 120 , from = Valencia , to = Miami ]
    and F2 :: Flight = [ seats = 370 , from = Miami , to = Ottawa ]
```

The notion of object instances comes before that of states. Currently, we have only created the object instances *S1*, *C1*, *C2*, *R11*, *R21*, *F1* and *F2*. They will need to be "registered" in a state later. Instance verifies that all objects being created are respecting the multiplicities declared above in classes (in the bidirectional sense). For example, after the type-checking stage, we have correctly that $R21\ .client \cong Set\{C2\}$, since *R21* appears as one reservation of *C2*, and where "$X \cong Y$" stands as a synonym for $\forall \tau.\ \tau \models \delta\ X \longrightarrow \tau \models \delta\ Y \longrightarrow \tau \models X \triangleq Y$.[1] As remark, the order of attributes and objects declarations is not important: mutually recursive constructions become de-facto supported. As illustration, we can include here the text displayed in the output window after evaluating the above Instance (we have manually pasted the text from the output window in Isabelle/jEdit):

$S1\ .flights \cong Set\{\ F1\ \}$
$C1\ .flights \cong Set\{\ F1\ \}$
$C1\ .cl\text{-}res \cong Set\{\ R11\ \}$
$C2\ .flights \cong Set\{\ F1\ \}$

C2 .cl-res ≅ *Set*{ *R21* }
R11 .flight ≅ *Set*{ *F1* }
R11 .client ≅ *Set*{ *C1* }
R11 .prev ≅ *Set*{}
R11 .next ≅ *Set*{}
R21 .flight ≅ *Set*{ *F1* }
R21 .client ≅ *Set*{ *C2* }
R21 .prev ≅ *Set*{}
R21 .next ≅ *Set*{}
F1 .passengers ≅ *Set*{ *S1* , *C1* , *C2* }
F1 .fl-res ≅ *Set*{ *R11* , *R21* }
F2 .passengers ≅ *Set*{}
F2 .fl-res ≅ *Set*{}

We can check that *S1* indeed exists and has the expected OCL type.

term *S1* ::· *Staff*

Once objects are constructed with Instance, it becomes possible to regroup them together into a state. This is what the next command State is doing by creating a state named σ_1, corresponding to the pre-state of Figure 8.1.

State σ_1 = [*S1*, *C1*, *C2*, *R11*, *R21*, *F1*, *F2*]

This generates a number of theorems from it, e. g.:

$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Staff .allInstances@pre*() ≜ *Set*{*S1*}
$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Client .allInstances@pre*() ≜ *Set*{*C1*,*C2*}
$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Reservation .allInstances@pre*() ≜ *Set*{*R11*,*R12*}
$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Flight .allInstances@pre*() ≜ *Set*{*F1*,*F2*}

At this point, it is not yet sure that σ_1 will be used in the pre-state or post-state. In any case, the above command also generates the following symmetric lemmas:

$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Staff .allInstances*() ≜ *Set*{*S1*}
$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Client .allInstances*() ≜ *Set*{*C1*,*C2*}
$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Reservation .allInstances*() ≜ *Set*{*R11*,*R12*}
$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Flight .allInstances*() ≜ *Set*{*F1*,*F2*}

Because all these lemmas are stated under the precondition that all object instances are defined entities, lemmas generated by State are actually proved in a particular locale [Bal14, Bal16] *state-*σ_1. Thus the header of *state-*σ_1 regroups these (mandatory) definedness assumptions, that have to be all satisfied before being able to use the rules defined in its body.

[1] Although such rule schemata may be explicitly generated by Instance (for most OCL expressions), they can also not be: at the time of writing, the complete type-checking process is at least fully executed from an extracted HOL function (as one consequence, the type-checking process terminates). This is feasible because for the moment, Instance only accepts "grounds objects" as arguments (the reader is referred to its syntax diagram detailed in Appendix I).

The next statement illustrates Chapter 6. It shows for instance that object instances can also be generated by State on the fly. Fresh variables are created meanwhile if needed, like σ_2-*object1*.

State σ_2 =
[*S1*
, ([*C1* with-only *name* = *Bertha*, *address* = *Saint−Malo* , *flights* = *F1* , *cl-res* = *R11*] :: *Client*)
, ([*C2* with-only *name* = *Arthur*,*address* = *Valencia*,*flights*=[*F1*,*F2*],*cl-res*=[self *4*,self *7*]]::*Client*)
, *R11*
, ([*R21* with-only *id* = *98765* , *flight* = *F1* , *next* = self *7*] :: *Reservation*)
, *F1*
, *F2*
, ([*id* = *19283* , *flight* = *F2*] :: *Reservation*)]

Similarly as with Instance, we can paste in the following what is currently being displayed in the output window (where "/∗8∗/" means the object having an oid equal to 8).[2]

σ_2-*object1* .*flights* ≅ *Set*{ /∗8∗/ }
σ_2-*object1* .*cl-res* ≅ *Set*{ /∗6∗/ }
σ_2-*object2* .*flights* ≅ *Set*{ /∗8∗/ , /∗9∗/ }
σ_2-*object2* .*cl-res* ≅ *Set*{ σ_2-*object4* , σ_2-*object7* }
σ_2-*object4* .*flight* ≅ *Set*{ /∗8∗/ }
σ_2-*object4* .*client* ≅ *Set*{ σ_2-*object2* }
σ_2-*object4* .*prev* ≅ *Set*{}
σ_2-*object4* .*next* ≅ *Set*{ σ_2-*object7* }
σ_2-*object7* .*flight* ≅ *Set*{ /∗9∗/ }
σ_2-*object7* .*client* ≅ *Set*{ σ_2-*object2* }
σ_2-*object7* .*prev* ≅ *Set*{ σ_2-*object4* }
σ_2-*object7* .*next* ≅ *Set*{}

Note that there is a mechanism to reference objects via the (invented) keyword self (it has no particular relation with the one used in Chapter 6), which takes a number designating the index of a particular object instance occurring in the list of declarations (the index starts with 0 as first position).

Similarly as for *state*-σ_1, we obtain another locale called *state*-σ_2, representing the post-state of Figure 8.1.

The Transition command relates the two states together.

Transition σ_1 σ_2

[2] As future work, it is plan for Instance to support the writing of arbitrary OCL expressions, including the assignment of potentially infinite collection types (for example "a set of sequence of bag of objects"). In particular, besides the cardinality of the manipulated collection types, the sole information required for checking multiplicities appears to be the oid of objects.

The first state is intended to be understood as the pre-state, and the second state as the post-state. In particular, we do not obtain similar proved theorems if we write Transition σ_1 σ_2 or Transition σ_2 σ_1 (assuming σ_1 and σ_2 are different). Generally, Transition establishes for a pair of a pre- and a post state (i.e. a state transition) that a number of crucial properties are satisfied. For instance, the well-formedness of the two given states is proven: $WFF(\sigma_1, \sigma_2)$.

Furthermore, for each object X additional lemmas are generated to situate X as an object existing in σ_1, σ_2, both, or in any permutations of σ_1 and σ_2. Such lemmas typically resemble as:

- $(\sigma_1, \sigma_2) \models X$.*oclIsNew*(), or
- $(\sigma_1, \sigma_2) \models X$.*oclIsDeleted*(), or
- $(\sigma_1, \sigma_2) \models X$.*oclIsAbsent*(), or
- $(\sigma_1, \sigma_2) \models X$.*oclIsMaintained*()

where the latter only means that the oid of X exists both in σ_1 and σ_2, in particular the values of the attribute fields of X have also not changed.

As completeness property, we can state the following lemma covering all disjunction case (for any X and τ) [BTW14]: $\tau \models \delta\ X \Longrightarrow \tau \models X.oclIsNew()$ $\vee\ \tau \models X.oclIsDeleted() \vee \tau \models X.oclIsMaintained() \vee \tau \models X.oclIsAbsent()$

Finally Transition proceeds as State: it builds a new locale, called *transition-*σ_1*-*σ_2, by particularly instantiating the two locales *state-*σ_1 and *state-*σ_2.

The following lemma establishes that the generated object presentations (like *S1* = (λ-. $\lfloor\lfloor S1_{Staff}\rfloor\rfloor$), *C1* = (λ-. $\lfloor\lfloor C1_{Client}\rfloor\rfloor$), etc.) satisfy the requirements of the locale *state-*σ_1. In particular, it has to be shown that the chosen object representations are defined and have distinct oids. Proving this lemma gives access to the already defined properties in this locale.

lemma σ_1: *state-interpretation-*σ_1 τ
⟨*proof*⟩

This instance proof goes analogously.

lemma σ_2: *state-interpretation-*σ_2 τ
⟨*proof*⟩

The latter proof gives access to the locale *transition-*σ_1*-*σ_2.

lemma σ_1-σ_2: *pp-*σ_1*-*σ_2 τ
⟨*proof*⟩

For convenience, one can introduce the empty state here

definition σ_0 :: $\mathfrak{A}$ *state* where σ_0 = *state.make Map.empty Map.empty*

so that the following abbreviations can be written

definition σ_{t1} = *transition-*σ_1*-*σ_2*.*σ_1 *oid3 oid4 oid5 oid6 oid7 oid8 oid9*
$\lceil\lceil S1\ (\sigma_0, \sigma_0)\rceil\rceil$ $\lceil\lceil C1\ (\sigma_0, \sigma_0)\rceil\rceil$ $\lceil\lceil C2\ (\sigma_0, \sigma_0)\rceil\rceil$ $\lceil\lceil R11\ (\sigma_0, \sigma_0)\rceil\rceil$

$$\lceil\lceil R21\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F2\ (\sigma_0, \sigma_0)\rceil\rceil$$

definition $\sigma_{t2} = \textit{transition-}\sigma_1\textit{-}\sigma_2.\sigma_2\ \textit{oid3 oid4 oid5 oid6 oid7 oid8 oid9 oid10}$
$\lceil\lceil S1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil \sigma_2\textit{-object1}\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil \sigma_2\textit{-object2}\ (\sigma_0, \sigma_0)\rceil\rceil$
$\lceil\lceil R11\ (\sigma_0, \sigma_0)\rceil\rceil$
$\lceil\lceil \sigma_2\textit{-object4}\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F2\ (\sigma_0, \sigma_0)\rceil\rceil$
$\lceil\lceil \sigma_2\textit{-object7}\ (\sigma_0, \sigma_0)\rceil\rceil$

definition $\sigma_{s1} = \textit{state-}\sigma_1.\sigma_1\ \textit{oid3 oid4 oid5 oid6 oid7 oid8 oid9}$
$\lceil\lceil S1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil C1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil C2\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil R11\ (\sigma_0, \sigma_0)\rceil\rceil$
$\lceil\lceil R21\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F2\ (\sigma_0, \sigma_0)\rceil\rceil$

definition $\sigma_{s2} = \textit{state-}\sigma_2.\sigma_2\ \textit{oid3 oid4 oid5 oid6 oid7 oid8 oid9 oid10}$
$\lceil\lceil S1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil \sigma_2\textit{-object1}\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil \sigma_2\textit{-object2}\ (\sigma_0, \sigma_0)\rceil\rceil$
$\lceil\lceil R11\ (\sigma_0, \sigma_0)\rceil\rceil$
$\lceil\lceil \sigma_2\textit{-object4}\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F1\ (\sigma_0, \sigma_0)\rceil\rceil\ \lceil\lceil F2\ (\sigma_0, \sigma_0)\rceil\rceil$
$\lceil\lceil \sigma_2\textit{-object7}\ (\sigma_0, \sigma_0)\rceil\rceil$

Both formats are, fortunately, equivalent; this means that for these states, we can access properties from both state and transition locales, in which the object representations are "wired" in the same way.

lemma $\sigma_{t1}\text{-}\sigma_{s1}$: $\sigma_{t1} = \sigma_{s1}$
⟨*proof*⟩

lemma $\sigma_{t2}\text{-}\sigma_{s2}$: $\sigma_{t2} = \sigma_{s2}$
⟨*proof*⟩

The next lemma becomes a shortcut of the one generated by Transition, but explicitly instantiated.

lemma $\textit{WFF}\ (\sigma_{t1}, \sigma_{t2})$
⟨*proof*⟩

lemma *F1-val-seatsATpre*: $(\sigma_{s1}, \sigma) \models \textit{F1}\ .\textit{seats@pre} \triangleq \text{«}\textit{120}\text{»}$
⟨*proof*⟩

lemma *F1-val-seatsATpre′*: $\sigma_{s1} \models_{pre} \textit{F1}\ .\textit{seats@pre} \triangleq \text{«}\textit{120}\text{»}$
⟨*proof*⟩

lemma *F2-val-seatsATpre*: $(\sigma_{s1}, \sigma) \models \textit{F2}\ .\textit{seats@pre} \triangleq \text{«}\textit{370}\text{»}$
⟨*proof*⟩

lemma *F2-val-seatsATpre′*: $\sigma_{s1} \models_{pre} \textit{F2}\ .\textit{seats@pre} \triangleq \text{«}\textit{370}\text{»}$
⟨*proof*⟩

lemma *F1-val-seats*: $(\sigma, \sigma_{s2}) \models F1\ .seats \triangleq «120»$
⟨*proof*⟩

lemma *F1-val-seats'*: $\sigma_{s2} \models_{post} F1\ .seats \triangleq «120»$
⟨*proof*⟩

lemma *F2-val-seats*: $(\sigma, \sigma_{s2}) \models F2\ .seats \triangleq «370»$
⟨*proof*⟩

lemma *F2-val-seats'*: $\sigma_{s2} \models_{post} F2\ .seats \triangleq «370»$
⟨*proof*⟩

lemma *C1-valid*: $(\sigma_{s1}, \sigma') \models (\upsilon\ C1)$
⟨*proof*⟩

lemma *R11-val-clientATpre*: $(\sigma_{s1}, \sigma') \models R11\ .client@pre \triangleq C1$
⟨*proof*⟩

Annotations of the Class Model in OCL

Subsequently, we state a desired class invariant for `Flight`'s in the usual OCL syntax:

Context f: *Flight*
Inv A : $\mathbf{0} <_{int} (f\ .seats)$
Inv B : $f\ .fl\text{-}res\ ->size_{Seq}() \leq_{int} (f\ .seats)$
Inv C : $f\ .passengers\ ->select_{Set}(p \mid p\ .oclIsTypeOf(Client)) \doteq ((f\ .fl\text{-}res)->collect_{Seq}(c \mid c\ .client\ .oclAsType(Person))->asSet_{Seq}())$

Model Analysis: A satisfiability proof of the invariants

We wish to analyse our class model and show that the entire set of invariants can be satisfied, i. e. there exist legal states that satisfy all constraints imposed by the class invariants.

lemma *Flight-consistent*: $\exists\ \tau.\ Flight\text{-}Aat\text{-}pre\ \tau \wedge\ Flight\text{-}A\ \tau$
⟨*proof*⟩

Context r: *Reservation*
Inv A : $\mathbf{0} <_{int} (r\ .id)$
Inv B : $r\ .next <>\ null\ implies\ (r\ .flight\ .to \doteq r\ .next\ .flight\ .from)$
Inv C : $r\ .next <>\ null\ implies\ (r\ .client \doteq r\ .next\ .client)$

Context *Client* :: *book* $(f : Flight)$
Pre : $f\ .passengers\ ->excludes_{Set}(self\ .oclAsType(Person))$
and $(f\ .fl\text{-}res\ ->size_{Seq}() <_{int} (f\ .seats))$

Post: f .passengers $\doteq$ (f .passengers@pre $->including_{Set}(self$ .oclAsType(Person)))
and (let $r = self$ .cl-res $->select_{Set}(r \mid r$ .flight $\doteq f)->any_{Set}()$ in
(r .oclIsNew())
and (r .prev $\doteq null$)
and (r .next $\doteq null$))

Context Client :: booknext (f : Flight, r : Reservation)
Pre : f .passengers $->excludes_{Set}(self$ .oclAsType(Person))
and (f .fl-res $->size_{Seq}() <_{int} (f$ .seats))
and (r .client $\doteq self$)
and (f .from $\doteq (r$ .flight .to))
Post: f .passengers $\doteq$ (f .passengers@pre $->including_{Set}(self$ .oclAsType(Person)))
and (let $r = self$ .cl-res $->select_{Set}(r \mid r$ .flight $\doteq f)->any_{Set}()$ in
(r .oclIsNew())
and (r .prev $\doteq r$)
and (r .next $\doteq null$))

Context Client :: cancel (r : Reservation)
Pre : r .client $\doteq self$
Post: $self$ .cl-res $->select_{Set}(res \mid res$ .flight $\doteq r$ .flight@pre)
$->isEmpty_{Set}()$

Context Reservation :: connections () : Set(Integer)
Post : result $\triangleq$ if ($self$ .next $\doteq null$)
then ($Set\{\}->including_{Set}(self$ .id))
else ($self$ .next .connections()$->including_{Set}(self$ .id))
endif
Pre : true

Proving the Implementability of Operations

An operation contract is said to be non-blocking, if and only if there exist input and input states where the pre-condition is satisfied. Moreover, a contract is said to be implementable, if and only if for all inputs satisfying the pre-condition output data exists that satisfies the post-condition.

definition $cancel_{pre} :: (\cdot Client) \Rightarrow (\cdot Reservation) \Rightarrow \cdot Boolean_{base}$
where $cancel_{pre}$ $self\ r \equiv (r$ .client@pre) $\doteq self$

definition $cancel_{post} :: (\cdot Client) \Rightarrow (\cdot Reservation) \Rightarrow (\cdot Void_{base}) \Rightarrow \cdot Boolean_{base}$

where $cancel_{post}$ $self\ r\ result \equiv self$ .cl-res$->select_{Set}(res|res$ .flight $\doteq r$ .flight@pre)$->isEmpty_{Set}()$

lemma $cancel_{nonblocking} : \exists\ self\ r\ \sigma.\ (\sigma, \sigma') \models (cancel_{pre}$ $self\ r)$
⟨proof⟩

lemma $cancel_{nonblocking\_pre} : \exists\ self\ r\ \sigma.\ \sigma \models_{pre} (cancel_{pre}\ \ self\ r)$
⟨*proof*⟩

lemma $cancel_{implementable}$:
assumes *pre-satisfied*: $\sigma \models_{pre} (cancel_{pre}\ \ self\ r)$
shows $\exists\ \sigma'\ result.\ ((\sigma, \sigma') \models \delta\ self) \longrightarrow$
$((\sigma, \sigma') \models \upsilon\ r) \longrightarrow$
$((\sigma, \sigma') \models (cancel_{post}\ \ self\ r\ result))$
⟨*proof*⟩

> As remark, the pre-condition $\sigma \models_{pre} cancel_{pre}\ self\ r$ has not been used; in the special case of the operation "cancel", the post-condition is satisfiable for *arbitrary* defined and valid input, even input that does not satisfy the pre-condition.

end

8.3 Mega Theorem Proving: Kilo in Practice, Giga in View

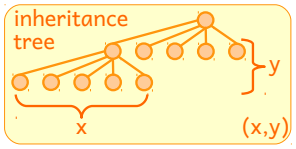
At this point, the reader is perhaps convinced that this is an impressing stunt, but may have remaining doubts about its practical value. While there are other projects supporting our experience that code generation in Isabelle is a maturing technology producing reasonably efficient code for tools (e. g. [ELN+14]), the question remains to be settled if the generated code is sufficiently controllable in an interactive setting ("no blobs making the IDE freeze") and scales well enough to relevant examples.

To this end, we study the following scaling scenario: we implement a new package called "*the Tree Package*", which is internally (lazily) calling the Class Model Package to generate a sample of class models, where each class model is arranged as a tree, and we run an experiment over the key parameters of this sample. In particular in all class models, every class will exactly inherit from one class (using the _ < _ relation), except `OclAny` standing as the only root.

Generated and Proved Theorems

We present Figure 8.2 a table reporting the number of theorems associated to each tested class model. Numbers of generated theorems are indicated by powers of 1000 (so Kilo and Mega). The class models we are measuring can be uniquely identified by pairs (X, Y) where X is the exact number of subclasses of every class having at least one subclass; and Y is the depth of the inheritance tree (without `OclAny`).[3] Only two components are needed for a unique identification, because we are only considering perfect trees, i. e. where all leaf nodes are at the same depth. In particular, for space reasons, the inheritance tree depicted in Figure 8.2 is not a perfect tree but just shows what X and Y are representing. Class-models appear sorted in the table according to the following priority:

[3] The Tree Package comes with the higher-order meta-command Tree, which precisely takes X and Y as arguments.



c	depth 1		depth 2		depth 3		depth 4		depth 5		depth c	
12	(c, 1)	11K	(3, 2)	12K							(1, c)	14K
14	(c, 1)	16K			(2, 3)	17K					(1, c)	20K
20	(c, 1)	39K	(4, 2)	39K							(1, c)	52K
30	(c, 1)	115K	(5, 2)	115K			(2, 4)	121K			(1, c)	155K
39	(c, 1)	240K			(3, 3)	240K					(1, c)	330K
42	(c, 1)	294K	(6, 2)	288K							(1, c)	409K
56	(c, 1)	661K	(7, 2)	649K							(1, c)	964K
62	(c, 1)	882K							(2, 5)	907K	(1, c)	1.3M
72	(c, 1)	1.3M	(8, 2)	1.3M							(1, c)	2M
84	(c, 1)	2.1M			(4, 3)	2.1M					(1, c)	3.3M
90	(c, 1)	2.5M	(9, 2)	2.5M							(1, c)	4.2M

Figure 8.2: Number of generated theorems, measured by a minus of two find_theorems for $c \leq 14$. Otherwise numbers are estimated from the size of associated deep-certificates.

c	depth 1		depth 2		depth 3		depth 4		depth 5		depth c	
12	(c, 1)	3.5M	(3, 2)	3.6M							(1, c)	4.4M
14	(c, 1)	5.1M			(2, 3)	5.4M					(1, c)	6.5M
20	(c, 1)	13M	(4, 2)	13M							(1, c)	17M
30	(c, 1)	38M	(5, 2)	38M			(2, 4)	40M			(1, c)	51M
39	(c, 1)	79M			(3, 3)	79M					(1, c)	109M
42	(c, 1)	97M	(6, 2)	95M							(1, c)	135M
56	(c, 1)	218M	(7, 2)	214M							(1, c)	318M
62	(c, 1)	291M							(2, 5)	299M	(1, c)	432M
72	(c, 1)	448M	(8, 2)	438M							(1, c)	683M
84	(c, 1)	700M			(4, 3)	693M					(1, c)	1.1G
90	(c, 1)	855M	(9, 2)	834M							(1, c)	1.4G

Figure 8.3: Size of generated deep-certificates as stored in the file system, all provided by the operating system (independently of Isabelle)

1. by row using the number of classes in the class model (c = number of classes without `OclAny`); then

2. by column using the depth of the inheritance tree.

For instance, class models in the row $[(30, 1), (5, 2), (2, 4), (1, 30)]$ are sorted in increasing order by depth, all having 31 classes ($c = 30$, `OclAny` counts for 1).

With only `OclAny` as class, we generate 151 definitions and theorems (`thm`'s); by adding another class, it reaches 335 `thm`'s. Since generated theorems may occur in the Isabelle simplifier-set as hints, it becomes desirable to have at the same time more theorems, short and quick proofs whenever applicable. As an extreme example, we chose $c = 90$ where the generation of the deep-certificate consists of nearly 4 million of `thm`'s (loading it in Isabelle to check it, however, is unfeasible at the time of writing and with the computer used before this thesis was released). Note that in these artificial class models we have $2^{(n^2)}$ casts, so there is an inherent combinatorial explosion in the generation process.

As a side remark, the presented table is not trying to reach an arbitrary maximal number of theorems, it would suffice to produce otherwise simpler lines of the form

meta
lemma a_n : "$x_n = True \Longrightarrow x_n$" by *simp*

for several increasing n. Instead, the presented table is mainly reporting the number of UML/OCL generated theorems to not only compare the performance of (X, Y) versus (Y, X), but to also serve as a point of reference for various future improvements. One can indirectly observe for example the number of theorems the own packages of Isabelle are performing, since the generation also relies on underlying Isabelle packages, like datatype. Besides, certain design decisions regarding the semantics of UML/OCL could also be easily monitored: e. g., would the generation be affected if we implement casts in $O(1)$ versus casts in $O(n)$? By comparison, the simple fact of starting Isabelle 2016 already provides 17133 theorems in HOL, whereas for Isabelle 2015 we get 15688 theorems. On the one hand, whenever we are modelling a class model with at least 15 classes (where $c = 14$), one can expect to type-check a theory with a density comparatively similar (or at least similar) to what we obtain by typing all HOL for example. On the other hand, a rich object-logic can be considered as usable, as soon as it can be compiled at least once (or as soon as one has a strong evidence that it can be correctly compiled, by following the principle developed in Chapter 5). So the table is also reporting the minimum value of c where, after this value, one would need to manually disable the generation of too large theorems, depending one's own desired targeted performance. In the table all theorems we have implemented (until now) are set to be fully generated by default, so 15 (or $c = 14$) is a relative value that can be increased as well as decreased: by manually adjusting which theorems actually need to be generated. Finally, this adjustment also depends on the domain-specific problems one is attempting to resolve, and the desired proving policy: e. g., automated theorem proving, interactive theorem proving, etc.

Note that, since all theorem names are also generated, they differ only in the names of classes involved, e. g. "down_cast_kind$_X$_from_Y_to_Z", with X, Y and Z varying over class names. Thus, searching particular patterns with find_theorems resembles to many other Isabelle packages (e. g., like datatype).

Last, Figure 8.3 is similar as Figure 8.2 except that it measures the disk usage of Isabelle deep-certificates[4]: all sizes are not estimated, but really provided by

[4]For equity reason, all names of all classes have been chosen to have the same length. For instance, with 4 bytes and an alphabet of 26 letters, we had enough fresh names for correctly

the operating system. So they are indicated in a power of 1024 bytes (Kilo, Mega, Giga). The question whether it was easy or not to generate all these files is now discussed in the next subsection.

Time and Space to Generate deep-Certificates

Besides the constraint of a high number of theorems, time or space for the generation is also a criteria to consider as enhancement. Below, we present the time used for producing one deep-certificate on a computer with 4 cores[5], e. g. for the pair $(2, 2)$, using all target intermediate languages. We also list the size of the respective source code extracted by each target language (where for OCaml and Isabelle/ML, the extraction of type signatures has been deactivated by hand, with a minor patch in the source code of Isabelle):

Haskell	19 sec	source = 15.6 kLOC	compiled object files = 3.3M
OCaml	16 sec	source = 13.6 kLOC	compiled object files = 8.5M
Scala	131 sec	source = 46.4 kLOC	compilation in RAM
Isabelle/ML	29 sec	source = 14.3 kLOC	compilation in RAM
self	1 sec	already reflected in RAM	already reflected in RAM

Currently the incremental compilation is not yet implemented for Scala, and half implemented for Isabelle/ML, so in the worst case, the complete source of the meta-translation in Isabelle/HOL (counting about 10 kLOC of Isabelle/HOL) must be extracted every-time, i. e., 10 kLOC in front of *each* meta-command. So if a theory contains lots of meta-commands, all of them have to be taken into account before reaching the final generation_syntax deep flush_all. For example, for a theory containing 12 non-lazy meta-commands (and 9 lazy meta-commands which are only ignored by Haskell, OCaml, Isabelle/ML, and self), we obtain as results:

Haskell	1 min	51 sec
OCaml	1 min	39 sec
Scala	45 min	11 sec
Isabelle/ML	9 min	34 sec
self	0 min	2 sec

By comparison, the pair $(2, 2)$ does not have lazy meta-commands in its associated deep-certificate.

Finally the resources needed to generate the deep-certificate of the pair $(1, 56)$ are 9G of RAM memory and 1 min; for $c = 90$ we used 28G and 7 min. However these benchmarks were performed in 2014 without using the self-mode. In 2016, we obtain almost similar performances for time, for instance $(1, 30)$ costs 4 sec. For space, performances have been improved, e. g., with the self-mode, the generation of $(1, 56)$ only consumes less than 1G of RAM and 1 min, for $c = 90$ also 1G of RAM.

---

conducting our benchmark, and compare a hundred of classes.

[5]We assume that the meta-tool has already been reflected before measuring each listed time. As remark, one can observe that most functions of the meta-translation of Figure D.3 are unrelated, so they can be split into several files in parallel, e. g., in the picture simultaneous processing can normally treat up to 10 files at the same time.

All results in this subsection only concern generation not typing, resources needed for loading and typing these lemmas in Isabelle/jEdit will be detailed in the next subsection.

Typing a deep-Certificate versus Typing in shallow-Mode

This subsection stands as a pre-requisite for the subsection detailing the numbers of generated theorems in Figure 8.2, because an Isabelle file becomes recognized as a set of theorems, only after being type-checked by the system. We confront the two strategies of Figure 6.1, namely the resources needed for the type-checker of Isabelle to reach

- the end of a deep-certificate,
- versus the end of the associated file in shallow-mode.

Semantically speaking, these two typing require the core library of Featherweight OCL [BTW14] yet allocating 1394M.

The typing of the deep-certificate for the pair $(3, 2)$ (where $c = 12$) runs in 3 min 44 sec and 436M memory. Its pre-processing in Isabelle/jEdit takes about 18% of that time, and the remaining 82% represents to complete proofs checking.

Independently, the loading of the same pair $(3, 2)$ in shallow-mode runs in 3 min 04 sec and 485M memory, takes less than 1% of pre-processing time. Contrasting with the deep-certificate, the shallow-mode depends on the entire meta-tool project (of size 1066M), which is moreover reflected with a certain cost from Isabelle/HOL to Isabelle/ML: 40s and an increase of 414M.

We notice here that the code reflection of Isabelle seems to make only use of at most one single core. However the meta-tool needs to be reflected only once, so this can be an advantage with a lot of class models in parallel in the same editor when experimenting in shallow. On the contrary in deep, each certificate consumes generally a high pre-processing time no matter files in parallel. The pre-processing is fast in shallow because of generally few meta-commands, whereas for the deep the cost comes from the high number of Isar_HOL commands already generated. We still believe it feasible to separate these Isar_HOL commands into separate Isabelle theory files, instead of one single theory file, because certain theorems are actually not related together. Then we could count on the native parallelism support of Isabelle to improve the overall performance.

CHAPTER 9

Conclusion

A Summary on Related Work

On the one hand, HOL-OCL 2.0 presented in this thesis shares similarities with its predecessor HOL-OCL [BW09, BW08a, BW08b]. The latter is also based on a shallow embedding of UML class models and OCL into Isabelle/HOL. However, HOL-OCL is based on a "hand-coded" series of packages (instead of a generated, reflection-based approach) implemented for an older version of Isabelle/HOL and uses the old Proof General user interface that limits a UML/OCL specific user experience. Moreover, HOL-OCL 2.0 complies to the latest OCL standard which, in particular, supports a four valued logic instead of a three valued logic used in older versions.

On the other hand, while presented ideas have similarities with the way one can apply Isabelle to build a family of formal method tools [WW07], there had been dramatic improvements in the last eight years of the Isabelle platform that encouraged us to a re-implementation emphasising recent technologies. These improvements consist most notably in:

- pervasive parallelism in the prover kernel, which enables us to profit from the computer power of recent multi-core hardware,
- dramatic improvements on the code generation, paving the way to develop tactic code for logical components (or "packages") in the full Isabelle framework, with unlimited switches between HOL and the ML layer in Isabelle/jEdit, and
- new front-end technologies like Isabelle's Prover IDE which allow for new paradigms in user interaction and theory exploration.

The idea to use ML for supporting datatype theories is in itself very old and deeply linked from the very beginning with theorem proving environments such as Edinburgh LCF, HOL4, HOL Light, Isabelle and Coq.

In relation with Coq, some similarities might exist between certain parts of the SimSoC-Cert project[1] [SMTB11, Shi13] and certain parts of the present work. In SimSoC-Cert, there is a particular tool taking as input the reference

[1] `https://gforge.inria.fr/projects/simsoc-cert/`

manuals of several vendor's processors, for example the SH4 manual [Ren06], and generating in output either a Coq certificate that is intended to be readably inspected, or a C file that can be further compiled for an efficient execution (compared to a native execution in Coq). Since our approach in this thesis intends to be generic, e. g., can serve to support decision procedures, or the construction of arbitrary packages (as soon as one can write a datatype representing some domain-specific language, and a constructive embedding function from this datatype to Isar_HOL), we believe the methodology of SimSoC-Cert to generate some certificates for the certification of processors' simulators can be as well transposed here. However, although SimSoC-Cert took advantage of the Coq type-system (in particular dependent types) to ease the pretty-printing process of a Coq certificate, there are in SimSoC-Cert no common platform combining both the deep-mode and the shallow-mode at the same time, also no meta-model of Coq in Coq that has been used as target's certificate (e. g., CoqInCoq [BW97, Bar10]). Instead, the CompCert C meta-model in Coq has been employed to provide many other advantages [Ler09], like among other a convenient compiling infrastructure targeting assembly code, associated with a large library of verified code. Besides, all the code in the SH4 manual are not shown in textual OCL but with a C-like syntax, that can be made quickly acceptable for CompCert's input. As another difference with the present work, when SimSoC-Cert was implemented, there was to our knowledge no easy facilities to mix OCaml code with Coq code inside a same editing environment without leaving the editing session, or to modify the source code of Coq in the (highest) IDE session at run-time (i. e., something similar as the Isabelle command ML, and the implicit code reflection mechanism integrated in the system of ML's antiquotations to refer to Isar_HOL values [WC07]).

The application to *object-oriented* datatype theories is also not new—earlier works in this line can be cited for example [Wen97]. In contrast to HOL-OCL [BW08c], we applied these techniques to UML under *closed world assumption* for a standard-conform 4-valued logics for OCL, which is seen as the semantic framework for DSL's. This is particularly important and challenging since heterogeneous system specifications need to be combined in a seamless way, and since semantically correct tools have to be developed for these language combinations.

As a summary of Section 4.7, we would like to emphasise the following points:

- There are several compilers attempting a standard-conform semantics for UML/OCL, but few verification tools addressing the problems arising from a four-valued logics with two exceptional elements in all types with different strictness behaviour;

- The closest related work in this category are HOL-OCL [BW08b] (interactive proof) and OCL2FOL$^+$ [DC13, ADEM14] (automated proof); our work uses either a different semantic model reflecting the recent standard or goes for a less axiomatic approach;

- While object-oriented specification languages supporting `null` are quite common [BCF$^+$13, Mey97, LPC$^+$13, BLS05], none of them provides a strict exception element for modelling exceptions as first-class citizen. The implicit handling of strict and non-strict exceptional elements in OCL al-

lows for a particular concise specification style avoiding explicit tests for memory;

- Notably, both JML and Spec# limit null elements to class types and provide a type system supporting non-null types. In the case of JML, the non-null types are even chosen as the default types [CR05]. While non-null types can partly be simulated by non-null cardinalities, full support of non-null types clearly simplify specifications drastically, as many cases resulting in potential invalid states (e. g., de-referencing a null) are already ruled out by the type system.

Conclusion and Future Work

We presented HOL-OCL 2.0, based on a core library Featherweight OCL, a formal, machine checked semantics for UML/OCL in Isabelle/HOL. HOL-OCL 2.0 comprises a meta-tool to construct semantic based tools for textual domain specific languages. The meta-tool fundamentally relies on the code generator of Isabelle, and Isabelle theories, to define a model-transformation in Isabelle/Isar_HOL from a UML meta-model (class-models, plus OCL invariants and contracts) to an Isar_HOL meta-model. Compared to conventional implementations of *code-generators* for OCL, the resulting meta-tool is clearly not competitive in terms of compilation size of models, on the other hand, we argue that this comparison is unfair since these tools do not bother to construct the underlying *semantic theory* of UML and OCL in HOL in order to allow formal proofs over it. Our tool is unique that it actually provides two ways to load the number of theorems resulting from class-models: natively at run-time, with a straight interaction with the kernel of Isabelle (in shallow-mode); or as an Isabelle certificate to be loaded afterwards like an object-logic (in deep-mode).

Based on a library with operations for basic and collection types that contain the exception elements `invalid` and `null`, HOL-OCL 2.0 allows for the specifications of programs based on object-oriented data structures. Our work makes this notion precise and allows for a comparison to other object-oriented specification languages such as Eiffel, Spec# or JML. A particular feature of our approach is that the datatype theories are constructed from axiomatic definitions over a constructed typed object universe, which allows for the automatic *derivation* of the entire set of rules guaranteeing logical consistency.[2] Since the HOL-OCL 2.0 environment dynamically instantiates and discharges such rules during the object-oriented modelling activity (for instance typically those presented in Chapter 7), our approach is, as we believe, relevant for other object-oriented verification methods which axiomatize their underlying theory and therefore raise the question of trust in their foundations.

Due to parallelization techniques inherited from Isabelle, HOL-OCL 2.0—for which we still see a large potential for optimisations— remains fairly usable in an interactive setting for medium-sized class-models. Automatic generation with proofs of the datatype theory is, as our implementation shows, still feasible in an interactive use: for the running Flight example 2301 definitions and lemmas are

[2] Our two examples Appendix B and Appendix C sketch how this construction can be captured by an automated process.

generated in 1 second in deep-mode, while their proofs asynchronously terminate in shallow-mode 2 minutes later (in a background thread). Still, unrelated lemmas can be selectively activated or deactivated: by default all are proved.

It is our ultimate goal to complement HOL-OCL 2.0 by the most common behavioural model types of the UML, namely textually presented state machines and sequence diagrams. The resulting environment could serve as a demonstrator for formal techniques for UML and a bridge to industrial partners active in the embedded systems domain.

Our work on HOL-OCL 2.0 stands in the context of a standardisation initiative using formal methods for UML/OCL. In particular, a formal semantics for a core-language based on denotational semantic definitions has been developed in this thesis. The body of rules for interactive and automated proof techniques has been derived by means of interactive theorem proving, pushing at the same time the frontiers of meta theorem proving and mega theorem proving. Since the approach can guarantee logical consistency, not only for thousands of generated rules, but precisely the foundational core-library of Featherweight OCL in itself, our experience can be re-used for other standardisation efforts of "real" programming languages, or at least provide further evidence that this kind of work is nowadays absolutely feasible and worth the effort. A large number of "issues" have been detected, both inconsistencies or formal gaps, and our proposals to resolve them consistently finally found their way in the standardisation process. Ultimately, we aim at providing a machine-checked formal semantics that can be included in the OCL standard, i. e., replacing the current Annex A. This effort may stimulate tool-development, as a clarified semantics helps to develop, for example, optimised schemes of compilation of four-valued OCL logics to recent SMT solvers.

The Flight Model (Modelled by Hand)

This chapter is exactly similar as Section 8.2, except that proofs are displayed.

theory
Flight-Model
imports
../src/UML-OCL

begin

Class Model

This part corresponds to the writing in Isabelle of the code shown in Figure 3.2.

Class *Flight*
Attributes
seats : *Integer*
from : *String*
to : *String*
End

lemma *id* = (λ*x*. *x*)
by (*rule id-def*)

As remark, we are checking for example that the constant *id* already exists, and that one can also use this name in the following attribute: no conflict will happen.

Class *Reservation*
Attributes
id : *Integer*
date : *Week*
End

Class *Person*
Attributes
name : *String*
End

Class *Client* < *Person*
Attributes
address : *String*
End

Class *Staff* < *Person*

```
End

Association passengers
  Between Person      [*]
          Role passengers
        Flight        [*]
          Role flights
End

Aggregation flights
  Between Flight      [1]
          Role flight
        Reservation   [*]
          Role fl-res Sequence-
End

Association reservations
  Between Client      [1]
          Role client
        Reservation   [*]
          Role cl-res
End

Association connection
  Between Reservation [0 . . 1]
          Role next
        Reservation   [0 . . 1]
          Role prev
End
```

In complement to Figure 3.2, we define an enumeration type.

```
Enum Week
  [ Mon, Tue, Wed, Thu, Fri, Sat, Sun ]
End!
```

Two State Instances of the Class Model

The creation of (typed) object instances is performed in HOL-OCL 2.0 with the command Instance:

```
Instance S1  :: Staff  = [ name = Merlin , flights = F1 ]
    and C1   :: Client = [ name = Bertha , address = Miami , flights = F1 , cl-res = R11 ]
    and C2   :: Client = [ name = Arthur , address = Valencia , flights = F1 , cl-res = R21 ]
    and R11  :: Reservation = [ id = 12345 , flight = F1 , date = Mon ]
    and R21  :: Reservation = [ id = 98765 , flight = F1 ]
    and F1   :: Flight = [ seats = 120 , from = Valencia , to = Miami ]
    and F2   :: Flight = [ seats = 370 , from = Miami , to = Ottawa ]
```

The notion of object instances comes before that of states. Currently, we have only created the object instances *S1*, *C1*, *C2*, *R11*, *R21*, *F1* and *F2*. They will need to be "registered" in a state later. Instance verifies that all objects being created are respecting the multiplicities declared above in classes (in the bidirectional sense). For example, after the type-checking stage, we have correctly that $R21\ .client \cong Set\{C2\}$, since *R21* appears as one reservation of *C2*, and where "$X \cong Y$" stands as a synonym for $\forall \tau.\ \tau \models \delta\ X \longrightarrow \tau \models \delta\ Y \longrightarrow \tau \models X \triangleq Y$.[1]As remark, the order of attributes and objects declarations is not important: mutually recursive constructions become de-facto supported. As illustration, we can include here the text displayed in the output window after evaluating the above Instance (we have manually pasted the text from the output window in Isabelle/jEdit):

$S1\ .flights \cong Set\{\ F1\ \}$
$C1\ .flights \cong Set\{\ F1\ \}$
$C1\ .cl\text{-}res \cong Set\{\ R11\ \}$

C2 .flights ≅ *Set{ F1 }*
C2 .cl-res ≅ *Set{ R21 }*
R11 .flight ≅ *Set{ F1 }*
R11 .client ≅ *Set{ C1 }*
R11 .prev ≅ *Set{}*
R11 .next ≅ *Set{}*
R21 .flight ≅ *Set{ F1 }*
R21 .client ≅ *Set{ C2 }*
R21 .prev ≅ *Set{}*
R21 .next ≅ *Set{}*
F1 .passengers ≅ *Set{ S1 , C1 , C2 }*
F1 .fl-res ≅ *Set{ R11 , R21 }*
F2 .passengers ≅ *Set{}*
F2 .fl-res ≅ *Set{}*

We can check that *S1* indeed exists and has the expected OCL type.

term *S1* ::· *Staff*

Once objects are constructed with Instance, it becomes possible to regroup them together into a state. This is what the next command State is doing by creating a state named σ_1, corresponding to the pre-state of Figure 8.1.

State σ_1 = [*S1*, *C1*, *C2*, *R11*, *R21*, *F1*, *F2*]

This generates a number of theorems from it, e. g.:

$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Staff .allInstances@pre*() ≜ *Set{S1}*
$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Client .allInstances@pre*() ≜ *Set{C1,C2}*
$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Reservation .allInstances@pre*() ≜ *Set{R11,R12}*
$\bigwedge\sigma.\ (\sigma_1, \sigma) \models$ *Flight .allInstances@pre*() ≜ *Set{F1,F2}*

At this point, it is not yet sure that σ_1 will be used in the pre-state or post-state. In any case, the above command also generates the following symmetric lemmas:

$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Staff .allInstances*() ≜ *Set{S1}*
$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Client .allInstances*() ≜ *Set{C1,C2}*
$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Reservation .allInstances*() ≜ *Set{R11,R12}*
$\bigwedge\sigma.\ (\sigma, \sigma_1) \models$ *Flight .allInstances*() ≜ *Set{F1,F2}*

Because all these lemmas are stated under the precondition that all object instances are defined entities, lemmas generated by State are actually proved in a particular locale [Bal14, Bal16] *state-*σ_1. Thus the header of *state-*σ_1 regroups these (mandatory) definedness assumptions, that have to be all satisfied before being able to use the rules defined in its body.

The next statement illustrates Chapter 6. It shows for instance that object instances can also be generated by State on the fly. Fresh variables are created meanwhile if needed, like σ_2*-object1*.

State σ_2 =
[*S1*
, ([*C1* with-only *name* = *Bertha*, *address* = *Saint−Malo* , *flights* = *F1* , *cl-res* = *R11*] :: *Client*)
, ([*C2* with-only *name* = *Arthur*,*address* = *Valencia*,*flights*=[*F1*,*F2*],*cl-res*=[self *4*,self *7*]]::*Client*)
, *R11*
, ([*R21* with-only *id* = *98765* , *flight* = *F1* , *next* = self *7*] :: *Reservation*)
, *F1*
, *F2*

[1] Although such rule schemata may be explicitly generated by Instance (for most OCL expressions), they can also not be: at the time of writing, the complete type-checking process is at least fully executed from an extracted HOL function (as one consequence, the type-checking process terminates). This is feasible because for the moment, Instance only accepts "grounds objects" as arguments (the reader is referred to its syntax diagram detailed in Appendix I).

, ([*id* = *19283* , *flight* = *F2*] :: *Reservation*)]

Similarly as with Instance, we can paste in the following what is currently being displayed in the output window (where "/∗8∗/" means the object having an oid equal to 8).[2]

σ_2-*object1* .*flights* ≅ *Set*{ /∗8∗/ }
σ_2-*object1* .*cl-res* ≅ *Set*{ /∗6∗/ }
σ_2-*object2* .*flights* ≅ *Set*{ /∗8∗/ , /∗9∗/ }
σ_2-*object2* .*cl-res* ≅ *Set*{ σ_2-*object4* , σ_2-*object7* }
σ_2-*object4* .*flight* ≅ *Set*{ /∗8∗/ }
σ_2-*object4* .*client* ≅ *Set*{ σ_2-*object2* }
σ_2-*object4* .*prev* ≅ *Set*{}
σ_2-*object4* .*next* ≅ *Set*{ σ_2-*object7* }
σ_2-*object7* .*flight* ≅ *Set*{ /∗9∗/ }
σ_2-*object7* .*client* ≅ *Set*{ σ_2-*object2* }
σ_2-*object7* .*prev* ≅ *Set*{ σ_2-*object4* }
σ_2-*object7* .*next* ≅ *Set*{}

Note that there is a mechanism to reference objects via the (invented) keyword self (it has no particular relation with the one used in Chapter 6), which takes a number designating the index of a particular object instance occurring in the list of declarations (the index starts with 0 as first position).

Similarly as for *state*-σ_1, we obtain another locale called *state*-σ_2, representing the post-state of Figure 8.1.

The Transition command relates the two states together.

Transition σ_1 σ_2

The first state is intended to be understood as the pre-state, and the second state as the post-state. In particular, we do not obtain similar proved theorems if we write Transition σ_1 σ_2 or Transition σ_2 σ_1 (assuming σ_1 and σ_2 are different). Generally, Transition establishes for a pair of a pre- and a post state (i.e. a state transition) that a number of crucial properties are satisfied. For instance, the well-formedness of the two given states is proven: *WFF*(σ_1, σ_2).

Furthermore, for each object X additional lemmas are generated to situate X as an object existing in σ_1, σ_2, both, or in any permutations of σ_1 and σ_2. Such lemmas typically resemble as:

- (σ_1, σ_2) ⊨ X .*oclIsNew*(), or
- (σ_1, σ_2) ⊨ X .*oclIsDeleted*(), or
- (σ_1, σ_2) ⊨ X .*oclIsAbsent*(), or
- (σ_1, σ_2) ⊨ X .*oclIsMaintained*()

where the latter only means that the oid of X exists both in σ_1 and σ_2, in particular the values of the attribute fields of X have also not changed.

As completeness property, we can state the following lemma covering all disjunction case (for any X and τ) [BTW14]: $\tau \models \delta\ X \Longrightarrow \tau \models X.oclIsNew() \vee \tau \models X.oclIsDeleted() \vee \tau \models X.oclIsMaintained() \vee \tau \models X.oclIsAbsent()$

Finally Transition proceeds as State: it builds a new locale, called *transition*-σ_1-σ_2, by particularly instantiating the two locales *state*-σ_1 and *state*-σ_2.

The following lemma establishes that the generated object presentations (like *S1* = (λ-. $\lfloor\lfloor S1_{Staff}\rfloor\rfloor$), *C1* = (λ-. $\lfloor\lfloor C1_{Client}\rfloor\rfloor$), etc.) satisfy the requirements of the locale *state*-σ_1. In particular, it has to be shown that the chosen object representations are defined and have distinct oids. Proving this lemma gives access to the already defined properties in this locale.

lemma σ_1: *state-interpretation*-σ_1 τ
by(*simp add*: *state-interpretation*-σ_1-*def*,
default, *simp add*: *pp-oid*-σ_1-σ_2,

[2] As future work, it is plan for Instance to support the writing of arbitrary OCL expressions, including the assignment of potentially infinite collection types (for example "a set of sequence of bag of objects"). In particular, besides the cardinality of the manipulated collection types, the sole information required for checking multiplicities appears to be the oid of objects.

(*simp add*: *pp-object-*σ_1*-*σ_2)+)

This instance proof goes analogously.

lemma σ_2: *state-interpretation-*σ_2 τ
by(*simp add*: *state-interpretation-*σ_2*-def*,
default, *simp add*: *pp-oid-*σ_1*-*σ_2,
(*simp add*: *pp-object-*σ_1*-*σ_2)+)

The latter proof gives access to the locale *transition-*σ_1*-*σ_2.

lemma σ_1*-*σ_2: *pp-*σ_1*-*σ_2 τ
by(*simp add*: *pp-*σ_1*-*σ_2*-def*,
default, *simp add*: *pp-oid-*σ_1*-*σ_2,
(*simp add*: *pp-object-*σ_1*-*σ_2)+,
(*simp add*: *pp-oid-*σ_1*-*σ_2)+)

For convenience, one can introduce the empty state here

definition σ_0 :: $\mathfrak{A}$ *state* where σ_0 = *state.make Map.empty Map.empty*

so that the following abbreviations can be written

definition σ_{t1} = *transition-*σ_1*-*σ_2*.*σ_1 *oid3 oid4 oid5 oid6 oid7 oid8 oid9*
⌈⌈*S1* (σ_0, σ_0)⌉⌉ ⌈⌈*C1* (σ_0, σ_0)⌉⌉ ⌈⌈*C2* (σ_0, σ_0)⌉⌉ ⌈⌈*R11* (σ_0, σ_0)⌉⌉
⌈⌈*R21* (σ_0, σ_0)⌉⌉ ⌈⌈*F1* (σ_0, σ_0)⌉⌉ ⌈⌈*F2* (σ_0, σ_0)⌉⌉

definition σ_{t2} = *transition-*σ_1*-*σ_2*.*σ_2 *oid3 oid4 oid5 oid6 oid7 oid8 oid9 oid10*
⌈⌈*S1* (σ_0, σ_0)⌉⌉ ⌈⌈σ_2*-object1* (σ_0, σ_0)⌉⌉ ⌈⌈σ_2*-object2* (σ_0, σ_0)⌉⌉ ⌈⌈*R11* (σ_0, σ_0)⌉⌉
⌈⌈σ_2*-object4* (σ_0, σ_0)⌉⌉ ⌈⌈*F1* (σ_0, σ_0)⌉⌉ ⌈⌈*F2* (σ_0, σ_0)⌉⌉
⌈⌈σ_2*-object7* (σ_0, σ_0)⌉⌉

definition σ_{s1} = *state-*σ_1*.*σ_1 *oid3 oid4 oid5 oid6 oid7 oid8 oid9*
⌈⌈*S1* (σ_0, σ_0)⌉⌉ ⌈⌈*C1* (σ_0, σ_0)⌉⌉ ⌈⌈*C2* (σ_0, σ_0)⌉⌉ ⌈⌈*R11* (σ_0, σ_0)⌉⌉
⌈⌈*R21* (σ_0, σ_0)⌉⌉ ⌈⌈*F1* (σ_0, σ_0)⌉⌉ ⌈⌈*F2* (σ_0, σ_0)⌉⌉

definition σ_{s2} = *state-*σ_2*.*σ_2 *oid3 oid4 oid5 oid6 oid7 oid8 oid9 oid10*
⌈⌈*S1* (σ_0, σ_0)⌉⌉ ⌈⌈σ_2*-object1* (σ_0, σ_0)⌉⌉ ⌈⌈σ_2*-object2* (σ_0, σ_0)⌉⌉ ⌈⌈*R11* (σ_0, σ_0)⌉⌉
⌈⌈σ_2*-object4* (σ_0, σ_0)⌉⌉ ⌈⌈*F1* (σ_0, σ_0)⌉⌉ ⌈⌈*F2* (σ_0, σ_0)⌉⌉
⌈⌈σ_2*-object7* (σ_0, σ_0)⌉⌉

Both formats are, fortunately, equivalent; this means that for these states, we can access properties from both state and transition locales, in which the object representations are "wired" in the same way.

lemma σ_{t1}*-*σ_{s1}: σ_{t1} = σ_{s1}
unfolding σ_{t1}*-def* σ_{s1}*-def*
apply(*subst transition-*σ_1*-*σ_2*.*σ_1*-def*)
by(*rule* σ_1*-*σ_2[*simplified pp-*σ_1*-*σ_2*-def*], *simp*)

lemma σ_{t2}*-*σ_{s2}: σ_{t2} = σ_{s2}
unfolding σ_{t2}*-def* σ_{s2}*-def*
apply(*subst transition-*σ_1*-*σ_2*.*σ_2*-def*)
by(*rule* σ_1*-*σ_2[*simplified pp-*σ_1*-*σ_2*-def*], *simp*)

The next lemma becomes a shortcut of the one generated by Transition, but explicitly instantiated.

lemma *WFF* (σ_{t1}, σ_{t2})
unfolding σ_{t1}*-*σ_{s1} σ_{t2}*-*σ_{s2} σ_{s1}*-def* σ_{s2}*-def*
apply(*rule transition-*σ_1*-*σ_2*.basic-*σ_1*-*σ_2*-wff*)
apply(*rule* σ_1*-*σ_2[*simplified pp-*σ_1*-*σ_2*-def*])
by(*simp-all add*: *pp-oid-*σ_1*-*σ_2 *pp-object-*σ_1*-*σ_2

oid-of-$\mathfrak{A}$*-def oid-of-ty*$_{Staff}$*-def oid-of-ty*$_{Client}$*-def oid-of-ty*$_{Reservation}$*-def oid-of-ty*$_{Flight}$*-def*

S1 $_{Staff}$-def C1 $_{Client}$-def C2 $_{Client}$-def R11 $_{Reservation}$-def R21 $_{Reservation}$-def F1 $_{Flight}$-def F2 $_{Flight}$-def σ$_2$-object1 $_{Client}$-def σ$_2$-object2 $_{Client}$-def σ$_2$-object4 $_{Reservation}$-def σ$_2$-object7 $_{Reservation}$-def)

```
lemma F1-val-seatsATpre: (σs1, σ) ⊨ F1 .seats@pre ≜ «120»
    proof(simp add: UML-Logic.foundation22 k-def )
      show F1 .seats@pre (σs1, σ) = ⌊⌊120⌋⌋
        proof − note S1 = σ1[simplified state-interpretation-σ1-def, of (σ0, σ0)]
        show ?thesis
          apply(simp add: dotFlight--seatsat-pre F1-def deref-oidFlight-def in-pre-state-def
                          F1 Flight-def oid-of-tyFlight-def oid8-def)
          apply(subst (8) σs1-def, simp add: state-σ1.σ1-def[OF S1], simp add: pp-oid-σ1-σ2)
          apply(simp add: selectFlight--seats-def F1-def F1 Flight-def)
          by(simp add: reconst-basetype-def)
        qed
     qed

lemma F1-val-seatsATpre′: σs1 ⊨pre F1 .seats@pre ≜ «120»
by(simp add: OclValid-at-pre-def F1-val-seatsATpre)

lemma F2-val-seatsATpre: (σs1, σ) ⊨ F2 .seats@pre ≜ «370»
    proof(simp add: UML-Logic.foundation22 k-def )
      show F2 .seats@pre (σs1, σ) = ⌊⌊370⌋⌋
        proof − note S1 = σ1[simplified state-interpretation-σ1-def, of (σ0, σ0)]
        show ?thesis
          apply(simp add: dotFlight--seatsat-pre F2-def deref-oidFlight-def in-pre-state-def
                          F2 Flight-def oid-of-tyFlight-def oid9-def)
          apply(subst (8) σs1-def, simp add: state-σ1.σ1-def[OF S1], simp add: pp-oid-σ1-σ2)
          apply(simp add: selectFlight--seats-def F2-def F2 Flight-def)
          by(simp add: reconst-basetype-def)
        qed
     qed

lemma F2-val-seatsATpre′: σs1 ⊨pre F2 .seats@pre ≜ «370»
by(simp add: OclValid-at-pre-def F2-val-seatsATpre)

lemma F1-val-seats: (σ, σs2) ⊨ F1 .seats ≜ «120»
proof(simp add: UML-Logic.foundation22 k-def )
  show F1 .seats (σ, σs2) = ⌊⌊120⌋⌋
      proof − note S2 = σ2[simplified state-interpretation-σ2-def, of (σ0, σ0)]
           show ?thesis
           apply(simp add: dotFlight--seats F1-def deref-oidFlight-def in-post-state-def F1 Flight-def
                           oid-of-tyFlight-def oid8-def)
           apply(subst (8) σs2-def, simp add: state-σ2.σ2-def[OF S2], simp add: pp-oid-σ1-σ2)
           apply(simp add: selectFlight--seats-def F1-def F1 Flight-def)
           by(simp add: reconst-basetype-def)
      qed
qed

lemma F1-val-seats′: σs2 ⊨post F1 .seats ≜ «120»
by(simp add: OclValid-at-post-def F1-val-seats)

lemma F2-val-seats: (σ, σs2) ⊨ F2 .seats ≜ «370»
proof(simp add: UML-Logic.foundation22 k-def )
  show  F2 .seats (σ, σs2) = ⌊⌊370⌋⌋
```

```
      proof − note  S2 = σ2[simplified state-interpretation-σ2-def, of (σ0, σ0)]
          show ?thesis
          apply(simp add: dotFlight--seats F2-def deref-oidFlight-def in-post-state-def F2Flight-def
                          oid-of-tyFlight-def oid9-def)
          apply(subst (8) σs2-def, simp add: state-σ2.σ2-def[OF S2], simp add: pp-oid-σ1-σ2)
          apply(simp add: selectFlight--seats-def F2-def F2Flight-def)
          by(simp add: reconst-basetype-def)
      qed
qed

lemma F2-val-seats': σs2 ⊨post F2 .seats ≜ «370»
by(simp add: OclValid-at-post-def F2-val-seats)

lemma C1-valid: (σs1, σ') ⊨ (υ C1)
by(simp add: OclValid-def C1-def)

lemma R11-val-clientATpre: (σs1, σ') ⊨ R11 .client@pre ≜ C1
  proof(simp add: foundation22)

  have C1-deref-val: (σs1, σ') ⊨ deref-oidClient fst reconst-basetype 4 ≜ C1
    proof(simp add: foundation22)
    show deref-oidClient fst reconst-basetype 4 (σs1, σ') = C1 (σs1, σ')
      proof − note S1 = σ1[simplified state-interpretation-σ1-def, of (σ0, σ0)]
      show ?thesis
          apply(simp add: deref-oidClient-def)
          apply(subst (8) σs1-def, simp add: state-σ1.σ1-def[OF S1], simp add: pp-oid-σ1-σ2)
          by(simp add: reconst-basetype-def C1-def)
      qed
    qed

  show R11 .client@pre (σs1, σ') = C1 (σs1, σ')
  proof − note S1 = σ1[simplified state-interpretation-σ1-def, of (σ0, σ0)]
  show ?thesis
    apply(simp add: dotReservation-1---clientat-pre R11-def deref-oidReservation-def in-pre-state-def
                    R11Reservation-def oid-of-tyReservation-def oid6-def)
    apply(subst (8) σs1-def, simp add: state-σ1.σ1-def[OF S1], simp add: pp-oid-σ1-σ2)
    apply(simp add: deref-assocsReservation-1---client-def deref-assocs-def oidReservation-1---client-def)
    apply(subst (3) σs1-def, simp add: state-σ1.σ1-def[OF S1] map-of-list-def
          oidClient-0---flights-def oidStaff-0---flights-def oidClient-0---cl-res-def)
    apply(simp add: switch2-01-def switch2-10-def choose-0-def choose-1-def deref-assocs-list-def
                    pp-oid-σ1-σ2 R11-def R11Reservation-def oid-of-tyReservation-def List.member-def)
    apply(simp add: selectReservation--client-def select-object-anySet-def select-objectSet-def)
    apply(subgoal-tac (let s = Set{deref-oidClient fst reconst-basetype 4} in
                       if s->sizeSet() ≜ 1 then s->anySet() else ⊥ endif) (σs1, σ') = C1 (σs1, σ'))
     apply(subgoal-tac Set{deref-oidClient fst reconst-basetype 4} =
           select-object Set{} UML-Set.OclIncluding id (deref-oidClient fst reconst-basetype) [4])
      apply(simp only: Let-def)
     apply(simp add: select-object-def)
    apply(simp only: Let-def)
    apply(subst cp-OclIf, subst OclSize-singleton[simplified OclValid-def])
     apply(subst cp-valid)
     using C1-deref-val[simplified OclValid-def StrongEq-def true-def]
     apply(simp, subst cp-valid[symmetric], simp add: C1-valid[simplified OclValid-def])
    using C1-deref-val[simplified OclValid-def StrongEq-def true-def]
    by(subst cp-OclIf[symmetric], simp)
  qed
qed
```

Annotations of the Class Model in OCL

Subsequently, we state a desired class invariant for `Flight`'s in the usual OCL syntax:

Context f: *Flight*
  Inv A : $\mathbf{0} <_{int} (f\ .seats)$
  Inv B : $f\ .fl\text{-}res\ ->size_{Seq}() \leq_{int} (f\ .seats)$
  Inv C : $f\ .passengers\ ->select_{Set}(p \mid p\ .oclIsTypeOf(Client))$
  $\doteq ((f\ .fl\text{-}res)->collect_{Seq}(c \mid c\ .client\ .oclAsType(Person))->asSet_{Seq}())$

Model Analysis: A satisfiability proof of the invariants

We wish to analyse our class model and show that the entire set of invariants can be satisfied, i. e. there exist legal states that satisfy all constraints imposed by the class invariants.

lemma *Flight-consistent*: $\exists\ \tau.$ *Flight-Aat-pre* $\tau \wedge$ *Flight-A* τ
proof (*rule-tac* x=$(\sigma_{t1}, \sigma_{t2})$ in *exI*, *rule conjI*)

The following auxiliary fact establishes that $\tau \models \delta\ S \Longrightarrow \tau \models S->forAll_{Set}(X|P) \triangleq (S \triangleq Set\{\}\ or\ P)$ from the library is applicable since $OclAsType_{Flight}$-$\mathfrak{A}$ *.allInstances@pre*() is indeed defined.

have *forall-trivial*: $\bigwedge\tau\ P.$ *let* $S = OclAsType_{Flight}$-$\mathfrak{A}$ *.allInstances@pre*() *in*
  $(\tau \models (S->forAll_{Set}(X|P) \triangleq (S \triangleq Set\{\}\ or\ P)))$
  unfolding *Let-def* by(*rule OclForall-body-trivial*, *rule OclAllInstances-at-pre-defined*)
show *Flight-Aat-pre* $(\sigma_{t1}, \sigma_{t2})$
proof −
  have ∗: $(\sigma_{t1}, \sigma_{t2}) \models (\mathbf{0} <_{int} (F1\ .seats@pre))$
    apply(*subst UML-Logic.StrongEq-L-subst3-rev*[*OF F1-val-seatsATpre*,
      simplified σ_{t1}-σ_{s1}[*symmetric*]],*simp*)
    by(*simp add*: *OclInt0′*)
  have ∗∗: $(\sigma_{t1}, \sigma_{t2}) \models (\mathbf{0} <_{int} (F2\ .seats@pre))$
    apply(*subst UML-Logic.StrongEq-L-subst3-rev*[*OF F2-val-seatsATpre*,
      simplified σ_{t1}-σ_{s1}[*symmetric*]],*simp*)
    by(*simp add*: *OclInt0′*)

Now we calculate:

have $((\sigma_{t1}, \sigma_{t2}) \models Flight\ .allInstances@pre()->forAll_{Set}(self|$
  $Flight\ .allInstances@pre()->forAll_{Set}(f|\mathbf{0} <_{int} f\ .seats@pre))) =$
  $((\sigma_{t1}, \sigma_{t2}) \models Flight\ .allInstances@pre() \triangleq Set\{\}\ or$
  $Flight\ .allInstances@pre()->forAll_{Set}(f|\ \mathbf{0} <_{int} f\ .seats@pre))$
  by(*simp add*: *StrongEq-L-subst3*[*OF - forall-trivial*[*simplified Let-def*],
    where $P = \lambda x.\ x$])
also
have $\ldots = ((\sigma_{t1}, \sigma_{t2}) \models ((Set\{F1, F2\} \triangleq Set\{\})\ or$
  $(Set\{F1, F2\}->forAll_{Set}(f|\ \mathbf{0} <_{int}\ f\ .seats@pre))))$
  unfolding *Flight-def*
  apply(*subst StrongEq-L-subst3*[where x=$OclAsType_{Flight}$-$\mathfrak{A}$ *.allInstances@pre*()],
    simp, *simp add*: σ_{t1}-*def* σ_{t1}-σ_{s1}[*simplified* σ_{t1}-*def* σ_{s1}-*def*])
  apply(*rule* $StrictRefEq_{Set}$*.StrictRefEq-vs-StrongEq′*
    [*THEN iffD1*, *OF - - state-*σ_1.σ_1*-OclAllInstances-at-pre-exec-Flight*
      [*OF* σ_1[*simplified state-interpretation-*σ_1*-def*],
    simplified Flight-def]])
  apply(*rule OclAllInstances-at-pre-valid*)
  apply(*simp add*: *F1-def F2-def*)
  by(*simp add*: $OclAsType_{Flight}$-$\mathfrak{A}$*-def*)+
also
have $\ldots = ((\sigma_{t1}, \sigma_{t2}) \models Set\{F1, F2\} \triangleq Set\{\}\ or$
  $(\mathbf{0} <_{int}\ (F2\ .seats@pre))\ and\ (\mathbf{0} <_{int}\ (F1\ .seats@pre)))$
  apply(*simp*, *simp add*: *OclValid-def*, *subst* (*1 2*) *cp-OclOr*,
    subst cp-OclIf, *subst* (*1 2 3*) *cp-OclAnd*, *subst cp-OclIf*)
  by(*simp add*: *F1-def F2-def OclIf-def*)

```
        also
        have    ... = True
                by(simp,rule foundation25', simp add: foundation10' * ** )
        finally show ?thesis
                unfolding Flight-Aat-pre-def  by simp
    qed
next
```

Analogously for the first part, the following auxiliary fact establishes that $\tau \models \delta\ S \Longrightarrow \tau \models S{->}forAll_{Set}(X|P) \triangleq (S \triangleq Set\{\}\ or\ P)$ from the library is applicable since $OclAsType_{Flight}$-$\mathfrak{A}$ $.allInstances()$ is indeed defined.

```
    have forall-trivial: ⋀τ P. let S = OclAsType_Flight-𝔄 .allInstances() in
                (τ ⊨ (S->forAll_Set(X|P) ≜ (S ≜ Set{} or P)))
        by(simp add: Let-def, rule OclForall-body-trivial, rule OclAllInstances-at-post-defined)
    show Flight-A (σ_t1, σ_t2)
    proof −
        have *: (σ_t1, σ_t2) ⊨ 0 <_int F1 .seats
                apply(subst UML-Logic.StrongEq-L-subst3-rev[OF F1-val-seats,
                                                    simplified σ_t2-σ_s2[symmetric]],simp)
                by(simp add: OclInt0')
        have**: (σ_t1, σ_t2) ⊨ 0 <_int F2 .seats
                apply(subst UML-Logic.StrongEq-L-subst3-rev[OF F2-val-seats,
                                                    simplified σ_t2-σ_s2[symmetric]],simp)
                by(simp add: OclInt0')
        have    ((σ_t1, σ_t2) ⊨ Flight .allInstances()->forAll_Set(self|
                                Flight .allInstances()->forAll_Set(f|0 <_int f .seats))) =
                ((σ_t1, σ_t2) ⊨ Flight .allInstances() ≜ Set{} or
                                Flight .allInstances()->forAll_Set(f| 0 <_int f .seats))
                by(simp add: StrongEq-L-subst3[OF - forall-trivial[simplified Let-def],
                                        where P = λx. x])
        also
        have     ... = ((σ_t1, σ_t2) ⊨ Set{F1,F2} ≜ Set{} or
                                Set{F1,F2}->forAll_Set(f| 0 <_int  f .seats))
                unfolding Flight-def
                apply(subst StrongEq-L-subst3[where x = OclAsType_Flight-𝔄 .allInstances()],
                      simp, simp add: σ_t2-def σ_t2-σ_s2[simplified σ_t2-def σ_s2-def])
                apply(rule StrictRefEq_Set.StrictRefEq-vs-StrongEq'
                            [THEN iffD1, OF - - state-σ_2.σ_2-OclAllInstances-at-post-exec-Flight
                                    [OF σ_2[simplified state-interpretation-σ_2-def],
                             simplified Flight-def]])
                apply(rule OclAllInstances-at-post-valid)
                apply(simp add: F1-def F2-def)
                by(simp add: OclAsType_Flight-𝔄-def)+
        also
        have     ... = ((σ_t1, σ_t2) ⊨ Set{F1, F2} ≜ Set{} or
                                (0 <_int  (F2 .seats)) and (0 <_int  (F1 .seats)))
                apply(simp, simp add: OclValid-def, subst (1 2) cp-OclOr,
                      subst cp-OclIf, subst (1 2 3) cp-OclAnd, subst cp-OclIf)
                by(simp add: F1-def F2-def OclIf-def)
        also
        have    ... = True
                 by(simp,rule foundation25', simp add: foundation10' * ** )
        finally show ?thesis
                unfolding Flight-A-def  by simp
    qed
qed
```

Context r: *Reservation*
  Inv A : $\mathbf{0} <_{int} (r\ .id)$
  Inv B : $r\ .next <> null\ implies\ (r\ .flight\ .to \doteq r\ .next\ .flight\ .from)$
  Inv C : $r\ .next <> null\ implies\ (r\ .client \doteq r\ .next\ .client)$

Context *Client* :: *book* (f : *Flight*)
  Pre : $f\ .passengers\ ->excludes_{Set}(self\ .oclAsType(Person))$
        $and\ (f\ .fl\text{-}res\ ->size_{Seq}() <_{int} (f\ .seats))$
  Post: $f\ .passengers \doteq (f\ .passengers@pre\ ->including_{Set}(self\ .oclAsType(Person)))$
        $and\ (let\ r = self\ .cl\text{-}res\ ->select_{Set}(r \mid r\ .flight \doteq f)->any_{Set}()\ in$
            $(r\ .oclIsNew())$
            $and\ (r\ .prev \doteq null)$
            $and\ (r\ .next \doteq null))$

Context *Client* :: *booknext* (f : *Flight*, r : *Reservation*)
  Pre : $f\ .passengers\ ->excludes_{Set}(self\ .oclAsType(Person))$
        $and\ (f\ .fl\text{-}res\ ->size_{Seq}() <_{int} (f\ .seats))$
        $and\ (r\ .client \doteq self)$
        $and\ (f\ .from \doteq (r\ .flight\ .to))$
  Post: $f\ .passengers \doteq (f\ .passengers@pre\ ->including_{Set}(self\ .oclAsType(Person)))$
        $and\ (let\ r = self\ .cl\text{-}res\ ->select_{Set}(r \mid r\ .flight \doteq f)->any_{Set}()\ in$
            $(r\ .oclIsNew())$
            $and\ (r\ .prev \doteq r)$
            $and\ (r\ .next \doteq null))$

Context *Client* :: *cancel* (r : *Reservation*)
  Pre : $r\ .client \doteq self$
  Post: $self\ .cl\text{-}res\ ->select_{Set}(res \mid res\ .flight \doteq r\ .flight@pre)$
            $->isEmpty_{Set}()$

Context *Reservation* :: *connections* () : *Set*(*Integer*)
  Post : $result \triangleq if\ (self\ .next \doteq null)$
            $then\ (Set\{\}->including_{Set}(self\ .id))$
            $else\ (self\ .next\ .connections()->including_{Set}(self\ .id))$
            $endif$
  Pre : $true$

Proving the Implementability of Operations

An operation contract is said to be non-blocking, if and only if there exist input and input states where the pre-condition is satisfied. Moreover, a contract is said to be implementable, if and only if for all inputs satisfying the pre-condition output data exists that satisfies the post-condition.

definition $cancel_{pre} :: (\cdot Client) \Rightarrow (\cdot Reservation) \Rightarrow \cdot Boolean_{base}$
where $cancel_{pre}\ \ self\ r \equiv (r\ .client@pre) \doteq self$

definition $cancel_{post} :: (\cdot Client) \Rightarrow (\cdot Reservation) \Rightarrow (\cdot Void_{base}) \Rightarrow \cdot Boolean_{base}$
where $cancel_{post}\ \ self\ r\ result \equiv self\ .cl\text{-}res->select_{Set}(res|res\ .flight \doteq r\ .flight@pre)->isEmpty_{Set}()$

lemma $cancel_{nonblocking} : \exists\ self\ r\ \sigma.\ \ (\sigma, \sigma') \models (cancel_{pre}\ \ self\ r)$
 apply(*rule exI*[where $x = C1$], *rule exI*[where $x = R11$], *rule exI*[where $x = \sigma_{t1}$])
 using *R11-val-clientATpre*[*simplified OclValid-def StrongEq-def true-def* σ_{t1}-σ_{s1}[*symmetric*], *of* σ']
 apply(*simp add*: $cancel_{pre}$-*def StrictRefEq*$_{Object\text{-}Reservation}$ *StrictRefEq*$_{Object}$-*def OclValid-def*)
by(*subst cp-valid*, *simp*, *subst cp-valid*[*symmetric*],
   simp add: *C1-valid*[*simplified OclValid-def* σ_{t1}-σ_{s1}[*symmetric*]])

```
lemma cancel_nonblocking_pre : ∃ self r σ.  σ ⊨_pre (cancel_pre  self r)
 apply(rule exI[where x = C1], rule exI[where x = R11], rule exI[where x = σ_t1])
 apply(simp add: OclValid-at-pre-def, intro allI)
 proof − fix σ' show (σ_t1, σ') ⊨ cancel_pre C1 R11
 using R11-val-clientATpre[simplified OclValid-def StrongEq-def true-def σ_t1-σ_s1[symmetric], of σ']
    apply(simp add: cancel_pre-def StrictRefEq_Object_Reservation StrictRefEq_Object-def OclValid-at-pre-def
OclValid-def)
 by(subst cp-valid, simp, subst cp-valid[symmetric],
    simp add: C1-valid[simplified OclValid-def σ_t1-σ_s1[symmetric]])
qed

lemma cancel_implementable :
 assumes pre-satisfied: σ ⊨_pre (cancel_pre  self r)
 shows                  ∃ σ' result. ((σ, σ') ⊨ δ self) ⟶
                                     ((σ, σ') ⊨ υ r) ⟶
                                     ((σ, σ') ⊨ (cancel_post  self r result))
proof −
 def σ'' ≡ (| heap = K ⌊in_Client (mk_Client (mkℰ𝒳𝒯_Client 0 None) None)⌋
            , assocs = Map.empty (oid_Client-0---cl-res ↦ []) |)

 have self-definition: ⋀τ. τ ⊨ δ self ⟹ ∃ ta xa x. self τ = ⌊⌊mk_Client (mkℰ𝒳𝒯_Client ta xa) x⌋⌋
  apply(simp add:OclValid-def defined-def true-def false-def split: split-if-asm)
  proof − fix τ show self τ ≠ ⊥ τ ∧ self τ ≠ null τ ⟹
        ∃ ta xa x. self τ = ⌊⌊mk_Client (mkℰ𝒳𝒯_Client ta xa) x⌋⌋
  apply(case-tac self τ, simp add: bot-option-def bot-fun-def, simp)
  proof − fix a show ⌊a⌋ ≠ ⊥ τ ∧ ⌊a⌋ ≠ null τ ⟹
        self τ = ⌊a⌋ ⟹ ∃ ta xa x. a = ⌊mk_Client (mkℰ𝒳𝒯_Client ta xa) x⌋
  apply(case-tac a, simp add: null-fun-def null-option-def bot-option-def, simp)
  proof − fix aa show  ⌊⌊aa⌋⌋ ≠ ⊥ τ ∧ ⌊⌊aa⌋⌋ ≠ null τ ⟹
         self τ = ⌊⌊aa⌋⌋ ⟹
         a = ⌊aa⌋ ⟹ ∃ ta xa x. aa = mk_Client (mkℰ𝒳𝒯_Client ta xa) x
  apply(case-tac aa, simp)
  proof − fix x1 x2 show  self τ = ⌊⌊mk_Client x1 x2⌋⌋ ⟹ ∃ ta xa. x1 = mkℰ𝒳𝒯_Client ta xa
  by(case-tac x1, simp)
  qed qed qed qed

 have self-empty: (σ, σ'') ⊨ δ self ⟹ (σ, σ'') ⊨ (self .cl-res ≜ Set{})
  apply(drule self-definition, elim exE)
  apply(simp add: OclValid-def StrongEq-def dot_Client-0---cl-res)
  apply(simp add: deref-oid_Client-def in-post-state-def, subst (8) σ''-def)
  apply(simp add: Let-def K-def oid-of-option-def deref-assocs_Client-0---cl-res-def deref-assocs-def)
  apply(subst (3) σ''-def, simp add: select_Client--cl-res-def)
  by(simp add: oid-of-ty_Client-def deref-assocs-list-def switch_2-01-def select-object_Set-def select-object-def)

 show ?thesis
  apply(rule exI[where x = σ''], rule exI[where x = null], intro impI)
  apply(simp add: cancel_post-def)
  apply(subst StrongEq-L-subst3[OF - self-empty])
    apply(rule UML-Set.cp-intro''_Set(2))
    apply(simp only: cp-def)
    apply(rule exI[where x = λX τ. (λ-. X)->select_Set(res|StrictRefEq_Object res .flight r .flight@pre) τ],
          subst cp-OclSelect, simp)
 by(simp+)
qed
```

As remark, the pre-condition $\sigma \models_{pre} cancel_{pre}\ self\ r$ has not been used; in the special case of the operation "cancel", the post-condition is satisfiable for *arbitrary* defined and valid input, even input that does not satisfy the pre-condition.

```
end
```

APPENDIX B

The Flight Model (Generated Theory, Floor 1)

This chapter has been generated from Appendix A (by discarding all the Isar_HOL commands of Appendix A and only keeping its meta-commands).

theory *Flight-Model-generated* imports *../src/UML-Main* *../src/compiler/Static* *../src/compiler/Generator-dynamic* begin

B.1 Enum

datatype $ty\text{-}enum_{Week} = constr_{Mon}$
| $constr_{Tue}$
| $constr_{Wed}$
| $constr_{Thu}$
| $constr_{Fri}$
| $constr_{Sat}$
| $constr_{Sun}$

type-synonym $Week_{base} = \langle\langle ty\text{-}enum_{Week}\rangle_\perp\rangle_\perp$

type-synonym $'\mathfrak{A}\ Week_{generic} = ('\mathfrak{A}, Week_{base})\ val$

overloading $StrictRefEq \equiv (StrictRefEq::'\mathfrak{A}\ Week_{generic} \Rightarrow \text{-})$

begin

definition $StrictRefEq_{Week} : (x::'\mathfrak{A}\ Week_{generic}) \doteq y \equiv (\lambda\tau.\ if\ (((v\ (x)))\ (\tau)) = (true\ (\tau)) \wedge (((v\ (y)))\ (\tau)) = (true\ (\tau))\ then\ ((x \triangleq y)\ (\tau))\ else\ (invalid\ (\tau)))$

end

definition $Mon = (\lambda\text{-}.\ \lfloor\lfloor(constr_{Mon}::ty\text{-}enum_{Week})\rfloor\rfloor)$
definition $Tue = (\lambda\text{-}.\ \lfloor\lfloor(constr_{Tue}::ty\text{-}enum_{Week})\rfloor\rfloor)$
definition $Wed = (\lambda\text{-}.\ \lfloor\lfloor(constr_{Wed}::ty\text{-}enum_{Week})\rfloor\rfloor)$
definition $Thu = (\lambda\text{-}.\ \lfloor\lfloor(constr_{Thu}::ty\text{-}enum_{Week})\rfloor\rfloor)$
definition $Fri = (\lambda\text{-}.\ \lfloor\lfloor(constr_{Fri}::ty\text{-}enum_{Week})\rfloor\rfloor)$
definition $Sat = (\lambda\text{-}.\ \lfloor\lfloor(constr_{Sat}::ty\text{-}enum_{Week})\rfloor\rfloor)$
definition $Sun = (\lambda\text{-}.\ \lfloor\lfloor(constr_{Sun}::ty\text{-}enum_{Week})\rfloor\rfloor)$

B.2 Class Model: The Construction of the Object Universe

datatype $ty\mathcal{EXT}_{Flight} = mk\mathcal{EXT}_{Flight}\ oid$
datatype $ty_{Flight} = mk_{Flight}\ ty\mathcal{EXT}_{Flight}\ int\ option\ string\ option\ string\ option\ oid\ list\ option$
datatype $ty\mathcal{EXT}_{Client} = mk\mathcal{EXT}_{Client}\ oid\ string\ option$
datatype $ty_{Client} = mk_{Client}\ ty\mathcal{EXT}_{Client}\ string\ option$
datatype $ty\mathcal{EXT}_{Staff} = mk\mathcal{EXT}_{Staff}\ oid\ string\ option$
datatype $ty_{Staff} = mk_{Staff}\ ty\mathcal{EXT}_{Staff}$
datatype $ty\mathcal{EXT}_{Person} = mk\mathcal{EXT}_{Person\text{-}Staff}\ ty_{Staff}$
| $mk\mathcal{EXT}_{Person\text{-}Client}\ ty_{Client}$
| $mk\mathcal{EXT}_{Person}\ oid$
datatype $ty_{Person} = mk_{Person}\ ty\mathcal{EXT}_{Person}\ string\ option$
datatype $ty\mathcal{EXT}_{Reservation} = mk\mathcal{EXT}_{Reservation}\ oid$
datatype $ty_{Reservation} = mk_{Reservation}\ ty\mathcal{EXT}_{Reservation}\ int\ option\ ty\text{-}enum_{Week}\ option\ oid\ option$
datatype $ty\mathcal{EXT}_{OclAny} = mk\mathcal{EXT}_{OclAny\text{-}Reservation}\ ty_{Reservation}$
| $mk\mathcal{EXT}_{OclAny\text{-}Person}\ ty_{Person}$
| $mk\mathcal{EXT}_{OclAny\text{-}Staff}\ ty_{Staff}$
| $mk\mathcal{EXT}_{OclAny\text{-}Client}\ ty_{Client}$
| $mk\mathcal{EXT}_{OclAny\text{-}Flight}\ ty_{Flight}$
| $mk\mathcal{EXT}_{OclAny}\ oid$
datatype $ty_{OclAny} = mk_{OclAny}\ ty\mathcal{EXT}_{OclAny}$

```
datatype 𝔄 = in_Flight ty_Flight
           | in_Client ty_Client
           | in_Staff ty_Staff
           | in_Person ty_Person
           | in_Reservation ty_Reservation
           | in_OclAny ty_OclAny

type-synonym Void = 𝔄 Void
type-synonym Boolean = 𝔄 Boolean
type-synonym Integer = 𝔄 Integer
type-synonym Real = 𝔄 Real
type-synonym String = 𝔄 String
type-synonym 'α val' = (𝔄, 'α) val
type-notation val' (·(-))

type-synonym Flight = ⟨⟨ty_Flight⟩⊥⟩⊥
type-synonym Client = ⟨⟨ty_Client⟩⊥⟩⊥
type-synonym Staff = ⟨⟨ty_Staff⟩⊥⟩⊥
type-synonym Person = ⟨⟨ty_Person⟩⊥⟩⊥
type-synonym Reservation = ⟨⟨ty_Reservation⟩⊥⟩⊥
type-synonym OclAny = ⟨⟨ty_OclAny⟩⊥⟩⊥

type-synonym Sequence-Person = (𝔄, ty_Person option option Sequence_base) val
type-synonym Set-Person = (𝔄, ty_Person option option Set_base) val
type-synonym Sequence-Flight = (𝔄, ty_Flight option option Sequence_base) val
type-synonym Set-Flight = (𝔄, ty_Flight option option Set_base) val
type-synonym Sequence-Client = (𝔄, ty_Client option option Sequence_base) val
type-synonym Set-Client = (𝔄, ty_Client option option Set_base) val
type-synonym Sequence-Reservation = (𝔄, ty_Reservation option option Sequence_base) val
type-synonym Set-Reservation = (𝔄, ty_Reservation option option Set_base) val

type-synonym Week = 𝔄 Week_generic

instantiation ty_Flight :: object
begin
  definition oid-of-ty_Flight-def : oid-of = (λ mk_Flight t - - - - ⇒ (case t of (mkℰ𝒳𝒯_Flight (t)) ⇒ t))
  instance ..
end
instantiation ty_Client :: object
begin
  definition oid-of-ty_Client-def : oid-of = (λ mk_Client t - ⇒ (case t of (mkℰ𝒳𝒯_Client (t) (-)) ⇒ t))
  instance ..
end
instantiation ty_Staff :: object
begin
  definition oid-of-ty_Staff-def : oid-of = (λ mk_Staff t ⇒ (case t of (mkℰ𝒳𝒯_Staff (t) (-)) ⇒ t))
  instance ..
end
instantiation ty_Person :: object
begin
  definition oid-of-ty_Person-def : oid-of = (λ mk_Person t - ⇒ (case t of (mkℰ𝒳𝒯_Person (t)) ⇒ t
    | (mkℰ𝒳𝒯_Person-Client (t)) ⇒ (oid-of (t))
    | (mkℰ𝒳𝒯_Person-Staff (t)) ⇒ (oid-of (t))))
  instance ..
end
instantiation ty_Reservation :: object
begin
  definition oid-of-ty_Reservation-def : oid-of = (λ mk_Reservation t - - - ⇒ (case t of (mkℰ𝒳𝒯_Reservation (t)) ⇒ t))
  instance ..
end
instantiation ty_OclAny :: object
begin
  definition oid-of-ty_OclAny-def : oid-of = (λ mk_OclAny t ⇒ (case t of (mkℰ𝒳𝒯_OclAny (t)) ⇒ t
    | (mkℰ𝒳𝒯_OclAny-Flight (t)) ⇒ (oid-of (t))
    | (mkℰ𝒳𝒯_OclAny-Client (t)) ⇒ (oid-of (t))
    | (mkℰ𝒳𝒯_OclAny-Staff (t)) ⇒ (oid-of (t))
    | (mkℰ𝒳𝒯_OclAny-Person (t)) ⇒ (oid-of (t))
    | (mkℰ𝒳𝒯_OclAny-Reservation (t)) ⇒ (oid-of (t))))
```

```
  instance ..
end


instantiation 𝔄 :: object
begin
  definition oid-of-𝔄-def : oid-of = (λ in_Flight Flight ⇒ oid-of Flight
    | in_Client Client ⇒ oid-of Client
    | in_Staff Staff ⇒ oid-of Staff
    | in_Person Person ⇒ oid-of Person
    | in_Reservation Reservation ⇒ oid-of Reservation
    | in_OclAny OclAny ⇒ oid-of OclAny)
  instance ..
end
```

B.3 Class Model: Instantiation of the Generic Strict Equality

```
overloading StrictRefEq ≡ (StrictRefEq::(·Flight) ⇒ - ⇒ -)
begin
  definition StrictRefEq_Object-Flight : (x::·Flight) ≐ y ≡ StrictRefEq_Object x y
end
overloading StrictRefEq ≡ (StrictRefEq::(·Client) ⇒ - ⇒ -)
begin
  definition StrictRefEq_Object-Client : (x::·Client) ≐ y ≡ StrictRefEq_Object x y
end
overloading StrictRefEq ≡ (StrictRefEq::(·Staff) ⇒ - ⇒ -)
begin
  definition StrictRefEq_Object-Staff : (x::·Staff) ≐ y ≡ StrictRefEq_Object x y
end
overloading StrictRefEq ≡ (StrictRefEq::(·Person) ⇒ - ⇒ -)
begin
  definition StrictRefEq_Object-Person : (x::·Person) ≐ y ≡ StrictRefEq_Object x y
end
overloading StrictRefEq ≡ (StrictRefEq::(·Reservation) ⇒ - ⇒ -)
begin
  definition StrictRefEq_Object-Reservation : (x::·Reservation) ≐ y ≡ StrictRefEq_Object x y
end
overloading StrictRefEq ≡ (StrictRefEq::(·OclAny) ⇒ - ⇒ -)
begin
  definition StrictRefEq_Object-OclAny : (x::·OclAny) ≐ y ≡ StrictRefEq_Object x y
end


lemmas[simp,code-unfold] = StrictRefEq_Object-Flight
                           StrictRefEq_Object-Client
                           StrictRefEq_Object-Staff
                           StrictRefEq_Object-Person
                           StrictRefEq_Object-Reservation
                           StrictRefEq_Object-OclAny
```

B.4 Class Model: OclAsType

Definition

```
consts OclAsType_Flight :: 'α ⇒ ·Flight ((-) .oclAsType'(Flight'))
consts OclAsType_Client :: 'α ⇒ ·Client ((-) .oclAsType'(Client'))
consts OclAsType_Staff :: 'α ⇒ ·Staff ((-) .oclAsType'(Staff'))
consts OclAsType_Person :: 'α ⇒ ·Person ((-) .oclAsType'(Person'))
consts OclAsType_Reservation :: 'α ⇒ ·Reservation ((-) .oclAsType'(Reservation'))
consts OclAsType_OclAny :: 'α ⇒ ·OclAny ((-) .oclAsType'(OclAny'))


overloading OclAsType_Flight ≡ (OclAsType_Flight::(·Flight) ⇒ -)
begin
  definition OclAsType_Flight-Flight : (x::·Flight) .oclAsType(Flight) ≡ x
end
overloading OclAsType_Flight ≡ (OclAsType_Flight::(·OclAny) ⇒ -)
begin
  definition OclAsType_Flight-OclAny : (x::·OclAny) .oclAsType(Flight) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⌊⊥⌋⌋ ⇒ (null (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Flight (Flight))))⌋⌋ ⇒ ⌊⌊Flight⌋⌋
```

```
  | - ⇒ (invalid (τ))))
end
overloading OclAsType_Flight ≡ (OclAsType_Flight::(·Staff) ⇒ -)
begin
  definition OclAsType_Flight-Staff : (x::·Staff) .oclAsType(Flight) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Flight ≡ (OclAsType_Flight::(·Person) ⇒ -)
begin
  definition OclAsType_Flight-Person : (x::·Person) .oclAsType(Flight) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Flight ≡ (OclAsType_Flight::(·Client) ⇒ -)
begin
  definition OclAsType_Flight-Client : (x::·Client) .oclAsType(Flight) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Flight ≡ (OclAsType_Flight::(·Reservation) ⇒ -)
begin
  definition OclAsType_Flight-Reservation : (x::·Reservation) .oclAsType(Flight) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Client ≡ (OclAsType_Client::(·Client) ⇒ -)
begin
  definition OclAsType_Client-Client : (x::·Client) .oclAsType(Client) ≡ x
end
overloading OclAsType_Client ≡ (OclAsType_Client::(·Person) ⇒ -)
begin
  definition OclAsType_Client-Person : (x::·Person) .oclAsType(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊(mk_Person ((mkℰ𝒳𝒯_Person-Client (Client))) (-))⌋⌋ ⇒ ⌊⌊Client⌋⌋
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Client ≡ (OclAsType_Client::(·OclAny) ⇒ -)
begin
  definition OclAsType_Client-OclAny : (x::·OclAny) .oclAsType(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Client (Client))))⌋⌋ ⇒ ⌊⌊Client⌋⌋
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Client ≡ (OclAsType_Client::(·Staff) ⇒ -)
begin
  definition OclAsType_Client-Staff : (x::·Staff) .oclAsType(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Client ≡ (OclAsType_Client::(·Reservation) ⇒ -)
begin
  definition OclAsType_Client-Reservation : (x::·Reservation) .oclAsType(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Client ≡ (OclAsType_Client::(·Flight) ⇒ -)
begin
  definition OclAsType_Client-Flight : (x::·Flight) .oclAsType(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Staff ≡ (OclAsType_Staff::(·Staff) ⇒ -)
begin
  definition OclAsType_Staff-Staff : (x::·Staff) .oclAsType(Staff) ≡ x
end
overloading OclAsType_Staff ≡ (OclAsType_Staff::(·Person) ⇒ -)
begin
  definition OclAsType_Staff-Person : (x::·Person) .oclAsType(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊(mk_Person ((mkℰ𝒳𝒯_Person-Staff (Staff))) (-))⌋⌋ ⇒ ⌊⌊Staff⌋⌋
    | - ⇒ (invalid (τ))))
end
```

```
overloading OclAsType_Staff ≡ (OclAsType_Staff::(·OclAny) ⇒ -)
begin
  definition OclAsType_Staff-OclAny : (x::·OclAny) .oclAsType(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Staff (Staff))))⌋⌋ ⇒ ⌊⌊Staff⌋⌋
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Staff ≡ (OclAsType_Staff::(·Client) ⇒ -)
begin
  definition OclAsType_Staff-Client : (x::·Client) .oclAsType(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Staff ≡ (OclAsType_Staff::(·Reservation) ⇒ -)
begin
  definition OclAsType_Staff-Reservation : (x::·Reservation) .oclAsType(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Staff ≡ (OclAsType_Staff::(·Flight) ⇒ -)
begin
  definition OclAsType_Staff-Flight : (x::·Flight) .oclAsType(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Person ≡ (OclAsType_Person::(·Person) ⇒ -)
begin
  definition OclAsType_Person-Person : (x::·Person) .oclAsType(Person) ≡ x
end
overloading OclAsType_Person ≡ (OclAsType_Person::(·OclAny) ⇒ -)
begin
  definition OclAsType_Person-OclAny : (x::·OclAny) .oclAsType(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Person (Person))))⌋⌋ ⇒ ⌊⌊Person⌋⌋
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Person ≡ (OclAsType_Person::(·Client) ⇒ -)
begin
  definition OclAsType_Person-Client : (x::·Client) .oclAsType(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊Client⌋⌋ ⇒ ⌊⌊(mk_Person ((mkℰ𝒳𝒯_Person-Client (Client))) (None))⌋⌋))
end
overloading OclAsType_Person ≡ (OclAsType_Person::(·Staff) ⇒ -)
begin
  definition OclAsType_Person-Staff : (x::·Staff) .oclAsType(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊Staff⌋⌋ ⇒ ⌊⌊(mk_Person ((mkℰ𝒳𝒯_Person-Staff (Staff))) (None))⌋⌋))
end
overloading OclAsType_Person ≡ (OclAsType_Person::(·Reservation) ⇒ -)
begin
  definition OclAsType_Person-Reservation : (x::·Reservation) .oclAsType(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Person ≡ (OclAsType_Person::(·Flight) ⇒ -)
begin
  definition OclAsType_Person-Flight : (x::·Flight) .oclAsType(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Reservation ≡ (OclAsType_Reservation::(·Reservation) ⇒ -)
begin
  definition OclAsType_Reservation-Reservation : (x::·Reservation) .oclAsType(Reservation) ≡ x
end
overloading OclAsType_Reservation ≡ (OclAsType_Reservation::(·OclAny) ⇒ -)
begin
  definition OclAsType_Reservation-OclAny : (x::·OclAny) .oclAsType(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Reservation (Reservation))))⌋⌋ ⇒ ⌊⌊Reservation⌋⌋
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Reservation ≡ (OclAsType_Reservation::(·Staff) ⇒ -)
begin
```

```
  definition OclAsType_Reservation-Staff : (x::·Staff) .oclAsType(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Reservation ≡ (OclAsType_Reservation::(·Person) ⇒ -)
begin
  definition OclAsType_Reservation-Person : (x::·Person) .oclAsType(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Reservation ≡ (OclAsType_Reservation::(·Client) ⇒ -)
begin
  definition OclAsType_Reservation-Client : (x::·Client) .oclAsType(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_Reservation ≡ (OclAsType_Reservation::(·Flight) ⇒ -)
begin
  definition OclAsType_Reservation-Flight : (x::·Flight) .oclAsType(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | - ⇒ (invalid (τ))))
end
overloading OclAsType_OclAny ≡ (OclAsType_OclAny::(·OclAny) ⇒ -)
begin
  definition OclAsType_OclAny-OclAny : (x::·OclAny) .oclAsType(OclAny) ≡ x
end
overloading OclAsType_OclAny ≡ (OclAsType_OclAny::(·Flight) ⇒ -)
begin
  definition OclAsType_OclAny-Flight : (x::·Flight) .oclAsType(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊Flight⌋⌋ ⇒ ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Flight (Flight))))⌋⌋))
end
overloading OclAsType_OclAny ≡ (OclAsType_OclAny::(·Client) ⇒ -)
begin
  definition OclAsType_OclAny-Client : (x::·Client) .oclAsType(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊Client⌋⌋ ⇒ ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Client (Client))))⌋⌋))
end
overloading OclAsType_OclAny ≡ (OclAsType_OclAny::(·Staff) ⇒ -)
begin
  definition OclAsType_OclAny-Staff : (x::·Staff) .oclAsType(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊Staff⌋⌋ ⇒ ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Staff (Staff))))⌋⌋))
end
overloading OclAsType_OclAny ≡ (OclAsType_OclAny::(·Person) ⇒ -)
begin
  definition OclAsType_OclAny-Person : (x::·Person) .oclAsType(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊Person⌋⌋ ⇒ ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Person (Person))))⌋⌋))
end
overloading OclAsType_OclAny ≡ (OclAsType_OclAny::(·Reservation) ⇒ -)
begin
  definition OclAsType_OclAny-Reservation : (x::·Reservation) .oclAsType(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (null (τ))
    | ⌊⌊Reservation⌋⌋ ⇒ ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Reservation (Reservation))))⌋⌋))
end


definition OclAsType_Flight-𝔄 = (λ (in_Flight (Flight)) ⇒ ⌊Flight⌋
    | (in_OclAny ((mk_OclAny ((mkℰ𝒳𝒯_OclAny-Flight (Flight)))))) ⇒ ⌊Flight⌋
    | - ⇒ None)
definition OclAsType_Client-𝔄 = (λ (in_Client (Client)) ⇒ ⌊Client⌋
    | (in_Person ((mk_Person ((mkℰ𝒳𝒯_Person-Client (Client))) (-)))) ⇒ ⌊Client⌋
    | (in_OclAny ((mk_OclAny ((mkℰ𝒳𝒯_OclAny-Client (Client)))))) ⇒ ⌊Client⌋
    | - ⇒ None)
definition OclAsType_Staff-𝔄 = (λ (in_Staff (Staff)) ⇒ ⌊Staff⌋
    | (in_Person ((mk_Person ((mkℰ𝒳𝒯_Person-Staff (Staff))) (-)))) ⇒ ⌊Staff⌋
    | (in_OclAny ((mk_OclAny ((mkℰ𝒳𝒯_OclAny-Staff (Staff)))))) ⇒ ⌊Staff⌋
    | - ⇒ None)
definition OclAsType_Person-𝔄 = (λ (in_Person (Person)) ⇒ ⌊Person⌋
    | (in_OclAny ((mk_OclAny ((mkℰ𝒳𝒯_OclAny-Person (Person)))))) ⇒ ⌊Person⌋
    | (in_Client (Client)) ⇒ ⌊(mk_Person ((mkℰ𝒳𝒯_Person-Client (Client))) (None))⌋
    | (in_Staff (Staff)) ⇒ ⌊(mk_Person ((mkℰ𝒳𝒯_Person-Staff (Staff))) (None))⌋
```

| - ⇒ *None*)
definition $OclAsType_{Reservation}$-$\mathfrak{A}$ = (λ ($in_{Reservation}$ (*Reservation*)) ⇒ ⌊*Reservation*⌋
| (in_{OclAny} ((mk_{OclAny} (($mk\mathcal{EXT}_{OclAny\text{-}Reservation}$ (*Reservation*)))))) ⇒ ⌊*Reservation*⌋
| - ⇒ *None*)
definition $OclAsType_{OclAny}$-$\mathfrak{A}$ = *Some o* (λ (in_{OclAny} (*OclAny*)) ⇒ *OclAny*
| (in_{Flight} (*Flight*)) ⇒ (mk_{OclAny} (($mk\mathcal{EXT}_{OclAny\text{-}Flight}$ (*Flight*))))
| (in_{Client} (*Client*)) ⇒ (mk_{OclAny} (($mk\mathcal{EXT}_{OclAny\text{-}Client}$ (*Client*))))
| (in_{Staff} (*Staff*)) ⇒ (mk_{OclAny} (($mk\mathcal{EXT}_{OclAny\text{-}Staff}$ (*Staff*))))
| (in_{Person} (*Person*)) ⇒ (mk_{OclAny} (($mk\mathcal{EXT}_{OclAny\text{-}Person}$ (*Person*))))
| ($in_{Reservation}$ (*Reservation*)) ⇒ (mk_{OclAny} (($mk\mathcal{EXT}_{OclAny\text{-}Reservation}$ (*Reservation*)))))

lemmas[*simp*,*code-unfold*] = $OclAsType_{Flight}$-*Flight*
$OclAsType_{Client}$-*Client*
$OclAsType_{Staff}$-*Staff*
$OclAsType_{Person}$-*Person*
$OclAsType_{Reservation}$-*Reservation*
$OclAsType_{OclAny}$-*OclAny*

Context Passing

lemma $cp\text{-}OclAsType_{Client}$-*Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Client}$-*Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Client*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Client}$-*OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Client}$-*Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Client}$-*Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Client}$-*Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Client}$-*Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Reservation*)
lemma $cp\text{-}OclAsType_{Client}$-*Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Reservation*)
lemma $cp\text{-}OclAsType_{Client}$-*OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Reservation*)
lemma $cp\text{-}OclAsType_{Client}$-*Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Reservation*)
lemma $cp\text{-}OclAsType_{Client}$-*Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Reservation*)
lemma $cp\text{-}OclAsType_{Client}$-*Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Reservation*)
lemma $cp\text{-}OclAsType_{Client}$-*Client-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*OclAny*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*OclAny*)
lemma $cp\text{-}OclAsType_{Client}$-*Reservation-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*OclAny*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*OclAny*)
lemma $cp\text{-}OclAsType_{Client}$-*OclAny-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*OclAny*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*OclAny*)
lemma $cp\text{-}OclAsType_{Client}$-*Person-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*OclAny*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*OclAny*)
lemma $cp\text{-}OclAsType_{Client}$-*Staff-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*OclAny*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*OclAny*)
lemma $cp\text{-}OclAsType_{Client}$-*Flight-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*OclAny*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*OclAny*)
lemma $cp\text{-}OclAsType_{Client}$-*Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Person*)
lemma $cp\text{-}OclAsType_{Client}$-*Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Person*)
lemma $cp\text{-}OclAsType_{Client}$-*OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Person*)
lemma $cp\text{-}OclAsType_{Client}$-*Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Person*)
lemma $cp\text{-}OclAsType_{Client}$-*Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Person*)
lemma $cp\text{-}OclAsType_{Client}$-*Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclAsType*(*Client*)))))
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Person*)
lemma $cp\text{-}OclAsType_{Client}$-*Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclAsType*(*Client*)))))

by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Staff*)
lemma *cp-OclAsType*$_{Client}$-*Reservation-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Staff*)
lemma *cp-OclAsType*$_{Client}$-*OclAny-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Staff*)
lemma *cp-OclAsType*$_{Client}$-*Person-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Staff*)
lemma *cp-OclAsType*$_{Client}$-*Staff-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Staff*)
lemma *cp-OclAsType*$_{Client}$-*Flight-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Staff*)
lemma *cp-OclAsType*$_{Client}$-*Client-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Flight*)
lemma *cp-OclAsType*$_{Client}$-*Reservation-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Flight*)
lemma *cp-OclAsType*$_{Client}$-*OclAny-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Flight*)
lemma *cp-OclAsType*$_{Client}$-*Person-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Flight*)
lemma *cp-OclAsType*$_{Client}$-*Staff-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Flight*)
lemma *cp-OclAsType*$_{Client}$-*Flight-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclAsType(Client)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Client}$-*Flight*)
lemma *cp-OclAsType*$_{Reservation}$-*Client-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*Client*)
lemma *cp-OclAsType*$_{Reservation}$-*Reservation-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*Client*)
lemma *cp-OclAsType*$_{Reservation}$-*OclAny-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*Client*)
lemma *cp-OclAsType*$_{Reservation}$-*Person-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*Client*)
lemma *cp-OclAsType*$_{Reservation}$-*Staff-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Client)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*Client*)
lemma *cp-OclAsType*$_{Reservation}$-*Flight-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Client)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*Client*)
lemma *cp-OclAsType*$_{Reservation}$-*Client-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType*$_{Reservation}$-*Reservation-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType*$_{Reservation}$-*OclAny-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType*$_{Reservation}$-*Person-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType*$_{Reservation}$-*Staff-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType*$_{Reservation}$-*Flight-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType*$_{Reservation}$-*Client-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*OclAny*)
lemma *cp-OclAsType*$_{Reservation}$-*Reservation-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*OclAny*)
lemma *cp-OclAsType*$_{Reservation}$-*OclAny-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*OclAny*)
lemma *cp-OclAsType*$_{Reservation}$-*Person-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*OclAny*)
lemma *cp-OclAsType*$_{Reservation}$-*Staff-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*OclAny*)
lemma *cp-OclAsType*$_{Reservation}$-*Flight-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclAsType(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Reservation}$-*OclAny*)

lemma *cp-OclAsType$_{Reservation}$-Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Person*)
lemma *cp-OclAsType$_{Reservation}$-Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Person*)
lemma *cp-OclAsType$_{Reservation}$-OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Person*)
lemma *cp-OclAsType$_{Reservation}$-Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Person*)
lemma *cp-OclAsType$_{Reservation}$-Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Person*)
lemma *cp-OclAsType$_{Reservation}$-Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Person*)
lemma *cp-OclAsType$_{Reservation}$-Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Staff*)
lemma *cp-OclAsType$_{Reservation}$-Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Staff*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Staff*)
lemma *cp-OclAsType$_{Reservation}$-OclAny-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Staff*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Staff*)
lemma *cp-OclAsType$_{Reservation}$-Person-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Staff*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Staff*)
lemma *cp-OclAsType$_{Reservation}$-Staff-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Staff*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Staff*)
lemma *cp-OclAsType$_{Reservation}$-Flight-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Staff*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Staff*)
lemma *cp-OclAsType$_{Reservation}$-Client-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Flight*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Flight*)
lemma *cp-OclAsType$_{Reservation}$-Reservation-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Flight*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Flight*)
lemma *cp-OclAsType$_{Reservation}$-OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Flight*)
lemma *cp-OclAsType$_{Reservation}$-Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Flight*)
lemma *cp-OclAsType$_{Reservation}$-Staff-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Flight*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Flight*)
lemma *cp-OclAsType$_{Reservation}$-Flight-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Flight*) .*oclAsType*(*Reservation*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Reservation}$-Flight*)
lemma *cp-OclAsType$_{OclAny}$-Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Client*)
lemma *cp-OclAsType$_{OclAny}$-Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Client*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Client*)
lemma *cp-OclAsType$_{OclAny}$-OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Client*)
lemma *cp-OclAsType$_{OclAny}$-Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Client*)
lemma *cp-OclAsType$_{OclAny}$-Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Client*)
lemma *cp-OclAsType$_{OclAny}$-Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Client*)
lemma *cp-OclAsType$_{OclAny}$-Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Reservation*)
lemma *cp-OclAsType$_{OclAny}$-Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Reservation*)
lemma *cp-OclAsType$_{OclAny}$-OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Reservation*)
lemma *cp-OclAsType$_{OclAny}$-Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Reservation*)
lemma *cp-OclAsType$_{OclAny}$-Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Reservation*)
lemma *cp-OclAsType$_{OclAny}$-Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Reservation*)
lemma *cp-OclAsType$_{OclAny}$-Client-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*OclAny*) .*oclAsType*(*OclAny*)))))

by(*rule cpI1*, *simp*)
lemma *cp-OclAsType$_{OclAny}$-Reservation-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*OclAny*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType$_{OclAny}$-OclAny-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*OclAny*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType$_{OclAny}$-Person-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*OclAny*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType$_{OclAny}$-Staff-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*OclAny*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType$_{OclAny}$-Flight-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*OclAny*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclAsType$_{OclAny}$-Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Person*)
lemma *cp-OclAsType$_{OclAny}$-Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Person*)
lemma *cp-OclAsType$_{OclAny}$-OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Person*)
lemma *cp-OclAsType$_{OclAny}$-Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Person*)
lemma *cp-OclAsType$_{OclAny}$-Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Person*)
lemma *cp-OclAsType$_{OclAny}$-Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Person*)
lemma *cp-OclAsType$_{OclAny}$-Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Staff*)
lemma *cp-OclAsType$_{OclAny}$-Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Staff*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Staff*)
lemma *cp-OclAsType$_{OclAny}$-OclAny-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Staff*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Staff*)
lemma *cp-OclAsType$_{OclAny}$-Person-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Staff*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Staff*)
lemma *cp-OclAsType$_{OclAny}$-Staff-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Staff*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Staff*)
lemma *cp-OclAsType$_{OclAny}$-Flight-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Staff*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Staff*)
lemma *cp-OclAsType$_{OclAny}$-Client-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Flight*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Flight*)
lemma *cp-OclAsType$_{OclAny}$-Reservation-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Flight*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Flight*)
lemma *cp-OclAsType$_{OclAny}$-OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Flight*)
lemma *cp-OclAsType$_{OclAny}$-Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Flight*)
lemma *cp-OclAsType$_{OclAny}$-Staff-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Flight*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Flight*)
lemma *cp-OclAsType$_{OclAny}$-Flight-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Flight*) .*oclAsType*(*OclAny*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{OclAny}$-Flight*)
lemma *cp-OclAsType$_{Person}$-Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Client*)
lemma *cp-OclAsType$_{Person}$-Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Client*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Client*)
lemma *cp-OclAsType$_{Person}$-OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Client*)
lemma *cp-OclAsType$_{Person}$-Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Client*)
lemma *cp-OclAsType$_{Person}$-Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Client*)
lemma *cp-OclAsType$_{Person}$-Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Client*)
lemma *cp-OclAsType$_{Person}$-Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Reservation*)
lemma *cp-OclAsType$_{Person}$-Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Reservation*)
lemma *cp-OclAsType$_{Person}$-OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Reservation*)
lemma *cp-OclAsType$_{Person}$-Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclAsType*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclAsType$_{Person}$-Reservation*)
lemma *cp-OclAsType$_{Person}$-Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclAsType*(*Person*)))))

by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Reservation*)
lemma *cp-*$OclAsType_{Person}$*-Flight-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Reservation*)
lemma *cp-*$OclAsType_{Person}$*-Client-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*OclAny*)
lemma *cp-*$OclAsType_{Person}$*-Reservation-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*OclAny*)
lemma *cp-*$OclAsType_{Person}$*-OclAny-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*OclAny*)
lemma *cp-*$OclAsType_{Person}$*-Person-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*OclAny*)
lemma *cp-*$OclAsType_{Person}$*-Staff-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Staff)))::\cdot OclAny)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*OclAny*)
lemma *cp-*$OclAsType_{Person}$*-Flight-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Flight)))::\cdot OclAny)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*OclAny*)
lemma *cp-*$OclAsType_{Person}$*-Client-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Person}$*-Reservation-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Person}$*-OclAny-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Person}$*-Person-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Person}$*-Staff-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Person}$*-Flight-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Person}$*-Client-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::Staff)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Staff*)
lemma *cp-*$OclAsType_{Person}$*-Reservation-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::Staff)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Staff*)
lemma *cp-*$OclAsType_{Person}$*-OclAny-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::Staff)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Staff*)
lemma *cp-*$OclAsType_{Person}$*-Person-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::Staff)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Staff*)
lemma *cp-*$OclAsType_{Person}$*-Staff-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Staff)))::Staff)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Staff*)
lemma *cp-*$OclAsType_{Person}$*-Flight-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Flight)))::Staff)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Staff*)
lemma *cp-*$OclAsType_{Person}$*-Client-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::Flight)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Flight*)
lemma *cp-*$OclAsType_{Person}$*-Reservation-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::Flight)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Flight*)
lemma *cp-*$OclAsType_{Person}$*-OclAny-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::Flight)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Flight*)
lemma *cp-*$OclAsType_{Person}$*-Person-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::Flight)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Flight*)
lemma *cp-*$OclAsType_{Person}$*-Staff-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::Flight)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Flight*)
lemma *cp-*$OclAsType_{Person}$*-Flight-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Flight)))::Flight)\ .oclAsType(Person)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Person}$-*Flight*)
lemma *cp-*$OclAsType_{Staff}$*-Client-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Client*)
lemma *cp-*$OclAsType_{Staff}$*-Reservation-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Client*)
lemma *cp-*$OclAsType_{Staff}$*-OclAny-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Client*)
lemma *cp-*$OclAsType_{Staff}$*-Person-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Client*)
lemma *cp-*$OclAsType_{Staff}$*-Staff-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Staff)))::\cdot Client)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Client*)
lemma *cp-*$OclAsType_{Staff}$*-Flight-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Flight)))::\cdot Client)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Client*)
lemma *cp-*$OclAsType_{Staff}$*-Client-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Reservation*)
lemma *cp-*$OclAsType_{Staff}$*-Reservation-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Reservation*)
lemma *cp-*$OclAsType_{Staff}$*-OclAny-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Reservation*)
lemma *cp-*$OclAsType_{Staff}$*-Person-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclAsType(Staff)))))$

by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Reservation*)
lemma *cp-*$OclAsType_{Staff}$*-Staff-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Reservation*)
lemma *cp-*$OclAsType_{Staff}$*-Flight-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Reservation*)
lemma *cp-*$OclAsType_{Staff}$*-Client-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*OclAny*)
lemma *cp-*$OclAsType_{Staff}$*-Reservation-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*OclAny*)
lemma *cp-*$OclAsType_{Staff}$*-OclAny-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*OclAny*)
lemma *cp-*$OclAsType_{Staff}$*-Person-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*OclAny*)
lemma *cp-*$OclAsType_{Staff}$*-Staff-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*OclAny*)
lemma *cp-*$OclAsType_{Staff}$*-Flight-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*OclAny*)
lemma *cp-*$OclAsType_{Staff}$*-Client-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Person*)
lemma *cp-*$OclAsType_{Staff}$*-Reservation-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Person*)
lemma *cp-*$OclAsType_{Staff}$*-OclAny-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Person*)
lemma *cp-*$OclAsType_{Staff}$*-Person-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Person*)
lemma *cp-*$OclAsType_{Staff}$*-Staff-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Person*)
lemma *cp-*$OclAsType_{Staff}$*-Flight-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Person*)
lemma *cp-*$OclAsType_{Staff}$*-Client-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Staff)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Staff}$*-Reservation-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Staff}$*-OclAny-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Staff}$*-Person-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Staff}$*-Staff-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Staff}$*-Flight-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-*$OclAsType_{Staff}$*-Client-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Flight*)
lemma *cp-*$OclAsType_{Staff}$*-Reservation-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Flight*)
lemma *cp-*$OclAsType_{Staff}$*-OclAny-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Flight*)
lemma *cp-*$OclAsType_{Staff}$*-Person-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Flight*)
lemma *cp-*$OclAsType_{Staff}$*-Staff-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Flight*)
lemma *cp-*$OclAsType_{Staff}$*-Flight-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclAsType(Staff)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Staff}$-*Flight*)
lemma *cp-*$OclAsType_{Flight}$*-Client-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Client*)
lemma *cp-*$OclAsType_{Flight}$*-Reservation-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Client*)
lemma *cp-*$OclAsType_{Flight}$*-OclAny-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Client*)
lemma *cp-*$OclAsType_{Flight}$*-Person-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Client*)
lemma *cp-*$OclAsType_{Flight}$*-Staff-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Client)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Client*)
lemma *cp-*$OclAsType_{Flight}$*-Flight-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Client)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Client*)
lemma *cp-*$OclAsType_{Flight}$*-Client-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Reservation*)
lemma *cp-*$OclAsType_{Flight}$*-Reservation-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Reservation*)
lemma *cp-*$OclAsType_{Flight}$*-OclAny-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}$-*Reservation*)

lemma $cp\text{-}OclAsType_{Flight}\text{-}Person\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Reservation$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Staff\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Reservation$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Flight\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Reservation$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Client\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}OclAny$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Reservation\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}OclAny$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}OclAny\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}OclAny$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Person\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}OclAny$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Staff\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}OclAny$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Flight\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}OclAny$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Client\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Person$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Reservation\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Person$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}OclAny\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Person$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Person\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Person$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Staff\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Person$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Flight\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Person$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Client\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Staff)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Reservation\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}OclAny\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Person\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Staff\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Flight\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp add*: $OclAsType_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Client\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Reservation\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Flight}\text{-}OclAny\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Person\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Staff\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclAsType_{Flight}\text{-}Flight\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclAsType(Flight)))))$
by(*rule cpI1*, *simp*)

lemmas[*simp*,*code-unfold*] = $cp\text{-}OclAsType_{Client}\text{-}Client\text{-}Client$
$cp\text{-}OclAsType_{Client}\text{-}Reservation\text{-}Client$
$cp\text{-}OclAsType_{Client}\text{-}OclAny\text{-}Client$
$cp\text{-}OclAsType_{Client}\text{-}Person\text{-}Client$
$cp\text{-}OclAsType_{Client}\text{-}Staff\text{-}Client$
$cp\text{-}OclAsType_{Client}\text{-}Flight\text{-}Client$
$cp\text{-}OclAsType_{Client}\text{-}Client\text{-}Reservation$
$cp\text{-}OclAsType_{Client}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclAsType_{Client}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclAsType_{Client}\text{-}Person\text{-}Reservation$
$cp\text{-}OclAsType_{Client}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclAsType_{Client}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclAsType_{Client}\text{-}Client\text{-}OclAny$
$cp\text{-}OclAsType_{Client}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclAsType_{Client}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclAsType_{Client}\text{-}Person\text{-}OclAny$

$cp\text{-}OclAsType_{Client}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclAsType_{Client}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclAsType_{Client}\text{-}Client\text{-}Person$
$cp\text{-}OclAsType_{Client}\text{-}Reservation\text{-}Person$
$cp\text{-}OclAsType_{Client}\text{-}OclAny\text{-}Person$
$cp\text{-}OclAsType_{Client}\text{-}Person\text{-}Person$
$cp\text{-}OclAsType_{Client}\text{-}Staff\text{-}Person$
$cp\text{-}OclAsType_{Client}\text{-}Flight\text{-}Person$
$cp\text{-}OclAsType_{Client}\text{-}Client\text{-}Staff$
$cp\text{-}OclAsType_{Client}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclAsType_{Client}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclAsType_{Client}\text{-}Person\text{-}Staff$
$cp\text{-}OclAsType_{Client}\text{-}Staff\text{-}Staff$
$cp\text{-}OclAsType_{Client}\text{-}Flight\text{-}Staff$
$cp\text{-}OclAsType_{Client}\text{-}Client\text{-}Flight$
$cp\text{-}OclAsType_{Client}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclAsType_{Client}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclAsType_{Client}\text{-}Person\text{-}Flight$
$cp\text{-}OclAsType_{Client}\text{-}Staff\text{-}Flight$
$cp\text{-}OclAsType_{Client}\text{-}Flight\text{-}Flight$
$cp\text{-}OclAsType_{Reservation}\text{-}Client\text{-}Client$
$cp\text{-}OclAsType_{Reservation}\text{-}Reservation\text{-}Client$
$cp\text{-}OclAsType_{Reservation}\text{-}OclAny\text{-}Client$
$cp\text{-}OclAsType_{Reservation}\text{-}Person\text{-}Client$
$cp\text{-}OclAsType_{Reservation}\text{-}Staff\text{-}Client$
$cp\text{-}OclAsType_{Reservation}\text{-}Flight\text{-}Client$
$cp\text{-}OclAsType_{Reservation}\text{-}Client\text{-}Reservation$
$cp\text{-}OclAsType_{Reservation}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclAsType_{Reservation}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclAsType_{Reservation}\text{-}Person\text{-}Reservation$
$cp\text{-}OclAsType_{Reservation}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclAsType_{Reservation}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclAsType_{Reservation}\text{-}Client\text{-}OclAny$
$cp\text{-}OclAsType_{Reservation}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclAsType_{Reservation}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclAsType_{Reservation}\text{-}Person\text{-}OclAny$
$cp\text{-}OclAsType_{Reservation}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclAsType_{Reservation}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclAsType_{Reservation}\text{-}Client\text{-}Person$
$cp\text{-}OclAsType_{Reservation}\text{-}Reservation\text{-}Person$
$cp\text{-}OclAsType_{Reservation}\text{-}OclAny\text{-}Person$
$cp\text{-}OclAsType_{Reservation}\text{-}Person\text{-}Person$
$cp\text{-}OclAsType_{Reservation}\text{-}Staff\text{-}Person$
$cp\text{-}OclAsType_{Reservation}\text{-}Flight\text{-}Person$
$cp\text{-}OclAsType_{Reservation}\text{-}Client\text{-}Staff$
$cp\text{-}OclAsType_{Reservation}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclAsType_{Reservation}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclAsType_{Reservation}\text{-}Person\text{-}Staff$
$cp\text{-}OclAsType_{Reservation}\text{-}Staff\text{-}Staff$
$cp\text{-}OclAsType_{Reservation}\text{-}Flight\text{-}Staff$
$cp\text{-}OclAsType_{Reservation}\text{-}Client\text{-}Flight$
$cp\text{-}OclAsType_{Reservation}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclAsType_{Reservation}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclAsType_{Reservation}\text{-}Person\text{-}Flight$
$cp\text{-}OclAsType_{Reservation}\text{-}Staff\text{-}Flight$
$cp\text{-}OclAsType_{Reservation}\text{-}Flight\text{-}Flight$
$cp\text{-}OclAsType_{OclAny}\text{-}Client\text{-}Client$
$cp\text{-}OclAsType_{OclAny}\text{-}Reservation\text{-}Client$
$cp\text{-}OclAsType_{OclAny}\text{-}OclAny\text{-}Client$
$cp\text{-}OclAsType_{OclAny}\text{-}Person\text{-}Client$
$cp\text{-}OclAsType_{OclAny}\text{-}Staff\text{-}Client$
$cp\text{-}OclAsType_{OclAny}\text{-}Flight\text{-}Client$
$cp\text{-}OclAsType_{OclAny}\text{-}Client\text{-}Reservation$
$cp\text{-}OclAsType_{OclAny}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclAsType_{OclAny}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclAsType_{OclAny}\text{-}Person\text{-}Reservation$
$cp\text{-}OclAsType_{OclAny}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclAsType_{OclAny}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclAsType_{OclAny}\text{-}Client\text{-}OclAny$
$cp\text{-}OclAsType_{OclAny}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclAsType_{OclAny}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclAsType_{OclAny}\text{-}Person\text{-}OclAny$
$cp\text{-}OclAsType_{OclAny}\text{-}Staff\text{-}OclAny$

$cp\text{-}OclAsType_{OclAny}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclAsType_{OclAny}\text{-}Client\text{-}Person$
$cp\text{-}OclAsType_{OclAny}\text{-}Reservation\text{-}Person$
$cp\text{-}OclAsType_{OclAny}\text{-}OclAny\text{-}Person$
$cp\text{-}OclAsType_{OclAny}\text{-}Person\text{-}Person$
$cp\text{-}OclAsType_{OclAny}\text{-}Staff\text{-}Person$
$cp\text{-}OclAsType_{OclAny}\text{-}Flight\text{-}Person$
$cp\text{-}OclAsType_{OclAny}\text{-}Client\text{-}Staff$
$cp\text{-}OclAsType_{OclAny}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclAsType_{OclAny}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclAsType_{OclAny}\text{-}Person\text{-}Staff$
$cp\text{-}OclAsType_{OclAny}\text{-}Staff\text{-}Staff$
$cp\text{-}OclAsType_{OclAny}\text{-}Flight\text{-}Staff$
$cp\text{-}OclAsType_{OclAny}\text{-}Client\text{-}Flight$
$cp\text{-}OclAsType_{OclAny}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclAsType_{OclAny}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclAsType_{OclAny}\text{-}Person\text{-}Flight$
$cp\text{-}OclAsType_{OclAny}\text{-}Staff\text{-}Flight$
$cp\text{-}OclAsType_{OclAny}\text{-}Flight\text{-}Flight$
$cp\text{-}OclAsType_{Person}\text{-}Client\text{-}Client$
$cp\text{-}OclAsType_{Person}\text{-}Reservation\text{-}Client$
$cp\text{-}OclAsType_{Person}\text{-}OclAny\text{-}Client$
$cp\text{-}OclAsType_{Person}\text{-}Person\text{-}Client$
$cp\text{-}OclAsType_{Person}\text{-}Staff\text{-}Client$
$cp\text{-}OclAsType_{Person}\text{-}Flight\text{-}Client$
$cp\text{-}OclAsType_{Person}\text{-}Client\text{-}Reservation$
$cp\text{-}OclAsType_{Person}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclAsType_{Person}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclAsType_{Person}\text{-}Person\text{-}Reservation$
$cp\text{-}OclAsType_{Person}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclAsType_{Person}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclAsType_{Person}\text{-}Client\text{-}OclAny$
$cp\text{-}OclAsType_{Person}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclAsType_{Person}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclAsType_{Person}\text{-}Person\text{-}OclAny$
$cp\text{-}OclAsType_{Person}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclAsType_{Person}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclAsType_{Person}\text{-}Client\text{-}Person$
$cp\text{-}OclAsType_{Person}\text{-}Reservation\text{-}Person$
$cp\text{-}OclAsType_{Person}\text{-}OclAny\text{-}Person$
$cp\text{-}OclAsType_{Person}\text{-}Person\text{-}Person$
$cp\text{-}OclAsType_{Person}\text{-}Staff\text{-}Person$
$cp\text{-}OclAsType_{Person}\text{-}Flight\text{-}Person$
$cp\text{-}OclAsType_{Person}\text{-}Client\text{-}Staff$
$cp\text{-}OclAsType_{Person}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclAsType_{Person}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclAsType_{Person}\text{-}Person\text{-}Staff$
$cp\text{-}OclAsType_{Person}\text{-}Staff\text{-}Staff$
$cp\text{-}OclAsType_{Person}\text{-}Flight\text{-}Staff$
$cp\text{-}OclAsType_{Person}\text{-}Client\text{-}Flight$
$cp\text{-}OclAsType_{Person}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclAsType_{Person}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclAsType_{Person}\text{-}Person\text{-}Flight$
$cp\text{-}OclAsType_{Person}\text{-}Staff\text{-}Flight$
$cp\text{-}OclAsType_{Person}\text{-}Flight\text{-}Flight$
$cp\text{-}OclAsType_{Staff}\text{-}Client\text{-}Client$
$cp\text{-}OclAsType_{Staff}\text{-}Reservation\text{-}Client$
$cp\text{-}OclAsType_{Staff}\text{-}OclAny\text{-}Client$
$cp\text{-}OclAsType_{Staff}\text{-}Person\text{-}Client$
$cp\text{-}OclAsType_{Staff}\text{-}Staff\text{-}Client$
$cp\text{-}OclAsType_{Staff}\text{-}Flight\text{-}Client$
$cp\text{-}OclAsType_{Staff}\text{-}Client\text{-}Reservation$
$cp\text{-}OclAsType_{Staff}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclAsType_{Staff}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclAsType_{Staff}\text{-}Person\text{-}Reservation$
$cp\text{-}OclAsType_{Staff}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclAsType_{Staff}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclAsType_{Staff}\text{-}Client\text{-}OclAny$
$cp\text{-}OclAsType_{Staff}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclAsType_{Staff}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclAsType_{Staff}\text{-}Person\text{-}OclAny$
$cp\text{-}OclAsType_{Staff}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclAsType_{Staff}\text{-}Flight\text{-}OclAny$

cp-OclAsType$_{Staff}$-Client-Person
cp-OclAsType$_{Staff}$-Reservation-Person
cp-OclAsType$_{Staff}$-OclAny-Person
cp-OclAsType$_{Staff}$-Person-Person
cp-OclAsType$_{Staff}$-Staff-Person
cp-OclAsType$_{Staff}$-Flight-Person
cp-OclAsType$_{Staff}$-Client-Staff
cp-OclAsType$_{Staff}$-Reservation-Staff
cp-OclAsType$_{Staff}$-OclAny-Staff
cp-OclAsType$_{Staff}$-Person-Staff
cp-OclAsType$_{Staff}$-Staff-Staff
cp-OclAsType$_{Staff}$-Flight-Staff
cp-OclAsType$_{Staff}$-Client-Flight
cp-OclAsType$_{Staff}$-Reservation-Flight
cp-OclAsType$_{Staff}$-OclAny-Flight
cp-OclAsType$_{Staff}$-Person-Flight
cp-OclAsType$_{Staff}$-Staff-Flight
cp-OclAsType$_{Staff}$-Flight-Flight
cp-OclAsType$_{Flight}$-Client-Client
cp-OclAsType$_{Flight}$-Reservation-Client
cp-OclAsType$_{Flight}$-OclAny-Client
cp-OclAsType$_{Flight}$-Person-Client
cp-OclAsType$_{Flight}$-Staff-Client
cp-OclAsType$_{Flight}$-Flight-Client
cp-OclAsType$_{Flight}$-Client-Reservation
cp-OclAsType$_{Flight}$-Reservation-Reservation
cp-OclAsType$_{Flight}$-OclAny-Reservation
cp-OclAsType$_{Flight}$-Person-Reservation
cp-OclAsType$_{Flight}$-Staff-Reservation
cp-OclAsType$_{Flight}$-Flight-Reservation
cp-OclAsType$_{Flight}$-Client-OclAny
cp-OclAsType$_{Flight}$-Reservation-OclAny
cp-OclAsType$_{Flight}$-OclAny-OclAny
cp-OclAsType$_{Flight}$-Person-OclAny
cp-OclAsType$_{Flight}$-Staff-OclAny
cp-OclAsType$_{Flight}$-Flight-OclAny
cp-OclAsType$_{Flight}$-Client-Person
cp-OclAsType$_{Flight}$-Reservation-Person
cp-OclAsType$_{Flight}$-OclAny-Person
cp-OclAsType$_{Flight}$-Person-Person
cp-OclAsType$_{Flight}$-Staff-Person
cp-OclAsType$_{Flight}$-Flight-Person
cp-OclAsType$_{Flight}$-Client-Staff
cp-OclAsType$_{Flight}$-Reservation-Staff
cp-OclAsType$_{Flight}$-OclAny-Staff
cp-OclAsType$_{Flight}$-Person-Staff
cp-OclAsType$_{Flight}$-Staff-Staff
cp-OclAsType$_{Flight}$-Flight-Staff
cp-OclAsType$_{Flight}$-Client-Flight
cp-OclAsType$_{Flight}$-Reservation-Flight
cp-OclAsType$_{Flight}$-OclAny-Flight
cp-OclAsType$_{Flight}$-Person-Flight
cp-OclAsType$_{Flight}$-Staff-Flight
cp-OclAsType$_{Flight}$-Flight-Flight

Execution with Invalid or Null as Argument

lemma *OclAsType$_{Client}$-Client-invalid* : ((*invalid*::*Client*) .*oclAsType*(*Client*)) = *invalid*
by(*simp*)
lemma *OclAsType$_{Client}$-Reservation-invalid* : ((*invalid*::*Reservation*) .*oclAsType*(*Client*)) = *invalid*
by(*rule ext*, *simp add*: *OclAsType$_{Client}$-Reservation bot-option-def invalid-def*)
lemma *OclAsType$_{Client}$-OclAny-invalid* : ((*invalid*::*OclAny*) .*oclAsType*(*Client*)) = *invalid*
by(*rule ext*, *simp add*: *OclAsType$_{Client}$-OclAny bot-option-def invalid-def*)
lemma *OclAsType$_{Client}$-Person-invalid* : ((*invalid*::*Person*) .*oclAsType*(*Client*)) = *invalid*
by(*rule ext*, *simp add*: *OclAsType$_{Client}$-Person bot-option-def invalid-def*)
lemma *OclAsType$_{Client}$-Staff-invalid* : ((*invalid*::*Staff*) .*oclAsType*(*Client*)) = *invalid*
by(*rule ext*, *simp add*: *OclAsType$_{Client}$-Staff bot-option-def invalid-def*)
lemma *OclAsType$_{Client}$-Flight-invalid* : ((*invalid*::*Flight*) .*oclAsType*(*Client*)) = *invalid*
by(*rule ext*, *simp add*: *OclAsType$_{Client}$-Flight bot-option-def invalid-def*)
lemma *OclAsType$_{Client}$-Client-null* : ((*null*::*Client*) .*oclAsType*(*Client*)) = *null*
by(*simp*)
lemma *OclAsType$_{Client}$-Reservation-null* : ((*null*::*Reservation*) .*oclAsType*(*Client*)) = *null*
by(*rule ext*, *simp add*: *OclAsType$_{Client}$-Reservation bot-option-def null-fun-def null-option-def*)

lemma $OclAsType_{Client}$-OclAny-null : ((null::·OclAny) .oclAsType(Client)) = null
by(rule ext, simp add: $OclAsType_{Client}$-OclAny bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Client}$-Person-null : ((null::·Person) .oclAsType(Client)) = null
by(rule ext, simp add: $OclAsType_{Client}$-Person bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Client}$-Staff-null : ((null::·Staff) .oclAsType(Client)) = null
by(rule ext, simp add: $OclAsType_{Client}$-Staff bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Client}$-Flight-null : ((null::·Flight) .oclAsType(Client)) = null
by(rule ext, simp add: $OclAsType_{Client}$-Flight bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Reservation}$-Client-invalid : ((invalid::·Client) .oclAsType(Reservation)) = invalid
by(rule ext, simp add: $OclAsType_{Reservation}$-Client bot-option-def invalid-def)
lemma $OclAsType_{Reservation}$-Reservation-invalid : ((invalid::·Reservation) .oclAsType(Reservation)) = invalid
by(simp)
lemma $OclAsType_{Reservation}$-OclAny-invalid : ((invalid::·OclAny) .oclAsType(Reservation)) = invalid
by(rule ext, simp add: $OclAsType_{Reservation}$-OclAny bot-option-def invalid-def)
lemma $OclAsType_{Reservation}$-Person-invalid : ((invalid::·Person) .oclAsType(Reservation)) = invalid
by(rule ext, simp add: $OclAsType_{Reservation}$-Person bot-option-def invalid-def)
lemma $OclAsType_{Reservation}$-Staff-invalid : ((invalid::·Staff) .oclAsType(Reservation)) = invalid
by(rule ext, simp add: $OclAsType_{Reservation}$-Staff bot-option-def invalid-def)
lemma $OclAsType_{Reservation}$-Flight-invalid : ((invalid::·Flight) .oclAsType(Reservation)) = invalid
by(rule ext, simp add: $OclAsType_{Reservation}$-Flight bot-option-def invalid-def)
lemma $OclAsType_{Reservation}$-Client-null : ((null::·Client) .oclAsType(Reservation)) = null
by(rule ext, simp add: $OclAsType_{Reservation}$-Client bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Reservation}$-Reservation-null : ((null::·Reservation) .oclAsType(Reservation)) = null
by(simp)
lemma $OclAsType_{Reservation}$-OclAny-null : ((null::·OclAny) .oclAsType(Reservation)) = null
by(rule ext, simp add: $OclAsType_{Reservation}$-OclAny bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Reservation}$-Person-null : ((null::·Person) .oclAsType(Reservation)) = null
by(rule ext, simp add: $OclAsType_{Reservation}$-Person bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Reservation}$-Staff-null : ((null::·Staff) .oclAsType(Reservation)) = null
by(rule ext, simp add: $OclAsType_{Reservation}$-Staff bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Reservation}$-Flight-null : ((null::·Flight) .oclAsType(Reservation)) = null
by(rule ext, simp add: $OclAsType_{Reservation}$-Flight bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{OclAny}$-Client-invalid : ((invalid::·Client) .oclAsType(OclAny)) = invalid
by(rule ext, simp add: $OclAsType_{OclAny}$-Client bot-option-def invalid-def)
lemma $OclAsType_{OclAny}$-Reservation-invalid : ((invalid::·Reservation) .oclAsType(OclAny)) = invalid
by(rule ext, simp add: $OclAsType_{OclAny}$-Reservation bot-option-def invalid-def)
lemma $OclAsType_{OclAny}$-OclAny-invalid : ((invalid::·OclAny) .oclAsType(OclAny)) = invalid
by(simp)
lemma $OclAsType_{OclAny}$-Person-invalid : ((invalid::·Person) .oclAsType(OclAny)) = invalid
by(rule ext, simp add: $OclAsType_{OclAny}$-Person bot-option-def invalid-def)
lemma $OclAsType_{OclAny}$-Staff-invalid : ((invalid::·Staff) .oclAsType(OclAny)) = invalid
by(rule ext, simp add: $OclAsType_{OclAny}$-Staff bot-option-def invalid-def)
lemma $OclAsType_{OclAny}$-Flight-invalid : ((invalid::·Flight) .oclAsType(OclAny)) = invalid
by(rule ext, simp add: $OclAsType_{OclAny}$-Flight bot-option-def invalid-def)
lemma $OclAsType_{OclAny}$-Client-null : ((null::·Client) .oclAsType(OclAny)) = null
by(rule ext, simp add: $OclAsType_{OclAny}$-Client bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{OclAny}$-Reservation-null : ((null::·Reservation) .oclAsType(OclAny)) = null
by(rule ext, simp add: $OclAsType_{OclAny}$-Reservation bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{OclAny}$-OclAny-null : ((null::·OclAny) .oclAsType(OclAny)) = null
by(simp)
lemma $OclAsType_{OclAny}$-Person-null : ((null::·Person) .oclAsType(OclAny)) = null
by(rule ext, simp add: $OclAsType_{OclAny}$-Person bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{OclAny}$-Staff-null : ((null::·Staff) .oclAsType(OclAny)) = null
by(rule ext, simp add: $OclAsType_{OclAny}$-Staff bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{OclAny}$-Flight-null : ((null::·Flight) .oclAsType(OclAny)) = null
by(rule ext, simp add: $OclAsType_{OclAny}$-Flight bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Person}$-Client-invalid : ((invalid::·Client) .oclAsType(Person)) = invalid
by(rule ext, simp add: $OclAsType_{Person}$-Client bot-option-def invalid-def)
lemma $OclAsType_{Person}$-Reservation-invalid : ((invalid::·Reservation) .oclAsType(Person)) = invalid
by(rule ext, simp add: $OclAsType_{Person}$-Reservation bot-option-def invalid-def)
lemma $OclAsType_{Person}$-OclAny-invalid : ((invalid::·OclAny) .oclAsType(Person)) = invalid
by(rule ext, simp add: $OclAsType_{Person}$-OclAny bot-option-def invalid-def)
lemma $OclAsType_{Person}$-Person-invalid : ((invalid::·Person) .oclAsType(Person)) = invalid
by(simp)
lemma $OclAsType_{Person}$-Staff-invalid : ((invalid::·Staff) .oclAsType(Person)) = invalid
by(rule ext, simp add: $OclAsType_{Person}$-Staff bot-option-def invalid-def)
lemma $OclAsType_{Person}$-Flight-invalid : ((invalid::·Flight) .oclAsType(Person)) = invalid
by(rule ext, simp add: $OclAsType_{Person}$-Flight bot-option-def invalid-def)
lemma $OclAsType_{Person}$-Client-null : ((null::·Client) .oclAsType(Person)) = null
by(rule ext, simp add: $OclAsType_{Person}$-Client bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Person}$-Reservation-null : ((null::·Reservation) .oclAsType(Person)) = null
by(rule ext, simp add: $OclAsType_{Person}$-Reservation bot-option-def null-fun-def null-option-def)
lemma $OclAsType_{Person}$-OclAny-null : ((null::·OclAny) .oclAsType(Person)) = null

by(*rule ext*, *simp add*: $OclAsType_{Person}$-*OclAny bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Person}$-*Person-null* : ((*null*::·*Person*) .*oclAsType*(*Person*)) = *null*
by(*simp*)
lemma $OclAsType_{Person}$-*Staff-null* : ((*null*::·*Staff*) .*oclAsType*(*Person*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Person}$-*Staff bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Person}$-*Flight-null* : ((*null*::·*Flight*) .*oclAsType*(*Person*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Person}$-*Flight bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Staff}$-*Client-invalid* : ((*invalid*::·*Client*) .*oclAsType*(*Staff*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Client bot-option-def invalid-def*)
lemma $OclAsType_{Staff}$-*Reservation-invalid* : ((*invalid*::·*Reservation*) .*oclAsType*(*Staff*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Reservation bot-option-def invalid-def*)
lemma $OclAsType_{Staff}$-*OclAny-invalid* : ((*invalid*::·*OclAny*) .*oclAsType*(*Staff*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*OclAny bot-option-def invalid-def*)
lemma $OclAsType_{Staff}$-*Person-invalid* : ((*invalid*::·*Person*) .*oclAsType*(*Staff*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Person bot-option-def invalid-def*)
lemma $OclAsType_{Staff}$-*Staff-invalid* : ((*invalid*::·*Staff*) .*oclAsType*(*Staff*)) = *invalid*
by(*simp*)
lemma $OclAsType_{Staff}$-*Flight-invalid* : ((*invalid*::·*Flight*) .*oclAsType*(*Staff*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Flight bot-option-def invalid-def*)
lemma $OclAsType_{Staff}$-*Client-null* : ((*null*::·*Client*) .*oclAsType*(*Staff*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Client bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Staff}$-*Reservation-null* : ((*null*::·*Reservation*) .*oclAsType*(*Staff*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Reservation bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Staff}$-*OclAny-null* : ((*null*::·*OclAny*) .*oclAsType*(*Staff*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*OclAny bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Staff}$-*Person-null* : ((*null*::·*Person*) .*oclAsType*(*Staff*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Person bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Staff}$-*Staff-null* : ((*null*::·*Staff*) .*oclAsType*(*Staff*)) = *null*
by(*simp*)
lemma $OclAsType_{Staff}$-*Flight-null* : ((*null*::·*Flight*) .*oclAsType*(*Staff*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Staff}$-*Flight bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Flight}$-*Client-invalid* : ((*invalid*::·*Client*) .*oclAsType*(*Flight*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Client bot-option-def invalid-def*)
lemma $OclAsType_{Flight}$-*Reservation-invalid* : ((*invalid*::·*Reservation*) .*oclAsType*(*Flight*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Reservation bot-option-def invalid-def*)
lemma $OclAsType_{Flight}$-*OclAny-invalid* : ((*invalid*::·*OclAny*) .*oclAsType*(*Flight*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*OclAny bot-option-def invalid-def*)
lemma $OclAsType_{Flight}$-*Person-invalid* : ((*invalid*::·*Person*) .*oclAsType*(*Flight*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Person bot-option-def invalid-def*)
lemma $OclAsType_{Flight}$-*Staff-invalid* : ((*invalid*::·*Staff*) .*oclAsType*(*Flight*)) = *invalid*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Staff bot-option-def invalid-def*)
lemma $OclAsType_{Flight}$-*Flight-invalid* : ((*invalid*::·*Flight*) .*oclAsType*(*Flight*)) = *invalid*
by(*simp*)
lemma $OclAsType_{Flight}$-*Client-null* : ((*null*::·*Client*) .*oclAsType*(*Flight*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Client bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Flight}$-*Reservation-null* : ((*null*::·*Reservation*) .*oclAsType*(*Flight*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Reservation bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Flight}$-*OclAny-null* : ((*null*::·*OclAny*) .*oclAsType*(*Flight*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*OclAny bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Flight}$-*Person-null* : ((*null*::·*Person*) .*oclAsType*(*Flight*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Person bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Flight}$-*Staff-null* : ((*null*::·*Staff*) .*oclAsType*(*Flight*)) = *null*
by(*rule ext*, *simp add*: $OclAsType_{Flight}$-*Staff bot-option-def null-fun-def null-option-def*)
lemma $OclAsType_{Flight}$-*Flight-null* : ((*null*::·*Flight*) .*oclAsType*(*Flight*)) = *null*
by(*simp*)

lemmas[*simp*,*code-unfold*] = $OclAsType_{Client}$-*Client-invalid*
$OclAsType_{Client}$-*Reservation-invalid*
$OclAsType_{Client}$-*OclAny-invalid*
$OclAsType_{Client}$-*Person-invalid*
$OclAsType_{Client}$-*Staff-invalid*
$OclAsType_{Client}$-*Flight-invalid*
$OclAsType_{Client}$-*Client-null*
$OclAsType_{Client}$-*Reservation-null*
$OclAsType_{Client}$-*OclAny-null*
$OclAsType_{Client}$-*Person-null*
$OclAsType_{Client}$-*Staff-null*
$OclAsType_{Client}$-*Flight-null*
$OclAsType_{Reservation}$-*Client-invalid*
$OclAsType_{Reservation}$-*Reservation-invalid*
$OclAsType_{Reservation}$-*OclAny-invalid*
$OclAsType_{Reservation}$-*Person-invalid*

$OclAsType_{Reservation}$-*Staff-invalid*
$OclAsType_{Reservation}$-*Flight-invalid*
$OclAsType_{Reservation}$-*Client-null*
$OclAsType_{Reservation}$-*Reservation-null*
$OclAsType_{Reservation}$-*OclAny-null*
$OclAsType_{Reservation}$-*Person-null*
$OclAsType_{Reservation}$-*Staff-null*
$OclAsType_{Reservation}$-*Flight-null*
$OclAsType_{OclAny}$-*Client-invalid*
$OclAsType_{OclAny}$-*Reservation-invalid*
$OclAsType_{OclAny}$-*OclAny-invalid*
$OclAsType_{OclAny}$-*Person-invalid*
$OclAsType_{OclAny}$-*Staff-invalid*
$OclAsType_{OclAny}$-*Flight-invalid*
$OclAsType_{OclAny}$-*Client-null*
$OclAsType_{OclAny}$-*Reservation-null*
$OclAsType_{OclAny}$-*OclAny-null*
$OclAsType_{OclAny}$-*Person-null*
$OclAsType_{OclAny}$-*Staff-null*
$OclAsType_{OclAny}$-*Flight-null*
$OclAsType_{Person}$-*Client-invalid*
$OclAsType_{Person}$-*Reservation-invalid*
$OclAsType_{Person}$-*OclAny-invalid*
$OclAsType_{Person}$-*Person-invalid*
$OclAsType_{Person}$-*Staff-invalid*
$OclAsType_{Person}$-*Flight-invalid*
$OclAsType_{Person}$-*Client-null*
$OclAsType_{Person}$-*Reservation-null*
$OclAsType_{Person}$-*OclAny-null*
$OclAsType_{Person}$-*Person-null*
$OclAsType_{Person}$-*Staff-null*
$OclAsType_{Person}$-*Flight-null*
$OclAsType_{Staff}$-*Client-invalid*
$OclAsType_{Staff}$-*Reservation-invalid*
$OclAsType_{Staff}$-*OclAny-invalid*
$OclAsType_{Staff}$-*Person-invalid*
$OclAsType_{Staff}$-*Staff-invalid*
$OclAsType_{Staff}$-*Flight-invalid*
$OclAsType_{Staff}$-*Client-null*
$OclAsType_{Staff}$-*Reservation-null*
$OclAsType_{Staff}$-*OclAny-null*
$OclAsType_{Staff}$-*Person-null*
$OclAsType_{Staff}$-*Staff-null*
$OclAsType_{Staff}$-*Flight-null*
$OclAsType_{Flight}$-*Client-invalid*
$OclAsType_{Flight}$-*Reservation-invalid*
$OclAsType_{Flight}$-*OclAny-invalid*
$OclAsType_{Flight}$-*Person-invalid*
$OclAsType_{Flight}$-*Staff-invalid*
$OclAsType_{Flight}$-*Flight-invalid*
$OclAsType_{Flight}$-*Client-null*
$OclAsType_{Flight}$-*Reservation-null*
$OclAsType_{Flight}$-*OclAny-null*
$OclAsType_{Flight}$-*Person-null*
$OclAsType_{Flight}$-*Staff-null*
$OclAsType_{Flight}$-*Flight-null*

Validity and Definedness Properties

lemma $OclAsType_{Person}$-*Client-defined* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclAsType(Person)))$
using *isdef*
by(*auto simp*: $OclAsType_{Person}$-*Client foundation16 null-option-def bot-option-def*)
lemma $OclAsType_{Person}$-*Staff-defined* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclAsType(Person)))$
using *isdef*
by(*auto simp*: $OclAsType_{Person}$-*Staff foundation16 null-option-def bot-option-def*)
lemma $OclAsType_{OclAny}$-*Flight-defined* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclAsType(OclAny)))$
using *isdef*

by(*auto simp*: $OclAsType_{OclAny}$-*Flight foundation16 null-option-def bot-option-def*)
lemma $OclAsType_{OclAny}$-*Client-defined* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclAsType(OclAny)))$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Client foundation16 null-option-def bot-option-def*)
lemma $OclAsType_{OclAny}$-*Staff-defined* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclAsType(OclAny)))$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Staff foundation16 null-option-def bot-option-def*)
lemma $OclAsType_{OclAny}$-*Person-defined* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclAsType(OclAny)))$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Person foundation16 null-option-def bot-option-def*)
lemma $OclAsType_{OclAny}$-*Reservation-defined* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclAsType(OclAny)))$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Reservation foundation16 null-option-def bot-option-def*)

Up Down Casting

lemma up_{OclAny}-$down_{Flight}$-*cast0* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (((X::\cdot Flight)\ .oclAsType(OclAny))\ .oclAsType(Flight)) \triangleq X$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Flight* $OclAsType_{Flight}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Flight}$.*split* ty_{Flight}.*split*)
lemma up_{Person}-$down_{Client}$-*cast0* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (((X::\cdot Client)\ .oclAsType(Person))\ .oclAsType(Client)) \triangleq X$
using *isdef*
by(*auto simp*: $OclAsType_{Person}$-*Client* $OclAsType_{Client}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Client}$.*split* ty_{Client}.*split*)
lemma up_{OclAny}-$down_{Client}$-*cast0* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (((X::\cdot Client)\ .oclAsType(OclAny))\ .oclAsType(Client)) \triangleq X$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Client* $OclAsType_{Client}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Client}$.*split* ty_{Client}.*split*)
lemma up_{Person}-$down_{Staff}$-*cast0* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (((X::\cdot Staff)\ .oclAsType(Person))\ .oclAsType(Staff)) \triangleq X$
using *isdef*
by(*auto simp*: $OclAsType_{Person}$-*Staff* $OclAsType_{Staff}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Staff}$.*split* ty_{Staff}.*split*)
lemma up_{OclAny}-$down_{Staff}$-*cast0* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (((X::\cdot Staff)\ .oclAsType(OclAny))\ .oclAsType(Staff)) \triangleq X$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Staff* $OclAsType_{Staff}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Staff}$.*split* ty_{Staff}.*split*)
lemma up_{OclAny}-$down_{Person}$-*cast0* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (((X::\cdot Person)\ .oclAsType(OclAny))\ .oclAsType(Person)) \triangleq X$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Person* $OclAsType_{Person}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
lemma up_{OclAny}-$down_{Reservation}$-*cast0* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (((X::Reservation)\ .oclAsType(OclAny))\ .oclAsType(Reservation)) \triangleq X$
using *isdef*
by(*auto simp*: $OclAsType_{OclAny}$-*Reservation* $OclAsType_{Reservation}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Reservation}$.*split* $ty_{Reservation}$.*split*)

lemma up_{OclAny}-$down_{Flight}$-*cast* :
shows $(((X::\cdot Flight)\ .oclAsType(OclAny))\ .oclAsType(Flight)) = X$
apply(*rule ext*, *rename-tac* τ)
apply(*rule foundation22*[*THEN iffD1*])
apply(*case-tac* $\tau \models (\delta\ (X))$, *simp add*: up_{OclAny}-$down_{Flight}$-*cast0*)

apply(*simp add*: *defined-split*, *elim disjE*)
apply((*erule StrongEq-L-subst2-rev*, *simp*, *simp*)+)
done
lemma up_{Person}-$down_{Client}$-*cast* :
shows (((*X*::·*Client*) .*oclAsType*(*Person*)) .*oclAsType*(*Client*)) = *X*
apply(*rule ext*, *rename-tac* τ)
apply(*rule foundation22*[*THEN iffD1*])
apply(*case-tac* τ ⊨ (δ (*X*)), *simp add*: up_{Person}-$down_{Client}$-*cast0*)
apply(*simp add*: *defined-split*, *elim disjE*)
apply((*erule StrongEq-L-subst2-rev*, *simp*, *simp*)+)
done
lemma up_{OclAny}-$down_{Client}$-*cast* :
shows (((*X*::·*Client*) .*oclAsType*(*OclAny*)) .*oclAsType*(*Client*)) = *X*
apply(*rule ext*, *rename-tac* τ)
apply(*rule foundation22*[*THEN iffD1*])
apply(*case-tac* τ ⊨ (δ (*X*)), *simp add*: up_{OclAny}-$down_{Client}$-*cast0*)
apply(*simp add*: *defined-split*, *elim disjE*)
apply((*erule StrongEq-L-subst2-rev*, *simp*, *simp*)+)
done
lemma up_{Person}-$down_{Staff}$-*cast* :
shows (((*X*::·*Staff*) .*oclAsType*(*Person*)) .*oclAsType*(*Staff*)) = *X*
apply(*rule ext*, *rename-tac* τ)
apply(*rule foundation22*[*THEN iffD1*])
apply(*case-tac* τ ⊨ (δ (*X*)), *simp add*: up_{Person}-$down_{Staff}$-*cast0*)
apply(*simp add*: *defined-split*, *elim disjE*)
apply((*erule StrongEq-L-subst2-rev*, *simp*, *simp*)+)
done
lemma up_{OclAny}-$down_{Staff}$-*cast* :
shows (((*X*::·*Staff*) .*oclAsType*(*OclAny*)) .*oclAsType*(*Staff*)) = *X*
apply(*rule ext*, *rename-tac* τ)
apply(*rule foundation22*[*THEN iffD1*])
apply(*case-tac* τ ⊨ (δ (*X*)), *simp add*: up_{OclAny}-$down_{Staff}$-*cast0*)
apply(*simp add*: *defined-split*, *elim disjE*)
apply((*erule StrongEq-L-subst2-rev*, *simp*, *simp*)+)
done
lemma up_{OclAny}-$down_{Person}$-*cast* :
shows (((*X*::·*Person*) .*oclAsType*(*OclAny*)) .*oclAsType*(*Person*)) = *X*
apply(*rule ext*, *rename-tac* τ)
apply(*rule foundation22*[*THEN iffD1*])
apply(*case-tac* τ ⊨ (δ (*X*)), *simp add*: up_{OclAny}-$down_{Person}$-*cast0*)
apply(*simp add*: *defined-split*, *elim disjE*)
apply((*erule StrongEq-L-subst2-rev*, *simp*, *simp*)+)
done
lemma up_{OclAny}-$down_{Reservation}$-*cast* :
shows (((*X*::·*Reservation*) .*oclAsType*(*OclAny*)) .*oclAsType*(*Reservation*)) = *X*
apply(*rule ext*, *rename-tac* τ)
apply(*rule foundation22*[*THEN iffD1*])
apply(*case-tac* τ ⊨ (δ (*X*)), *simp add*: up_{OclAny}-$down_{Reservation}$-*cast0*)
apply(*simp add*: *defined-split*, *elim disjE*)
apply((*erule StrongEq-L-subst2-rev*, *simp*, *simp*)+)
done

lemma $down_{Flight}$-up_{OclAny}-*cast* :
assumes *def-X*: *X* = ((*Y*::·*Flight*) .*oclAsType*(*OclAny*))
shows (τ ⊨ ((*not* ((υ (*X*)))) *or* ((*X* .*oclAsType*(*Flight*)) .*oclAsType*(*OclAny*)) ≐ *X*))
apply(*case-tac* (τ ⊨ ((*not* ((υ (*X*)))))), *rule foundation25*, *simp*)
by(*rule foundation25′*, *simp add*: *def-X* up_{OclAny}-$down_{Flight}$-*cast StrictRefEq*$_{Object}$-*sym*)
lemma $down_{Client}$-up_{Person}-*cast* :
assumes *def-X*: *X* = ((*Y*::·*Client*) .*oclAsType*(*Person*))
shows (τ ⊨ ((*not* ((υ (*X*)))) *or* ((*X* .*oclAsType*(*Client*)) .*oclAsType*(*Person*)) ≐ *X*))
apply(*case-tac* (τ ⊨ ((*not* ((υ (*X*)))))), *rule foundation25*, *simp*)
by(*rule foundation25′*, *simp add*: *def-X* up_{Person}-$down_{Client}$-*cast StrictRefEq*$_{Object}$-*sym*)
lemma $down_{Client}$-up_{OclAny}-*cast* :
assumes *def-X*: *X* = ((*Y*::·*Client*) .*oclAsType*(*OclAny*))
shows (τ ⊨ ((*not* ((υ (*X*)))) *or* ((*X* .*oclAsType*(*Client*)) .*oclAsType*(*OclAny*)) ≐ *X*))
apply(*case-tac* (τ ⊨ ((*not* ((υ (*X*)))))), *rule foundation25*, *simp*)
by(*rule foundation25′*, *simp add*: *def-X* up_{OclAny}-$down_{Client}$-*cast StrictRefEq*$_{Object}$-*sym*)
lemma $down_{Staff}$-up_{Person}-*cast* :
assumes *def-X*: *X* = ((*Y*::·*Staff*) .*oclAsType*(*Person*))
shows (τ ⊨ ((*not* ((υ (*X*)))) *or* ((*X* .*oclAsType*(*Staff*)) .*oclAsType*(*Person*)) ≐ *X*))
apply(*case-tac* (τ ⊨ ((*not* ((υ (*X*)))))), *rule foundation25*, *simp*)
by(*rule foundation25′*, *simp add*: *def-X* up_{Person}-$down_{Staff}$-*cast StrictRefEq*$_{Object}$-*sym*)

lemma $down_{Staff}$-up_{OclAny}-cast :
assumes def-X: $X = ((Y::Staff)\ .oclAsType(OclAny))$
shows $(\tau \models ((not\ ((v\ (X))))\ or\ ((X\ .oclAsType(Staff))\ .oclAsType(OclAny)) \doteq X))$
apply(case-tac $(\tau \models ((not\ ((v\ (X))))))$, rule foundation25, simp)
by(rule foundation25', simp add: def-X up_{OclAny}-$down_{Staff}$-cast $StrictRefEq_{Object}$-sym)
lemma $down_{Person}$-up_{OclAny}-cast :
assumes def-X: $X = ((Y::Person)\ .oclAsType(OclAny))$
shows $(\tau \models ((not\ ((v\ (X))))\ or\ ((X\ .oclAsType(Person))\ .oclAsType(OclAny)) \doteq X))$
apply(case-tac $(\tau \models ((not\ ((v\ (X))))))$, rule foundation25, simp)
by(rule foundation25', simp add: def-X up_{OclAny}-$down_{Person}$-cast $StrictRefEq_{Object}$-sym)
lemma $down_{Reservation}$-up_{OclAny}-cast :
assumes def-X: $X = ((Y::Reservation)\ .oclAsType(OclAny))$
shows $(\tau \models ((not\ ((v\ (X))))\ or\ ((X\ .oclAsType(Reservation))\ .oclAsType(OclAny)) \doteq X))$
apply(case-tac $(\tau \models ((not\ ((v\ (X))))))$, rule foundation25, simp)
by(rule foundation25', simp add: def-X up_{OclAny}-$down_{Reservation}$-cast $StrictRefEq_{Object}$-sym)

Const

lemma $OclAsType_{Client}$-Client-const : $(const\ ((X::Client))) \Longrightarrow (const\ (X\ .oclAsType(Client)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Client}$-Client prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Client}$-Reservation-const : $(const\ ((X::Reservation))) \Longrightarrow (const\ (X\ .oclAsType(Client)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Client}$-Reservation prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Client}$-OclAny-const : $(const\ ((X::OclAny))) \Longrightarrow (const\ (X\ .oclAsType(Client)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Client}$-OclAny prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Client}$-Person-const : $(const\ ((X::Person))) \Longrightarrow (const\ (X\ .oclAsType(Client)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Client}$-Person prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Client}$-Staff-const : $(const\ ((X::Staff))) \Longrightarrow (const\ (X\ .oclAsType(Client)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Client}$-Staff prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Client}$-Flight-const : $(const\ ((X::Flight))) \Longrightarrow (const\ (X\ .oclAsType(Client)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Client}$-Flight prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Reservation}$-Client-const : $(const\ ((X::Client))) \Longrightarrow (const\ (X\ .oclAsType(Reservation)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Reservation}$-Client prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Reservation}$-Reservation-const : $(const\ ((X::Reservation))) \Longrightarrow (const\ (X\ .oclAsType(Reservation)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Reservation}$-Reservation prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Reservation}$-OclAny-const : $(const\ ((X::OclAny))) \Longrightarrow (const\ (X\ .oclAsType(Reservation)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Reservation}$-OclAny prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Reservation}$-Person-const : $(const\ ((X::Person))) \Longrightarrow (const\ (X\ .oclAsType(Reservation)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Reservation}$-Person prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Reservation}$-Staff-const : $(const\ ((X::Staff))) \Longrightarrow (const\ (X\ .oclAsType(Reservation)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Reservation}$-Staff prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Reservation}$-Flight-const : $(const\ ((X::Flight))) \Longrightarrow (const\ (X\ .oclAsType(Reservation)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{Reservation}$-Flight prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{OclAny}$-Client-const : $(const\ ((X::Client))) \Longrightarrow (const\ (X\ .oclAsType(OclAny)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{OclAny}$-Client prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{OclAny}$-Reservation-const : $(const\ ((X::Reservation))) \Longrightarrow (const\ (X\ .oclAsType(OclAny)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{OclAny}$-Reservation prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{OclAny}$-OclAny-const : $(const\ ((X::OclAny))) \Longrightarrow (const\ (X\ .oclAsType(OclAny)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{OclAny}$-OclAny prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{OclAny}$-Person-const : $(const\ ((X::Person))) \Longrightarrow (const\ (X\ .oclAsType(OclAny)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{OclAny}$-Person prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{OclAny}$-Staff-const : $(const\ ((X::Staff))) \Longrightarrow (const\ (X\ .oclAsType(OclAny)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{OclAny}$-Staff prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{OclAny}$-Flight-const : $(const\ ((X::Flight))) \Longrightarrow (const\ (X\ .oclAsType(OclAny)))$
by(simp add: const-def, (metis (no-types) $OclAsType_{OclAny}$-Flight prod.collapse bot-option-def invalid-def null-fun-def null-option-def)?)
lemma $OclAsType_{Person}$-Client-const : $(const\ ((X::Client))) \Longrightarrow (const\ (X\ .oclAsType(Person)))$

by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Person}$-*Client prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Person}$-*Reservation-const* : (*const* ((*X*::·*Reservation*))) ⟹ (*const* (*X* .*oclAsType*(*Person*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Person}$-*Reservation prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Person}$-*OclAny-const* : (*const* ((*X*::·*OclAny*))) ⟹ (*const* (*X* .*oclAsType*(*Person*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Person}$-*OclAny prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Person}$-*Person-const* : (*const* ((*X*::·*Person*))) ⟹ (*const* (*X* .*oclAsType*(*Person*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Person}$-*Person prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Person}$-*Staff-const* : (*const* ((*X*::*Staff*))) ⟹ (*const* (*X* .*oclAsType*(*Person*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Person}$-*Staff prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Person}$-*Flight-const* : (*const* ((*X*::·*Flight*))) ⟹ (*const* (*X* .*oclAsType*(*Person*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Person}$-*Flight prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Staff}$-*Client-const* : (*const* ((*X*::·*Client*))) ⟹ (*const* (*X* .*oclAsType*(*Staff*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Staff}$-*Client prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Staff}$-*Reservation-const* : (*const* ((*X*::·*Reservation*))) ⟹ (*const* (*X* .*oclAsType*(*Staff*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Staff}$-*Reservation prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Staff}$-*OclAny-const* : (*const* ((*X*::·*OclAny*))) ⟹ (*const* (*X* .*oclAsType*(*Staff*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Staff}$-*OclAny prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Staff}$-*Person-const* : (*const* ((*X*::·*Person*))) ⟹ (*const* (*X* .*oclAsType*(*Staff*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Staff}$-*Person prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Staff}$-*Staff-const* : (*const* ((*X*::·*Staff*))) ⟹ (*const* (*X* .*oclAsType*(*Staff*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Staff}$-*Staff prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Staff}$-*Flight-const* : (*const* ((*X*::·*Flight*))) ⟹ (*const* (*X* .*oclAsType*(*Staff*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Staff}$-*Flight prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Flight}$-*Client-const* : (*const* ((*X*::·*Client*))) ⟹ (*const* (*X* .*oclAsType*(*Flight*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Flight}$-*Client prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Flight}$-*Reservation-const* : (*const* ((*X*::·*Reservation*))) ⟹ (*const* (*X* .*oclAsType*(*Flight*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Flight}$-*Reservation prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Flight}$-*OclAny-const* : (*const* ((*X*::·*OclAny*))) ⟹ (*const* (*X* .*oclAsType*(*Flight*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Flight}$-*OclAny prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Flight}$-*Person-const* : (*const* ((*X*::·*Person*))) ⟹ (*const* (*X* .*oclAsType*(*Flight*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Flight}$-*Person prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Flight}$-*Staff-const* : (*const* ((*X*::·*Staff*))) ⟹ (*const* (*X* .*oclAsType*(*Flight*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Flight}$-*Staff prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)
lemma $OclAsType_{Flight}$-*Flight-const* : (*const* ((*X*::·*Flight*))) ⟹ (*const* (*X* .*oclAsType*(*Flight*)))
by(*simp add*: *const-def*, (*metis* (*no-types*) $OclAsType_{Flight}$-*Flight prod.collapse bot-option-def invalid-def null-fun-def null-option-def*)?)

lemmas[*simp,code-unfold*] = $OclAsType_{Client}$-*Client-const*
$OclAsType_{Client}$-*Reservation-const*
$OclAsType_{Client}$-*OclAny-const*
$OclAsType_{Client}$-*Person-const*
$OclAsType_{Client}$-*Staff-const*
$OclAsType_{Client}$-*Flight-const*
$OclAsType_{Reservation}$-*Client-const*
$OclAsType_{Reservation}$-*Reservation-const*
$OclAsType_{Reservation}$-*OclAny-const*
$OclAsType_{Reservation}$-*Person-const*
$OclAsType_{Reservation}$-*Staff-const*
$OclAsType_{Reservation}$-*Flight-const*
$OclAsType_{OclAny}$-*Client-const*
$OclAsType_{OclAny}$-*Reservation-const*
$OclAsType_{OclAny}$-*OclAny-const*
$OclAsType_{OclAny}$-*Person-const*
$OclAsType_{OclAny}$-*Staff-const*
$OclAsType_{OclAny}$-*Flight-const*

$OclAsType_{Person}$-$Client$-$const$
$OclAsType_{Person}$-$Reservation$-$const$
$OclAsType_{Person}$-$OclAny$-$const$
$OclAsType_{Person}$-$Person$-$const$
$OclAsType_{Person}$-$Staff$-$const$
$OclAsType_{Person}$-$Flight$-$const$
$OclAsType_{Staff}$-$Client$-$const$
$OclAsType_{Staff}$-$Reservation$-$const$
$OclAsType_{Staff}$-$OclAny$-$const$
$OclAsType_{Staff}$-$Person$-$const$
$OclAsType_{Staff}$-$Staff$-$const$
$OclAsType_{Staff}$-$Flight$-$const$
$OclAsType_{Flight}$-$Client$-$const$
$OclAsType_{Flight}$-$Reservation$-$const$
$OclAsType_{Flight}$-$OclAny$-$const$
$OclAsType_{Flight}$-$Person$-$const$
$OclAsType_{Flight}$-$Staff$-$const$
$OclAsType_{Flight}$-$Flight$-$const$

B.5 Class Model: OclIsTypeOf

Definition

consts $OclIsTypeOf_{Flight} :: 'α \Rightarrow Boolean$ ((-) $.oclIsTypeOf'(Flight')$)
consts $OclIsTypeOf_{Client} :: 'α \Rightarrow Boolean$ ((-) $.oclIsTypeOf'(Client')$)
consts $OclIsTypeOf_{Staff} :: 'α \Rightarrow Boolean$ ((-) $.oclIsTypeOf'(Staff')$)
consts $OclIsTypeOf_{Person} :: 'α \Rightarrow Boolean$ ((-) $.oclIsTypeOf'(Person')$)
consts $OclIsTypeOf_{Reservation} :: 'α \Rightarrow Boolean$ ((-) $.oclIsTypeOf'(Reservation')$)
consts $OclIsTypeOf_{OclAny} :: 'α \Rightarrow Boolean$ ((-) $.oclIsTypeOf'(OclAny')$)

overloading $OclIsTypeOf_{Flight} \equiv (OclIsTypeOf_{Flight}::(\cdot Flight) \Rightarrow -)$
begin
  definition $OclIsTypeOf_{Flight}$-$Flight : (x::\cdot Flight)\ .oclIsTypeOf(Flight) \equiv (\lambda\tau.\ (case\ (x\ (\tau))\ of\ \bot \Rightarrow (invalid\ (\tau))$
    | $\lfloor\bot\rfloor \Rightarrow (true\ (\tau))$
    | $\lfloor\lfloor(mk_{Flight}\ ((mk\mathcal{EXT}_{Flight}\ (-)))\ (-)\ (-)\ (-)\ (-))\rfloor\rfloor \Rightarrow (true\ (\tau))))$
end
overloading $OclIsTypeOf_{Flight} \equiv (OclIsTypeOf_{Flight}::(\cdot OclAny) \Rightarrow -)$
begin
  definition $OclIsTypeOf_{Flight}$-$OclAny : (x::\cdot OclAny)\ .oclIsTypeOf(Flight) \equiv (\lambda\tau.\ (case\ (x\ (\tau))\ of\ \bot \Rightarrow (invalid\ (\tau))$
    | $\lfloor\bot\rfloor \Rightarrow (true\ (\tau))$
    | $\lfloor\lfloor(mk_{OclAny}\ ((mk\mathcal{EXT}_{OclAny\text{-}Flight}\ (-))))\rfloor\rfloor \Rightarrow (true\ (\tau))$
    | - $\Rightarrow (false\ (\tau))))$
end
overloading $OclIsTypeOf_{Flight} \equiv (OclIsTypeOf_{Flight}::(\cdot Staff) \Rightarrow -)$
begin
  definition $OclIsTypeOf_{Flight}$-$Staff : (x::\cdot Staff)\ .oclIsTypeOf(Flight) \equiv (\lambda\tau.\ (case\ (x\ (\tau))\ of\ \bot \Rightarrow (invalid\ (\tau))$
    | $\lfloor\bot\rfloor \Rightarrow (true\ (\tau))$
    | - $\Rightarrow (false\ (\tau))))$
end
overloading $OclIsTypeOf_{Flight} \equiv (OclIsTypeOf_{Flight}::(\cdot Person) \Rightarrow -)$
begin
  definition $OclIsTypeOf_{Flight}$-$Person : (x::\cdot Person)\ .oclIsTypeOf(Flight) \equiv (\lambda\tau.\ (case\ (x\ (\tau))\ of\ \bot \Rightarrow (invalid\ (\tau))$
    | $\lfloor\bot\rfloor \Rightarrow (true\ (\tau))$
    | - $\Rightarrow (false\ (\tau))))$
end
overloading $OclIsTypeOf_{Flight} \equiv (OclIsTypeOf_{Flight}::(\cdot Client) \Rightarrow -)$
begin
  definition $OclIsTypeOf_{Flight}$-$Client : (x::\cdot Client)\ .oclIsTypeOf(Flight) \equiv (\lambda\tau.\ (case\ (x\ (\tau))\ of\ \bot \Rightarrow (invalid\ (\tau))$
    | $\lfloor\bot\rfloor \Rightarrow (true\ (\tau))$
    | - $\Rightarrow (false\ (\tau))))$
end
overloading $OclIsTypeOf_{Flight} \equiv (OclIsTypeOf_{Flight}::(\cdot Reservation) \Rightarrow -)$
begin
  definition $OclIsTypeOf_{Flight}$-$Reservation : (x::\cdot Reservation)\ .oclIsTypeOf(Flight) \equiv (\lambda\tau.\ (case\ (x\ (\tau))\ of\ \bot \Rightarrow (invalid\ (\tau))$
    | $\lfloor\bot\rfloor \Rightarrow (true\ (\tau))$
    | - $\Rightarrow (false\ (\tau))))$
end
overloading $OclIsTypeOf_{Client} \equiv (OclIsTypeOf_{Client}::(\cdot Client) \Rightarrow -)$
begin
  definition $OclIsTypeOf_{Client}$-$Client : (x::\cdot Client)\ .oclIsTypeOf(Client) \equiv (\lambda\tau.\ (case\ (x\ (\tau))\ of\ \bot \Rightarrow (invalid\ (\tau))$
    | $\lfloor\bot\rfloor \Rightarrow (true\ (\tau))$

```
  | ⌊⌊(mk_Client ((mkℰ𝒳𝒯_Client (-) (-))) (-))⌋⌋ ⇒ (true (τ))))
end
overloading OclIsTypeOf_Client ≡ (OclIsTypeOf_Client::(·Person) ⇒ -)
begin
  definition OclIsTypeOf_Client-Person : (x::·Person) .oclIsTypeOf(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_Person ((mkℰ𝒳𝒯_Person-Client (-))) (-))⌋⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Client ≡ (OclIsTypeOf_Client::(·OclAny) ⇒ -)
begin
  definition OclIsTypeOf_Client-OclAny : (x::·OclAny) .oclIsTypeOf(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Client (-))))⌋⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Client ≡ (OclIsTypeOf_Client::(·Staff) ⇒ -)
begin
  definition OclIsTypeOf_Client-Staff : (x::·Staff) .oclIsTypeOf(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Client ≡ (OclIsTypeOf_Client::(·Reservation) ⇒ -)
begin
  definition OclIsTypeOf_Client-Reservation : (x::·Reservation) .oclIsTypeOf(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Client ≡ (OclIsTypeOf_Client::(·Flight) ⇒ -)
begin
  definition OclIsTypeOf_Client-Flight : (x::·Flight) .oclIsTypeOf(Client) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Staff ≡ (OclIsTypeOf_Staff::(·Staff) ⇒ -)
begin
  definition OclIsTypeOf_Staff-Staff : (x::·Staff) .oclIsTypeOf(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_Staff ((mkℰ𝒳𝒯_Staff (-) (-))))⌋⌋ ⇒ (true (τ))))
end
overloading OclIsTypeOf_Staff ≡ (OclIsTypeOf_Staff::(·Person) ⇒ -)
begin
  definition OclIsTypeOf_Staff-Person : (x::·Person) .oclIsTypeOf(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_Person ((mkℰ𝒳𝒯_Person-Staff (-))) (-))⌋⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Staff ≡ (OclIsTypeOf_Staff::(·OclAny) ⇒ -)
begin
  definition OclIsTypeOf_Staff-OclAny : (x::·OclAny) .oclIsTypeOf(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Staff (-))))⌋⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Staff ≡ (OclIsTypeOf_Staff::(·Client) ⇒ -)
begin
  definition OclIsTypeOf_Staff-Client : (x::·Client) .oclIsTypeOf(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Staff ≡ (OclIsTypeOf_Staff::(·Reservation) ⇒ -)
begin
  definition OclIsTypeOf_Staff-Reservation : (x::·Reservation) .oclIsTypeOf(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Staff ≡ (OclIsTypeOf_Staff::(·Flight) ⇒ -)
begin
  definition OclIsTypeOf_Staff-Flight : (x::·Flight) .oclIsTypeOf(Staff) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Person ≡ (OclIsTypeOf_Person::(·Person) ⇒ -)
```

```
begin
  definition OclIsTypeOf_Person-Person : (x::·Person) .oclIsTypeOf(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_Person ((mkℰ𝒳𝒯_Person (-))) (-))⌋⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Person ≡ (OclIsTypeOf_Person::(·OclAny) ⇒ -)
begin
  definition OclIsTypeOf_Person-OclAny : (x::·OclAny) .oclIsTypeOf(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Person (-))))⌋⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Person ≡ (OclIsTypeOf_Person::(·Client) ⇒ -)
begin
  definition OclIsTypeOf_Person-Client : (x::·Client) .oclIsTypeOf(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Person ≡ (OclIsTypeOf_Person::(·Staff) ⇒ -)
begin
  definition OclIsTypeOf_Person-Staff : (x::·Staff) .oclIsTypeOf(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Person ≡ (OclIsTypeOf_Person::(·Reservation) ⇒ -)
begin
  definition OclIsTypeOf_Person-Reservation : (x::·Reservation) .oclIsTypeOf(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Person ≡ (OclIsTypeOf_Person::(·Flight) ⇒ -)
begin
  definition OclIsTypeOf_Person-Flight : (x::·Flight) .oclIsTypeOf(Person) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Reservation ≡ (OclIsTypeOf_Reservation::(·Reservation) ⇒ -)
begin
  definition OclIsTypeOf_Reservation-Reservation : (x::·Reservation) .oclIsTypeOf(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒
(invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_Reservation ((mkℰ𝒳𝒯_Reservation (-))) (-) (-) (-))⌋⌋ ⇒ (true (τ))))
end
overloading OclIsTypeOf_Reservation ≡ (OclIsTypeOf_Reservation::(·OclAny) ⇒ -)
begin
  definition OclIsTypeOf_Reservation-OclAny : (x::·OclAny) .oclIsTypeOf(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid
(τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny-Reservation (-))))⌋⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Reservation ≡ (OclIsTypeOf_Reservation::(·Staff) ⇒ -)
begin
  definition OclIsTypeOf_Reservation-Staff : (x::·Staff) .oclIsTypeOf(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Reservation ≡ (OclIsTypeOf_Reservation::(·Person) ⇒ -)
begin
  definition OclIsTypeOf_Reservation-Person : (x::·Person) .oclIsTypeOf(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Reservation ≡ (OclIsTypeOf_Reservation::(·Client) ⇒ -)
begin
  definition OclIsTypeOf_Reservation-Client : (x::·Client) .oclIsTypeOf(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
    | ⌊⊥⌋ ⇒ (true (τ))
    | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_Reservation ≡ (OclIsTypeOf_Reservation::(·Flight) ⇒ -)
begin
  definition OclIsTypeOf_Reservation-Flight : (x::·Flight) .oclIsTypeOf(Reservation) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
```

```
  | ⌊⊥⌋ ⇒ (true (τ))
  | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_OclAny ≡ (OclIsTypeOf_OclAny::(·OclAny) ⇒ -)
begin
  definition OclIsTypeOf_OclAny-OclAny : (x::·OclAny) .oclIsTypeOf(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
  | ⌊⊥⌋ ⇒ (true (τ))
  | ⌊⌊(mk_OclAny ((mkℰ𝒳𝒯_OclAny (-))))⌋⌋ ⇒ (true (τ))
  | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_OclAny ≡ (OclIsTypeOf_OclAny::(·Flight) ⇒ -)
begin
  definition OclIsTypeOf_OclAny-Flight : (x::·Flight) .oclIsTypeOf(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
  | ⌊⊥⌋ ⇒ (true (τ))
  | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_OclAny ≡ (OclIsTypeOf_OclAny::(·Client) ⇒ -)
begin
  definition OclIsTypeOf_OclAny-Client : (x::·Client) .oclIsTypeOf(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
  | ⌊⊥⌋ ⇒ (true (τ))
  | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_OclAny ≡ (OclIsTypeOf_OclAny::(·Staff) ⇒ -)
begin
  definition OclIsTypeOf_OclAny-Staff : (x::·Staff) .oclIsTypeOf(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
  | ⌊⊥⌋ ⇒ (true (τ))
  | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_OclAny ≡ (OclIsTypeOf_OclAny::(·Person) ⇒ -)
begin
  definition OclIsTypeOf_OclAny-Person : (x::·Person) .oclIsTypeOf(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid (τ))
  | ⌊⊥⌋ ⇒ (true (τ))
  | - ⇒ (false (τ))))
end
overloading OclIsTypeOf_OclAny ≡ (OclIsTypeOf_OclAny::(·Reservation) ⇒ -)
begin
  definition OclIsTypeOf_OclAny-Reservation : (x::·Reservation) .oclIsTypeOf(OclAny) ≡ (λτ. (case (x (τ)) of ⊥ ⇒ (invalid
(τ))
  | ⌊⊥⌋ ⇒ (true (τ))
  | - ⇒ (false (τ))))
end


definition OclIsTypeOf_Flight-𝔄 = (λ (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsTypeOf(Flight))
  | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsTypeOf(Flight))
  | (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsTypeOf(Flight))
  | (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsTypeOf(Flight))
  | (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsTypeOf(Flight))
  | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsTypeOf(Flight)))
definition OclIsTypeOf_Client-𝔄 = (λ (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsTypeOf(Client))
  | (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsTypeOf(Client))
  | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsTypeOf(Client))
  | (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsTypeOf(Client))
  | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsTypeOf(Client))
  | (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsTypeOf(Client)))
definition OclIsTypeOf_Staff-𝔄 = (λ (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsTypeOf(Staff))
  | (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsTypeOf(Staff))
  | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsTypeOf(Staff))
  | (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsTypeOf(Staff))
  | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsTypeOf(Staff))
  | (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsTypeOf(Staff)))
definition OclIsTypeOf_Person-𝔄 = (λ (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsTypeOf(Person))
  | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsTypeOf(Person))
  | (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsTypeOf(Person))
  | (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsTypeOf(Person))
  | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsTypeOf(Person))
  | (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsTypeOf(Person)))
definition OclIsTypeOf_Reservation-𝔄 = (λ (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation)
.oclIsTypeOf(Reservation))
  | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsTypeOf(Reservation))
  | (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsTypeOf(Reservation))
  | (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsTypeOf(Reservation))
  | (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsTypeOf(Reservation))
```

| $(in_{Flight}\ (Flight)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x\rfloor\rfloor))\ (Flight))::\cdot Flight)\ .oclIsTypeOf(Reservation)))$
definition $OclIsTypeOf_{OclAny}\text{-}\mathfrak{A} = (\lambda\ (in_{OclAny}\ (OclAny)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x\rfloor\rfloor))\ (OclAny))::\cdot OclAny)\ .oclIsTypeOf(OclAny))$
| $(in_{Flight}\ (Flight)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x\rfloor\rfloor))\ (Flight))::\cdot Flight)\ .oclIsTypeOf(OclAny))$
| $(in_{Client}\ (Client)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x\rfloor\rfloor))\ (Client))::\cdot Client)\ .oclIsTypeOf(OclAny))$
| $(in_{Staff}\ (Staff)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x\rfloor\rfloor))\ (Staff))::\cdot Staff)\ .oclIsTypeOf(OclAny))$
| $(in_{Person}\ (Person)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x\rfloor\rfloor))\ (Person))::\cdot Person)\ .oclIsTypeOf(OclAny))$
| $(in_{Reservation}\ (Reservation)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x\rfloor\rfloor))\ (Reservation))::\cdot Reservation)\ .oclIsTypeOf(OclAny)))$

lemmas[*simp,code-unfold*] = $OclIsTypeOf_{Flight}$-*Flight*
$OclIsTypeOf_{Client}$-*Client*
$OclIsTypeOf_{Staff}$-*Staff*
$OclIsTypeOf_{Person}$-*Person*
$OclIsTypeOf_{Reservation}$-*Reservation*
$OclIsTypeOf_{OclAny}$-*OclAny*

Context Passing

lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Client)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Client)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Reservation*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Reservation*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Reservation*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Reservation*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Reservation*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Reservation*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*OclAny*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*OclAny*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*OclAny*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*OclAny*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*OclAny*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*OclAny*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Person*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Person*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Person*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Person*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Person*)
lemma $cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclIsTypeOf(Client)))))$

by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Client-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Staff)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Reservation-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Client}$-*OclAny-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Person-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Staff-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Flight-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Client-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Reservation-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Client}$-*OclAny-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Person-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Staff-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Client}$-*Flight-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclIsTypeOf(Client)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Client}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Client-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Reservation-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*OclAny-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Person-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Staff-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Client)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Flight-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Client)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Client-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Reservation-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*OclAny-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Person-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Staff-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Flight-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Client-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Reservation-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*OclAny-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Reservation}$-*Person-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}$-*OclAny*)

lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}OclAny$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}OclAny$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Person$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Person$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Person$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Person$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Person$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Person$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Staff)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Flight$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Flight$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Flight$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Flight$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Flight$)
lemma $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclIsTypeOf(Reservation)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Reservation}\text{-}Flight$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclIsTypeOf(OclAny)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{OclAny}\text{-}Client$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclIsTypeOf(OclAny)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{OclAny}\text{-}Client$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}OclAny\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclIsTypeOf(OclAny)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{OclAny}\text{-}Client$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}Person\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclIsTypeOf(OclAny)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{OclAny}\text{-}Client$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}Staff\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Client)\ .oclIsTypeOf(OclAny)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{OclAny}\text{-}Client$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}Flight\text{-}Client : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Client)\ .oclIsTypeOf(OclAny)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{OclAny}\text{-}Client$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclIsTypeOf(OclAny)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{OclAny}\text{-}Reservation$)
lemma $cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)$

.oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Reservation)
lemma cp-OclIsTypeOf$_{OclAny}$-OclAny-Reservation : (cp (p)) ⟹ (cp ((λx. (((p ((x::·OclAny)))::·Reservation) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Reservation)
lemma cp-OclIsTypeOf$_{OclAny}$-Person-Reservation : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Person)))::·Reservation) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Reservation)
lemma cp-OclIsTypeOf$_{OclAny}$-Staff-Reservation : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Staff)))::·Reservation) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Reservation)
lemma cp-OclIsTypeOf$_{OclAny}$-Flight-Reservation : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Flight)))::·Reservation) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Reservation)
lemma cp-OclIsTypeOf$_{OclAny}$-Client-OclAny : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Client)))::·OclAny) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp)
lemma cp-OclIsTypeOf$_{OclAny}$-Reservation-OclAny : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Reservation)))::·OclAny) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp)
lemma cp-OclIsTypeOf$_{OclAny}$-OclAny-OclAny : (cp (p)) ⟹ (cp ((λx. (((p ((x::·OclAny)))::·OclAny) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp)
lemma cp-OclIsTypeOf$_{OclAny}$-Person-OclAny : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Person)))::·OclAny) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp)
lemma cp-OclIsTypeOf$_{OclAny}$-Staff-OclAny : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Staff)))::·OclAny) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp)
lemma cp-OclIsTypeOf$_{OclAny}$-Flight-OclAny : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Flight)))::·OclAny) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp)
lemma cp-OclIsTypeOf$_{OclAny}$-Client-Person : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Client)))::·Person) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Person)
lemma cp-OclIsTypeOf$_{OclAny}$-Reservation-Person : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Reservation)))::·Person) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Person)
lemma cp-OclIsTypeOf$_{OclAny}$-OclAny-Person : (cp (p)) ⟹ (cp ((λx. (((p ((x::·OclAny)))::·Person) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Person)
lemma cp-OclIsTypeOf$_{OclAny}$-Person-Person : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Person)))::·Person) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Person)
lemma cp-OclIsTypeOf$_{OclAny}$-Staff-Person : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Staff)))::·Person) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Person)
lemma cp-OclIsTypeOf$_{OclAny}$-Flight-Person : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Flight)))::·Person) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Person)
lemma cp-OclIsTypeOf$_{OclAny}$-Client-Staff : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Client)))::·Staff) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Staff)
lemma cp-OclIsTypeOf$_{OclAny}$-Reservation-Staff : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Reservation)))::·Staff) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Staff)
lemma cp-OclIsTypeOf$_{OclAny}$-OclAny-Staff : (cp (p)) ⟹ (cp ((λx. (((p ((x::·OclAny)))::·Staff) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Staff)
lemma cp-OclIsTypeOf$_{OclAny}$-Person-Staff : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Person)))::·Staff) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Staff)
lemma cp-OclIsTypeOf$_{OclAny}$-Staff-Staff : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Staff)))::·Staff) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Staff)
lemma cp-OclIsTypeOf$_{OclAny}$-Flight-Staff : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Flight)))::·Staff) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Staff)
lemma cp-OclIsTypeOf$_{OclAny}$-Client-Flight : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Client)))::·Flight) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Flight)
lemma cp-OclIsTypeOf$_{OclAny}$-Reservation-Flight : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Reservation)))::·Flight) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Flight)
lemma cp-OclIsTypeOf$_{OclAny}$-OclAny-Flight : (cp (p)) ⟹ (cp ((λx. (((p ((x::·OclAny)))::·Flight) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Flight)
lemma cp-OclIsTypeOf$_{OclAny}$-Person-Flight : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Person)))::·Flight) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Flight)
lemma cp-OclIsTypeOf$_{OclAny}$-Staff-Flight : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Staff)))::·Flight) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Flight)
lemma cp-OclIsTypeOf$_{OclAny}$-Flight-Flight : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Flight)))::·Flight) .oclIsTypeOf(OclAny)))))
by(rule cpI1, simp add: OclIsTypeOf$_{OclAny}$-Flight)
lemma cp-OclIsTypeOf$_{Person}$-Client-Client : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Client)))::·Client) .oclIsTypeOf(Person)))))
by(rule cpI1, simp add: OclIsTypeOf$_{Person}$-Client)
lemma cp-OclIsTypeOf$_{Person}$-Reservation-Client : (cp (p)) ⟹ (cp ((λx. (((p ((x::·Reservation)))::·Client) .oclIsTypeOf(Person)))))
by(rule cpI1, simp add: OclIsTypeOf$_{Person}$-Client)
lemma cp-OclIsTypeOf$_{Person}$-OclAny-Client : (cp (p)) ⟹ (cp ((λx. (((p ((x::·OclAny)))::·Client) .oclIsTypeOf(Person)))))
by(rule cpI1, simp add: OclIsTypeOf$_{Person}$-Client)

lemma *cp-OclIsTypeOf*$_{Person}$*-Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Client*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Client*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Client*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Reservation*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Reservation*)
lemma *cp-OclIsTypeOf*$_{Person}$*-OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Reservation*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Reservation*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Reservation*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Reservation*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Client-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*OclAny*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-OclAny*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Reservation-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*OclAny*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-OclAny*)
lemma *cp-OclIsTypeOf*$_{Person}$*-OclAny-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*OclAny*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-OclAny*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Person-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*OclAny*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-OclAny*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Staff-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*OclAny*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-OclAny*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Flight-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*OclAny*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-OclAny*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Person}$*-OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Staff*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Staff*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Staff*)
lemma *cp-OclIsTypeOf*$_{Person}$*-OclAny-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Staff*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Staff*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Person-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Staff*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Staff*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Staff-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Staff*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Staff*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Flight-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Staff*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Staff*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Client-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Flight*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Flight*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Reservation-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Flight*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Flight*)
lemma *cp-OclIsTypeOf*$_{Person}$*-OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Flight*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclIsTypeOf*(*Person*)))))
by(*rule cpI1*, *simp add*: *OclIsTypeOf*$_{Person}$*-Flight*)
lemma *cp-OclIsTypeOf*$_{Person}$*-Staff-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Flight*) .*oclIsTypeOf*(*Person*)))))

by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Person}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Person}$-*Flight-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclIsTypeOf(Person)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Person}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Client-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Reservation-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*OclAny-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Person-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Staff-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Client)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Flight-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Client)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Client-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Reservation-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*OclAny-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Person-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Staff-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Flight-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Client-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Reservation-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*OclAny-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Person-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Staff-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Flight-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Client-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Reservation-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*OclAny-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Person-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Staff-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Flight-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Client-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Staff)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Reservation-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*OclAny-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Person-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Staff-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Flight-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Client-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclIsTypeOf(Staff)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Reservation-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclIsTypeOf(Staff)))))$

by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclIsTypeOf*(*Staff*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclIsTypeOf*(*Staff*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Staff-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Flight*) .*oclIsTypeOf*(*Staff*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Staff}$-*Flight-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Flight*) .*oclIsTypeOf*(*Staff*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Staff}$-*Flight*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Client*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Client*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Reservation*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Client-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*OclAny*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Reservation-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*OclAny*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*OclAny-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*OclAny*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Person-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*OclAny*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Staff-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*OclAny*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Flight-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*OclAny*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*OclAny*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Person*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Staff*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*OclAny-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Staff*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Staff*)
lemma *cp-OclIsTypeOf*$_{Flight}$-*Person-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Staff*) .*oclIsTypeOf*(*Flight*)))))
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}$-*Staff*)

lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}Staff\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}Flight\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp add*: $OclIsTypeOf_{Flight}\text{-}Staff$)
lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}Client\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}Reservation\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}OclAny\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}Person\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}Staff\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp*)
lemma $cp\text{-}OclIsTypeOf_{Flight}\text{-}Flight\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclIsTypeOf(Flight)))))$
by(*rule cpI1*, *simp*)

lemmas[*simp*,*code-unfold*] = $cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Client$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Client$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Person$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Person$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Client}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Client}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Client$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Client$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}OclAny$

$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Person$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Person$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}Client$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Person\text{-}Client$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}Person$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Person\text{-}Person$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsTypeOf_{OclAny}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Client\text{-}Client$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsTypeOf_{Person}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Person\text{-}Client$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Person}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Person}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Client\text{-}Person$

$cp\text{-}OclIsTypeOf_{Person}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsTypeOf_{Person}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Person\text{-}Person$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Person}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Person}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Person}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Client\text{-}Client$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Person\text{-}Client$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Client\text{-}Person$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Person\text{-}Person$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Staff}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Client\text{-}Client$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Person\text{-}Client$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Client\text{-}Person$
$cp\text{-}OclIsTypeOf_{Flight}\text{-}Reservation\text{-}Person$

cp-OclIsTypeOf$_{Flight}$-OclAny-Person
cp-OclIsTypeOf$_{Flight}$-Person-Person
cp-OclIsTypeOf$_{Flight}$-Staff-Person
cp-OclIsTypeOf$_{Flight}$-Flight-Person
cp-OclIsTypeOf$_{Flight}$-Client-Staff
cp-OclIsTypeOf$_{Flight}$-Reservation-Staff
cp-OclIsTypeOf$_{Flight}$-OclAny-Staff
cp-OclIsTypeOf$_{Flight}$-Person-Staff
cp-OclIsTypeOf$_{Flight}$-Staff-Staff
cp-OclIsTypeOf$_{Flight}$-Flight-Staff
cp-OclIsTypeOf$_{Flight}$-Client-Flight
cp-OclIsTypeOf$_{Flight}$-Reservation-Flight
cp-OclIsTypeOf$_{Flight}$-OclAny-Flight
cp-OclIsTypeOf$_{Flight}$-Person-Flight
cp-OclIsTypeOf$_{Flight}$-Staff-Flight
cp-OclIsTypeOf$_{Flight}$-Flight-Flight

Execution with Invalid or Null as Argument

lemma *OclIsTypeOf$_{Client}$-Client-invalid* : $((invalid::\cdot Client)\ .oclIsTypeOf(Client)) = invalid$
by(*rule ext*, *simp add*: *bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Client}$-Reservation-invalid* : $((invalid::\cdot Reservation)\ .oclIsTypeOf(Client)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Reservation bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Client}$-OclAny-invalid* : $((invalid::\cdot OclAny)\ .oclIsTypeOf(Client)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-OclAny bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Client}$-Person-invalid* : $((invalid::\cdot Person)\ .oclIsTypeOf(Client)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Person bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Client}$-Staff-invalid* : $((invalid::\cdot Staff)\ .oclIsTypeOf(Client)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Staff bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Client}$-Flight-invalid* : $((invalid::\cdot Flight)\ .oclIsTypeOf(Client)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Flight bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Client}$-Client-null* : $((null::\cdot Client)\ .oclIsTypeOf(Client)) = true$
by(*rule ext*, *simp add*: *bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Client}$-Reservation-null* : $((null::\cdot Reservation)\ .oclIsTypeOf(Client)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Reservation bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Client}$-OclAny-null* : $((null::\cdot OclAny)\ .oclIsTypeOf(Client)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-OclAny bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Client}$-Person-null* : $((null::\cdot Person)\ .oclIsTypeOf(Client)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Person bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Client}$-Staff-null* : $((null::\cdot Staff)\ .oclIsTypeOf(Client)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Staff bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Client}$-Flight-null* : $((null::\cdot Flight)\ .oclIsTypeOf(Client)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Client}$-Flight bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Reservation}$-Client-invalid* : $((invalid::\cdot Client)\ .oclIsTypeOf(Reservation)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Client bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Reservation}$-Reservation-invalid* : $((invalid::\cdot Reservation)\ .oclIsTypeOf(Reservation)) = invalid$
by(*rule ext*, *simp add*: *bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Reservation}$-OclAny-invalid* : $((invalid::\cdot OclAny)\ .oclIsTypeOf(Reservation)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-OclAny bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Reservation}$-Person-invalid* : $((invalid::\cdot Person)\ .oclIsTypeOf(Reservation)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Person bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Reservation}$-Staff-invalid* : $((invalid::\cdot Staff)\ .oclIsTypeOf(Reservation)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Staff bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Reservation}$-Flight-invalid* : $((invalid::\cdot Flight)\ .oclIsTypeOf(Reservation)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Flight bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{Reservation}$-Client-null* : $((null::\cdot Client)\ .oclIsTypeOf(Reservation)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Client bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Reservation}$-Reservation-null* : $((null::\cdot Reservation)\ .oclIsTypeOf(Reservation)) = true$
by(*rule ext*, *simp add*: *bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Reservation}$-OclAny-null* : $((null::\cdot OclAny)\ .oclIsTypeOf(Reservation)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-OclAny bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Reservation}$-Person-null* : $((null::\cdot Person)\ .oclIsTypeOf(Reservation)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Person bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Reservation}$-Staff-null* : $((null::\cdot Staff)\ .oclIsTypeOf(Reservation)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Staff bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{Reservation}$-Flight-null* : $((null::\cdot Flight)\ .oclIsTypeOf(Reservation)) = true$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{Reservation}$-Flight bot-option-def null-fun-def null-option-def*)
lemma *OclIsTypeOf$_{OclAny}$-Client-invalid* : $((invalid::\cdot Client)\ .oclIsTypeOf(OclAny)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{OclAny}$-Client bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{OclAny}$-Reservation-invalid* : $((invalid::\cdot Reservation)\ .oclIsTypeOf(OclAny)) = invalid$
by(*rule ext*, *simp add*: *OclIsTypeOf$_{OclAny}$-Reservation bot-option-def invalid-def*)
lemma *OclIsTypeOf$_{OclAny}$-OclAny-invalid* : $((invalid::\cdot OclAny)\ .oclIsTypeOf(OclAny)) = invalid$
by(*rule ext*, *simp add*: *bot-option-def invalid-def*)

lemma $OclIsTypeOf_{OclAny}$-Person-invalid : $((invalid::\cdot Person)\ .oclIsTypeOf(OclAny)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Person *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{OclAny}$-Staff-invalid : $((invalid::\cdot Staff)\ .oclIsTypeOf(OclAny)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Staff *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{OclAny}$-Flight-invalid : $((invalid::\cdot Flight)\ .oclIsTypeOf(OclAny)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Flight *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{OclAny}$-Client-null : $((null::\cdot Client)\ .oclIsTypeOf(OclAny)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Client *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{OclAny}$-Reservation-null : $((null::\cdot Reservation)\ .oclIsTypeOf(OclAny)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Reservation *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{OclAny}$-OclAny-null : $((null::\cdot OclAny)\ .oclIsTypeOf(OclAny)) = true$
by(*rule ext*, *simp add*: *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{OclAny}$-Person-null : $((null::\cdot Person)\ .oclIsTypeOf(OclAny)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Person *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{OclAny}$-Staff-null : $((null::\cdot Staff)\ .oclIsTypeOf(OclAny)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Staff *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{OclAny}$-Flight-null : $((null::\cdot Flight)\ .oclIsTypeOf(OclAny)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{OclAny}$-Flight *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Person}$-Client-invalid : $((invalid::\cdot Client)\ .oclIsTypeOf(Person)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Client *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Person}$-Reservation-invalid : $((invalid::\cdot Reservation)\ .oclIsTypeOf(Person)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Reservation *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Person}$-OclAny-invalid : $((invalid::\cdot OclAny)\ .oclIsTypeOf(Person)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-OclAny *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Person}$-Person-invalid : $((invalid::\cdot Person)\ .oclIsTypeOf(Person)) = invalid$
by(*rule ext*, *simp add*: *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Person}$-Staff-invalid : $((invalid::\cdot Staff)\ .oclIsTypeOf(Person)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Staff *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Person}$-Flight-invalid : $((invalid::\cdot Flight)\ .oclIsTypeOf(Person)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Flight *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Person}$-Client-null : $((null::\cdot Client)\ .oclIsTypeOf(Person)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Client *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Person}$-Reservation-null : $((null::\cdot Reservation)\ .oclIsTypeOf(Person)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Reservation *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Person}$-OclAny-null : $((null::\cdot OclAny)\ .oclIsTypeOf(Person)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-OclAny *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Person}$-Person-null : $((null::\cdot Person)\ .oclIsTypeOf(Person)) = true$
by(*rule ext*, *simp add*: *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Person}$-Staff-null : $((null::\cdot Staff)\ .oclIsTypeOf(Person)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Staff *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Person}$-Flight-null : $((null::\cdot Flight)\ .oclIsTypeOf(Person)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Person}$-Flight *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Staff}$-Client-invalid : $((invalid::\cdot Client)\ .oclIsTypeOf(Staff)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Client *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Staff}$-Reservation-invalid : $((invalid::\cdot Reservation)\ .oclIsTypeOf(Staff)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Reservation *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Staff}$-OclAny-invalid : $((invalid::\cdot OclAny)\ .oclIsTypeOf(Staff)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-OclAny *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Staff}$-Person-invalid : $((invalid::\cdot Person)\ .oclIsTypeOf(Staff)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Person *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Staff}$-Staff-invalid : $((invalid::\cdot Staff)\ .oclIsTypeOf(Staff)) = invalid$
by(*rule ext*, *simp add*: *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Staff}$-Flight-invalid : $((invalid::\cdot Flight)\ .oclIsTypeOf(Staff)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Flight *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Staff}$-Client-null : $((null::\cdot Client)\ .oclIsTypeOf(Staff)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Client *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Staff}$-Reservation-null : $((null::\cdot Reservation)\ .oclIsTypeOf(Staff)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Reservation *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Staff}$-OclAny-null : $((null::\cdot OclAny)\ .oclIsTypeOf(Staff)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-OclAny *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Staff}$-Person-null : $((null::\cdot Person)\ .oclIsTypeOf(Staff)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Person *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Staff}$-Staff-null : $((null::\cdot Staff)\ .oclIsTypeOf(Staff)) = true$
by(*rule ext*, *simp add*: *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Staff}$-Flight-null : $((null::\cdot Flight)\ .oclIsTypeOf(Staff)) = true$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Staff}$-Flight *bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Flight}$-Client-invalid : $((invalid::\cdot Client)\ .oclIsTypeOf(Flight)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-Client *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Flight}$-Reservation-invalid : $((invalid::\cdot Reservation)\ .oclIsTypeOf(Flight)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-Reservation *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Flight}$-OclAny-invalid : $((invalid::\cdot OclAny)\ .oclIsTypeOf(Flight)) = invalid$
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-OclAny *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Flight}$-Person-invalid : $((invalid::\cdot Person)\ .oclIsTypeOf(Flight)) = invalid$

by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-*Person bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Flight}$-*Staff-invalid* : ((*invalid*::·*Staff*) .*oclIsTypeOf*(*Flight*)) = *invalid*
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-*Staff bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Flight}$-*Flight-invalid* : ((*invalid*::·*Flight*) .*oclIsTypeOf*(*Flight*)) = *invalid*
by(*rule ext*, *simp add*: *bot-option-def invalid-def*)
lemma $OclIsTypeOf_{Flight}$-*Client-null* : ((*null*::·*Client*) .*oclIsTypeOf*(*Flight*)) = *true*
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-*Client bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Flight}$-*Reservation-null* : ((*null*::·*Reservation*) .*oclIsTypeOf*(*Flight*)) = *true*
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-*Reservation bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Flight}$-*OclAny-null* : ((*null*::·*OclAny*) .*oclIsTypeOf*(*Flight*)) = *true*
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-*OclAny bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Flight}$-*Person-null* : ((*null*::·*Person*) .*oclIsTypeOf*(*Flight*)) = *true*
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-*Person bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Flight}$-*Staff-null* : ((*null*::·*Staff*) .*oclIsTypeOf*(*Flight*)) = *true*
by(*rule ext*, *simp add*: $OclIsTypeOf_{Flight}$-*Staff bot-option-def null-fun-def null-option-def*)
lemma $OclIsTypeOf_{Flight}$-*Flight-null* : ((*null*::·*Flight*) .*oclIsTypeOf*(*Flight*)) = *true*
by(*rule ext*, *simp add*: *bot-option-def null-fun-def null-option-def*)

lemmas[*simp*,*code-unfold*] = $OclIsTypeOf_{Client}$-*Client-invalid*
$OclIsTypeOf_{Client}$-*Reservation-invalid*
$OclIsTypeOf_{Client}$-*OclAny-invalid*
$OclIsTypeOf_{Client}$-*Person-invalid*
$OclIsTypeOf_{Client}$-*Staff-invalid*
$OclIsTypeOf_{Client}$-*Flight-invalid*
$OclIsTypeOf_{Client}$-*Client-null*
$OclIsTypeOf_{Client}$-*Reservation-null*
$OclIsTypeOf_{Client}$-*OclAny-null*
$OclIsTypeOf_{Client}$-*Person-null*
$OclIsTypeOf_{Client}$-*Staff-null*
$OclIsTypeOf_{Client}$-*Flight-null*
$OclIsTypeOf_{Reservation}$-*Client-invalid*
$OclIsTypeOf_{Reservation}$-*Reservation-invalid*
$OclIsTypeOf_{Reservation}$-*OclAny-invalid*
$OclIsTypeOf_{Reservation}$-*Person-invalid*
$OclIsTypeOf_{Reservation}$-*Staff-invalid*
$OclIsTypeOf_{Reservation}$-*Flight-invalid*
$OclIsTypeOf_{Reservation}$-*Client-null*
$OclIsTypeOf_{Reservation}$-*Reservation-null*
$OclIsTypeOf_{Reservation}$-*OclAny-null*
$OclIsTypeOf_{Reservation}$-*Person-null*
$OclIsTypeOf_{Reservation}$-*Staff-null*
$OclIsTypeOf_{Reservation}$-*Flight-null*
$OclIsTypeOf_{OclAny}$-*Client-invalid*
$OclIsTypeOf_{OclAny}$-*Reservation-invalid*
$OclIsTypeOf_{OclAny}$-*OclAny-invalid*
$OclIsTypeOf_{OclAny}$-*Person-invalid*
$OclIsTypeOf_{OclAny}$-*Staff-invalid*
$OclIsTypeOf_{OclAny}$-*Flight-invalid*
$OclIsTypeOf_{OclAny}$-*Client-null*
$OclIsTypeOf_{OclAny}$-*Reservation-null*
$OclIsTypeOf_{OclAny}$-*OclAny-null*
$OclIsTypeOf_{OclAny}$-*Person-null*
$OclIsTypeOf_{OclAny}$-*Staff-null*
$OclIsTypeOf_{OclAny}$-*Flight-null*
$OclIsTypeOf_{Person}$-*Client-invalid*
$OclIsTypeOf_{Person}$-*Reservation-invalid*
$OclIsTypeOf_{Person}$-*OclAny-invalid*
$OclIsTypeOf_{Person}$-*Person-invalid*
$OclIsTypeOf_{Person}$-*Staff-invalid*
$OclIsTypeOf_{Person}$-*Flight-invalid*
$OclIsTypeOf_{Person}$-*Client-null*
$OclIsTypeOf_{Person}$-*Reservation-null*
$OclIsTypeOf_{Person}$-*OclAny-null*
$OclIsTypeOf_{Person}$-*Person-null*
$OclIsTypeOf_{Person}$-*Staff-null*
$OclIsTypeOf_{Person}$-*Flight-null*
$OclIsTypeOf_{Staff}$-*Client-invalid*
$OclIsTypeOf_{Staff}$-*Reservation-invalid*
$OclIsTypeOf_{Staff}$-*OclAny-invalid*
$OclIsTypeOf_{Staff}$-*Person-invalid*
$OclIsTypeOf_{Staff}$-*Staff-invalid*
$OclIsTypeOf_{Staff}$-*Flight-invalid*

OclIsTypeOf$_{Staff}$-Client-null
OclIsTypeOf$_{Staff}$-Reservation-null
OclIsTypeOf$_{Staff}$-OclAny-null
OclIsTypeOf$_{Staff}$-Person-null
OclIsTypeOf$_{Staff}$-Staff-null
OclIsTypeOf$_{Staff}$-Flight-null
OclIsTypeOf$_{Flight}$-Client-invalid
OclIsTypeOf$_{Flight}$-Reservation-invalid
OclIsTypeOf$_{Flight}$-OclAny-invalid
OclIsTypeOf$_{Flight}$-Person-invalid
OclIsTypeOf$_{Flight}$-Staff-invalid
OclIsTypeOf$_{Flight}$-Flight-invalid
OclIsTypeOf$_{Flight}$-Client-null
OclIsTypeOf$_{Flight}$-Reservation-null
OclIsTypeOf$_{Flight}$-OclAny-null
OclIsTypeOf$_{Flight}$-Person-null
OclIsTypeOf$_{Flight}$-Staff-null
OclIsTypeOf$_{Flight}$-Flight-null

Validity and Definedness Properties

lemma *OclIsTypeOf$_{Flight}$-Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsTypeOf(Flight)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Flight}$-Flight split*: *option.split ty$\mathcal{EXT}_{Flight}$.split ty$_{Flight}$.split*)

lemma *OclIsTypeOf$_{Flight}$-OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsTypeOf(Flight)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Flight}$-OclAny split*: *option.split ty$\mathcal{EXT}_{OclAny}$.split ty$_{OclAny}$.split*)
lemma *OclIsTypeOf$_{Flight}$-Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(Flight)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Flight}$-Staff split*: *option.split ty$\mathcal{EXT}_{Staff}$.split ty$_{Staff}$.split*)
lemma *OclIsTypeOf$_{Flight}$-Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(Flight)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Flight}$-Person split*: *option.split ty$\mathcal{EXT}_{Person}$.split ty$_{Person}$.split*)
lemma *OclIsTypeOf$_{Flight}$-Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsTypeOf(Flight)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Flight}$-Client split*: *option.split ty$\mathcal{EXT}_{Client}$.split ty$_{Client}$.split*)
lemma *OclIsTypeOf$_{Flight}$-Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsTypeOf(Flight)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Flight}$-Reservation split*: *option.split ty$\mathcal{EXT}_{Reservation}$.split ty$_{Reservation}$.split*)
lemma *OclIsTypeOf$_{Client}$-Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsTypeOf(Client)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Client}$-Client split*: *option.split ty$\mathcal{EXT}_{Client}$.split ty$_{Client}$.split*)
lemma *OclIsTypeOf$_{Client}$-Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(Client)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def OclIsTypeOf$_{Client}$-Person split*: *option.split ty$\mathcal{EXT}_{Person}$.split ty$_{Person}$.split*)
lemma *OclIsTypeOf$_{Client}$-OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsTypeOf(Client)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)

by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Client}$-*OclAny split*: *option.split* $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
lemma $OclIsTypeOf_{Client}$-*Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(Client)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Client}$-*Staff split*: *option.split* $ty\mathcal{EXT}_{Staff}$.*split* ty_{Staff}.*split*)
lemma $OclIsTypeOf_{Client}$-*Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsTypeOf(Client)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Client}$-*Reservation split*: *option.split* $ty\mathcal{EXT}_{Reservation}$.*split* $ty_{Reservation}$.*split*)
lemma $OclIsTypeOf_{Client}$-*Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsTypeOf(Client)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Client}$-*Flight split*: *option.split* $ty\mathcal{EXT}_{Flight}$.*split* ty_{Flight}.*split*)

lemma $OclIsTypeOf_{Staff}$-*Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(Staff)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Staff}$-*Staff split*: *option.split* $ty\mathcal{EXT}_{Staff}$.*split* ty_{Staff}.*split*)
lemma $OclIsTypeOf_{Staff}$-*Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(Staff)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Staff}$-*Person split*: *option.split* $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
lemma $OclIsTypeOf_{Staff}$-*OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsTypeOf(Staff)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Staff}$-*OclAny split*: *option.split* $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
lemma $OclIsTypeOf_{Staff}$-*Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsTypeOf(Staff)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Staff}$-*Client split*: *option.split* $ty\mathcal{EXT}_{Client}$.*split* ty_{Client}.*split*)

lemma $OclIsTypeOf_{Staff}$-*Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsTypeOf(Staff)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Staff}$-*Reservation split*: *option.split* $ty\mathcal{EXT}_{Reservation}$.*split* $ty_{Reservation}$.*split*)
lemma $OclIsTypeOf_{Staff}$-*Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsTypeOf(Staff)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Staff}$-*Flight split*: *option.split* $ty\mathcal{EXT}_{Flight}$.*split* ty_{Flight}.*split*)

lemma $OclIsTypeOf_{Person}$-*Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(Person)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Person}$-*Person split*: *option.split* $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
lemma $OclIsTypeOf_{Person}$-*OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsTypeOf(Person)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Person}$-*OclAny split*: *option.split* $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
lemma $OclIsTypeOf_{Person}$-*Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsTypeOf(Person)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Person}$-*Client split*: *option.split* $ty\mathcal{EXT}_{Client}$.*split* ty_{Client}.*split*)
lemma $OclIsTypeOf_{Person}$-*Staff-defined* :

assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsTypeOf(Person)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Person}$*-Staff split*: *option.split* $ty\mathcal{EXT}_{Staff}$*.split* ty_{Staff}*.split*)

lemma $OclIsTypeOf_{Person}$*-Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsTypeOf(Person)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Person}$*-Reservation split*: *option.split* $ty\mathcal{EXT}_{Reservation}$*.split* $ty_{Reservation}$*.split*)
lemma $OclIsTypeOf_{Person}$*-Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsTypeOf(Person)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Person}$*-Flight split*: *option.split* $ty\mathcal{EXT}_{Flight}$*.split* ty_{Flight}*.split*)
lemma $OclIsTypeOf_{Reservation}$*-Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsTypeOf(Reservation)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Reservation}$*-Reservation split*: *option.split* $ty\mathcal{EXT}_{Reservation}$*.split* $ty_{Reservation}$*.split*)
lemma $OclIsTypeOf_{Reservation}$*-OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsTypeOf(Reservation)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Reservation}$*-OclAny split*: *option.split* $ty\mathcal{EXT}_{OclAny}$*.split* ty_{OclAny}*.split*)
lemma $OclIsTypeOf_{Reservation}$*-Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsTypeOf(Reservation)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Reservation}$*-Staff split*: *option.split* $ty\mathcal{EXT}_{Staff}$*.split* ty_{Staff}*.split*)
lemma $OclIsTypeOf_{Reservation}$*-Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsTypeOf(Reservation)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Reservation}$*-Person split*: *option.split* $ty\mathcal{EXT}_{Person}$*.split* ty_{Person}*.split*)
lemma $OclIsTypeOf_{Reservation}$*-Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsTypeOf(Reservation)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Reservation}$*-Client split*: *option.split* $ty\mathcal{EXT}_{Client}$*.split* ty_{Client}*.split*)
lemma $OclIsTypeOf_{Reservation}$*-Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsTypeOf(Reservation)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{Reservation}$*-Flight split*: *option.split* $ty\mathcal{EXT}_{Flight}$*.split* ty_{Flight}*.split*)
lemma $OclIsTypeOf_{OclAny}$*-OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{OclAny}$*-OclAny split*: *option.split* $ty\mathcal{EXT}_{OclAny}$*.split* ty_{OclAny}*.split*)
lemma $OclIsTypeOf_{OclAny}$*-Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsTypeOf(OclAny)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{OclAny}$*-Flight split*: *option.split* $ty\mathcal{EXT}_{Flight}$*.split* ty_{Flight}*.split*)
lemma $OclIsTypeOf_{OclAny}$*-Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsTypeOf(OclAny)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{OclAny}$*-Client split*: *option.split* $ty\mathcal{EXT}_{Client}$*.split* ty_{Client}*.split*)
lemma $OclIsTypeOf_{OclAny}$*-Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$

shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(OclAny)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{OclAny}$*-Staff split*: *option.split* $ty\mathcal{EXT}_{Staff}$*.split* ty_{Staff}*.split*)

lemma $OclIsTypeOf_{OclAny}$*-Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(OclAny)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{OclAny}$*-Person split*: *option.split* $ty\mathcal{EXT}_{Person}$*.split* ty_{Person}*.split*)
lemma $OclIsTypeOf_{OclAny}$*-Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsTypeOf(OclAny)))$
apply(*insert isdef*[*simplified foundation18'*], *simp only*: *OclValid-def*, *subst cp-defined*)
by(*auto simp*: *cp-defined*[*symmetric*] *bot-option-def* $OclIsTypeOf_{OclAny}$*-Reservation split*: *option.split* $ty\mathcal{EXT}_{Reservation}$*.split* $ty_{Reservation}$*.split*)

lemma $OclIsTypeOf_{Flight}$*-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsTypeOf(Flight)))$
by(*rule* $OclIsTypeOf_{Flight}$*-Flight-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Flight}$*-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsTypeOf(Flight)))$
by(*rule* $OclIsTypeOf_{Flight}$*-OclAny-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Flight}$*-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(Flight)))$
by(*rule* $OclIsTypeOf_{Flight}$*-Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Flight}$*-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(Flight)))$
by(*rule* $OclIsTypeOf_{Flight}$*-Person-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Flight}$*-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsTypeOf(Flight)))$
by(*rule* $OclIsTypeOf_{Flight}$*-Client-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Flight}$*-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsTypeOf(Flight)))$
by(*rule* $OclIsTypeOf_{Flight}$*-Reservation-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Client}$*-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsTypeOf(Client)))$
by(*rule* $OclIsTypeOf_{Client}$*-Client-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Client}$*-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(Client)))$
by(*rule* $OclIsTypeOf_{Client}$*-Person-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Client}$*-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsTypeOf(Client)))$
by(*rule* $OclIsTypeOf_{Client}$*-OclAny-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Client}$*-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(Client)))$
by(*rule* $OclIsTypeOf_{Client}$*-Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Client}$*-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsTypeOf(Client)))$
by(*rule* $OclIsTypeOf_{Client}$*-Reservation-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Client}$*-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsTypeOf(Client)))$
by(*rule* $OclIsTypeOf_{Client}$*-Flight-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Staff}$*-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(Staff)))$
by(*rule* $OclIsTypeOf_{Staff}$*-Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsTypeOf_{Staff}$*-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(Staff)))$

by(*rule OclIsTypeOf$_{Staff}$-Person-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Staff}$-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsTypeOf(Staff)))$
by(*rule OclIsTypeOf$_{Staff}$-OclAny-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Staff}$-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsTypeOf(Staff)))$
by(*rule OclIsTypeOf$_{Staff}$-Client-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Staff}$-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsTypeOf(Staff)))$
by(*rule OclIsTypeOf$_{Staff}$-Reservation-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Staff}$-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsTypeOf(Staff)))$
by(*rule OclIsTypeOf$_{Staff}$-Flight-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Person}$-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsTypeOf(Person)))$
by(*rule OclIsTypeOf$_{Person}$-Person-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Person}$-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsTypeOf(Person)))$
by(*rule OclIsTypeOf$_{Person}$-OclAny-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Person}$-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsTypeOf(Person)))$
by(*rule OclIsTypeOf$_{Person}$-Client-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Person}$-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsTypeOf(Person)))$
by(*rule OclIsTypeOf$_{Person}$-Staff-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Person}$-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsTypeOf(Person)))$
by(*rule OclIsTypeOf$_{Person}$-Reservation-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Person}$-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsTypeOf(Person)))$
by(*rule OclIsTypeOf$_{Person}$-Flight-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Reservation}$-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsTypeOf(Reservation)))$
by(*rule OclIsTypeOf$_{Reservation}$-Reservation-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Reservation}$-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsTypeOf(Reservation)))$
by(*rule OclIsTypeOf$_{Reservation}$-OclAny-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Reservation}$-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsTypeOf(Reservation)))$
by(*rule OclIsTypeOf$_{Reservation}$-Staff-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Reservation}$-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsTypeOf(Reservation)))$
by(*rule OclIsTypeOf$_{Reservation}$-Person-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Reservation}$-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsTypeOf(Reservation)))$
by(*rule OclIsTypeOf$_{Reservation}$-Client-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{Reservation}$-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsTypeOf(Reservation)))$
by(*rule OclIsTypeOf$_{Reservation}$-Flight-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{OclAny}$-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny)))$
by(*rule OclIsTypeOf$_{OclAny}$-OclAny-defined[OF isdef[THEN foundation20]]*)
lemma *OclIsTypeOf$_{OclAny}$-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsTypeOf(OclAny)))$
by(*rule OclIsTypeOf$_{OclAny}$-Flight-defined[OF isdef[THEN foundation20]]*)

lemma $OclIsTypeOf_{OclAny}$-Client-defined′ :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsTypeOf(OclAny)))$
by(rule $OclIsTypeOf_{OclAny}$-Client-defined[OF isdef[THEN foundation20]])
lemma $OclIsTypeOf_{OclAny}$-Staff-defined′ :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsTypeOf(OclAny)))$
by(rule $OclIsTypeOf_{OclAny}$-Staff-defined[OF isdef[THEN foundation20]])
lemma $OclIsTypeOf_{OclAny}$-Person-defined′ :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsTypeOf(OclAny)))$
by(rule $OclIsTypeOf_{OclAny}$-Person-defined[OF isdef[THEN foundation20]])
lemma $OclIsTypeOf_{OclAny}$-Reservation-defined′ :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsTypeOf(OclAny)))$
by(rule $OclIsTypeOf_{OclAny}$-Reservation-defined[OF isdef[THEN foundation20]])

Up Down Casting

lemma $actualType_{Flight}$-larger-$staticType_{OclAny}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Flight)\ .oclIsTypeOf(OclAny)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{OclAny}$-Flight foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Flight}$-larger-$staticType_{Staff}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Flight)\ .oclIsTypeOf(Staff)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Staff}$-Flight foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Flight}$-larger-$staticType_{Person}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Flight)\ .oclIsTypeOf(Person)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Person}$-Flight foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Flight}$-larger-$staticType_{Client}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Flight)\ .oclIsTypeOf(Client)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Client}$-Flight foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Flight}$-larger-$staticType_{Reservation}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Flight)\ .oclIsTypeOf(Reservation)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Reservation}$-Flight foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Client}$-larger-$staticType_{Person}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Client)\ .oclIsTypeOf(Person)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Person}$-Client foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Client}$-larger-$staticType_{OclAny}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Client)\ .oclIsTypeOf(OclAny)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{OclAny}$-Client foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Client}$-larger-$staticType_{Staff}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Client)\ .oclIsTypeOf(Staff)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Staff}$-Client foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Client}$-larger-$staticType_{Reservation}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Client)\ .oclIsTypeOf(Reservation)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Reservation}$-Client foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Client}$-larger-$staticType_{Flight}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Client)\ .oclIsTypeOf(Flight)) \triangleq false$
using isdef
by(auto simp: $OclIsTypeOf_{Flight}$-Client foundation22 foundation16 null-option-def bot-option-def)
lemma $actualType_{Staff}$-larger-$staticType_{Person}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::Staff)\ .oclIsTypeOf(Person)) \triangleq false$
using isdef

by(*auto simp*: $OclIsTypeOf_{Person}$-*Staff foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Staff}$-*larger-staticType*$_{OclAny}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Staff)\ .oclIsTypeOf(OclAny)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{OclAny}$-*Staff foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Staff}$-*larger-staticType*$_{Client}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Staff)\ .oclIsTypeOf(Client)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Client}$-*Staff foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Staff}$-*larger-staticType*$_{Reservation}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Staff)\ .oclIsTypeOf(Reservation)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Reservation}$-*Staff foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Staff}$-*larger-staticType*$_{Flight}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Staff)\ .oclIsTypeOf(Flight)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Flight}$-*Staff foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Person}$-*larger-staticType*$_{OclAny}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(OclAny)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{OclAny}$-*Person foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Person}$-*larger-staticType*$_{Reservation}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Reservation)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Reservation}$-*Person foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Person}$-*larger-staticType*$_{Flight}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Flight)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Flight}$-*Person foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Reservation}$-*larger-staticType*$_{OclAny}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Reservation)\ .oclIsTypeOf(OclAny)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{OclAny}$-*Reservation foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Reservation}$-*larger-staticType*$_{Staff}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Reservation)\ .oclIsTypeOf(Staff)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Staff}$-*Reservation foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Reservation}$-*larger-staticType*$_{Person}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Reservation)\ .oclIsTypeOf(Person)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Person}$-*Reservation foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Reservation}$-*larger-staticType*$_{Client}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Reservation)\ .oclIsTypeOf(Client)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Client}$-*Reservation foundation22 foundation16 null-option-def bot-option-def*)
lemma $actualType_{Reservation}$-*larger-staticType*$_{Flight}$:
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Reservation)\ .oclIsTypeOf(Flight)) \triangleq false$
using *isdef*
by(*auto simp*: $OclIsTypeOf_{Flight}$-*Reservation foundation22 foundation16 null-option-def bot-option-def*)

lemma *down-cast-type*$_{OclAny}$-*from-OclAny-to-Flight* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Flight)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Flight}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}.split$ $ty_{OclAny}.split$)
by(*simp add*: *OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Staff}$-*from-OclAny-to-Flight* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Staff))$

assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Flight)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Flight}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Staff}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Person}$-*from-OclAny-to-Flight* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Flight)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Flight}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Person}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Client}$-*from-OclAny-to-Flight* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Flight)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Flight}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Client}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Reservation}$-*from-OclAny-to-Flight* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Flight)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Flight}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Reservation}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Person}$-*from-Person-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: *OclValid-def false-def true-def*)
lemma *down-cast-type*$_{OclAny}$-*from-OclAny-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: *OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Person}$-*from-OclAny-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Person}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Staff}$-*from-Person-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Staff))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: $OclIsTypeOf_{Staff}$-*Person OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Reservation}$-*from-Person-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: $OclIsTypeOf_{Reservation}$-*Person OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Flight}$-*from-Person-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$

shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: $OclIsTypeOf_{Flight}$-*Person OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Staff}$-*from-OclAny-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Staff))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Staff}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Reservation}$-*from-OclAny-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Reservation}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Flight}$-*from-OclAny-to-Client* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Client}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Flight}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Person}$-*from-Person-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: *OclValid-def false-def true-def*)
lemma *down-cast-type*$_{OclAny}$-*from-OclAny-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: *OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Person}$-*from-OclAny-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Person}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Client}$-*from-Person-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: $OclIsTypeOf_{Client}$-*Person OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Reservation}$-*from-Person-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: $OclIsTypeOf_{Reservation}$-*Person OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Flight}$-*from-Person-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$

using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*Person foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{Person}$.*split* ty_{Person}.*split*)
by(*simp add*: $OclIsTypeOf_{Flight}$-*Person OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Client}$-*from-OclAny-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Client}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Reservation}$-*from-OclAny-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Reservation}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Flight}$-*from-OclAny-to-Staff* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Staff}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Flight}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{OclAny}$-*from-OclAny-to-Person* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Person)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Person}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: *OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Reservation}$-*from-OclAny-to-Person* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Person)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Person}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Reservation}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Flight}$-*from-OclAny-to-Person* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Person)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Person}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Flight}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{OclAny}$-*from-OclAny-to-Reservation* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Reservation)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Reservation}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: *OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Staff}$-*from-OclAny-to-Reservation* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Staff))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Reservation)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Reservation}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Staff}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Person}$-*from-OclAny-to-Reservation* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Reservation)) \triangleq invalid$
using *istyp isdef*

apply(*auto simp*: $OclAsType_{Reservation}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Person}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Client}$-*from-OclAny-to-Reservation* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Reservation)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Reservation}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Client}$-*OclAny OclValid-def false-def true-def*)
lemma *down-cast-type*$_{Flight}$-*from-OclAny-to-Reservation* :
assumes *istyp*: $\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Reservation)) \triangleq invalid$
using *istyp isdef*
apply(*auto simp*: $OclAsType_{Reservation}$-*OclAny foundation22 foundation16 null-option-def bot-option-def split*: $ty\mathcal{EXT}_{OclAny}$.*split* ty_{OclAny}.*split*)
by(*simp add*: $OclIsTypeOf_{Flight}$-*OclAny OclValid-def false-def true-def*)

Const

B.6 Class Model: OclIsKindOf

Definition

consts $OclIsKindOf_{Flight} :: 'α \Rightarrow Boolean\ ((\text{-})\ .oclIsKindOf'(Flight'))$
consts $OclIsKindOf_{Client} :: 'α \Rightarrow Boolean\ ((\text{-})\ .oclIsKindOf'(Client'))$
consts $OclIsKindOf_{Staff} :: 'α \Rightarrow Boolean\ ((\text{-})\ .oclIsKindOf'(Staff'))$
consts $OclIsKindOf_{Person} :: 'α \Rightarrow Boolean\ ((\text{-})\ .oclIsKindOf'(Person'))$
consts $OclIsKindOf_{Reservation} :: 'α \Rightarrow Boolean\ ((\text{-})\ .oclIsKindOf'(Reservation'))$
consts $OclIsKindOf_{OclAny} :: 'α \Rightarrow Boolean\ ((\text{-})\ .oclIsKindOf'(OclAny'))$

overloading $OclIsKindOf_{Flight} \equiv (OclIsKindOf_{Flight}::(\cdot Flight) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Flight}$-*Flight* : $(x::\cdot Flight)\ .oclIsKindOf(Flight) \equiv (x\ .oclIsTypeOf(Flight))$
end
overloading $OclIsKindOf_{Flight} \equiv (OclIsKindOf_{Flight}::(\cdot OclAny) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Flight}$-*OclAny* : $(x::\cdot OclAny)\ .oclIsKindOf(Flight) \equiv (x\ .oclIsTypeOf(Flight))$
end
overloading $OclIsKindOf_{Flight} \equiv (OclIsKindOf_{Flight}::(\cdot Staff) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Flight}$-*Staff* : $(x::\cdot Staff)\ .oclIsKindOf(Flight) \equiv (x\ .oclIsTypeOf(Flight))$
end
overloading $OclIsKindOf_{Flight} \equiv (OclIsKindOf_{Flight}::(\cdot Person) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Flight}$-*Person* : $(x::\cdot Person)\ .oclIsKindOf(Flight) \equiv (x\ .oclIsTypeOf(Flight))$
end
overloading $OclIsKindOf_{Flight} \equiv (OclIsKindOf_{Flight}::(\cdot Client) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Flight}$-*Client* : $(x::\cdot Client)\ .oclIsKindOf(Flight) \equiv (x\ .oclIsTypeOf(Flight))$
end
overloading $OclIsKindOf_{Flight} \equiv (OclIsKindOf_{Flight}::(\cdot Reservation) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Flight}$-*Reservation* : $(x::\cdot Reservation)\ .oclIsKindOf(Flight) \equiv (x\ .oclIsTypeOf(Flight))$
end
overloading $OclIsKindOf_{Client} \equiv (OclIsKindOf_{Client}::(\cdot Client) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Client}$-*Client* : $(x::\cdot Client)\ .oclIsKindOf(Client) \equiv (x\ .oclIsTypeOf(Client))$
end
overloading $OclIsKindOf_{Client} \equiv (OclIsKindOf_{Client}::(\cdot Person) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Client}$-*Person* : $(x::\cdot Person)\ .oclIsKindOf(Client) \equiv (x\ .oclIsTypeOf(Client))$
end
overloading $OclIsKindOf_{Client} \equiv (OclIsKindOf_{Client}::(\cdot OclAny) \Rightarrow \text{-})$
begin
definition $OclIsKindOf_{Client}$-*OclAny* : $(x::\cdot OclAny)\ .oclIsKindOf(Client) \equiv (x\ .oclIsTypeOf(Client))$
end
overloading $OclIsKindOf_{Client} \equiv (OclIsKindOf_{Client}::(\cdot Staff) \Rightarrow \text{-})$
begin

definition $OclIsKindOf_{Client}$-$Staff$: $(x::\cdot Staff)$ $.oclIsKindOf(Client) \equiv (x$ $.oclIsTypeOf(Client))$
end
overloading $OclIsKindOf_{Client} \equiv (OclIsKindOf_{Client}::(\cdot Reservation) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Client}$-$Reservation$: $(x::\cdot Reservation)$ $.oclIsKindOf(Client) \equiv (x$ $.oclIsTypeOf(Client))$
end
overloading $OclIsKindOf_{Client} \equiv (OclIsKindOf_{Client}::(\cdot Flight) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Client}$-$Flight$: $(x::\cdot Flight)$ $.oclIsKindOf(Client) \equiv (x$ $.oclIsTypeOf(Client))$
end
overloading $OclIsKindOf_{Staff} \equiv (OclIsKindOf_{Staff}::(\cdot Staff) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Staff}$-$Staff$: $(x::\cdot Staff)$ $.oclIsKindOf(Staff) \equiv (x$ $.oclIsTypeOf(Staff))$
end
overloading $OclIsKindOf_{Staff} \equiv (OclIsKindOf_{Staff}::(\cdot Person) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Staff}$-$Person$: $(x::\cdot Person)$ $.oclIsKindOf(Staff) \equiv (x$ $.oclIsTypeOf(Staff))$
end
overloading $OclIsKindOf_{Staff} \equiv (OclIsKindOf_{Staff}::(\cdot OclAny) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Staff}$-$OclAny$: $(x::\cdot OclAny)$ $.oclIsKindOf(Staff) \equiv (x$ $.oclIsTypeOf(Staff))$
end
overloading $OclIsKindOf_{Staff} \equiv (OclIsKindOf_{Staff}::(\cdot Client) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Staff}$-$Client$: $(x::\cdot Client)$ $.oclIsKindOf(Staff) \equiv (x$ $.oclIsTypeOf(Staff))$
end
overloading $OclIsKindOf_{Staff} \equiv (OclIsKindOf_{Staff}::(\cdot Reservation) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Staff}$-$Reservation$: $(x::\cdot Reservation)$ $.oclIsKindOf(Staff) \equiv (x$ $.oclIsTypeOf(Staff))$
end
overloading $OclIsKindOf_{Staff} \equiv (OclIsKindOf_{Staff}::(\cdot Flight) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Staff}$-$Flight$: $(x::\cdot Flight)$ $.oclIsKindOf(Staff) \equiv (x$ $.oclIsTypeOf(Staff))$
end
overloading $OclIsKindOf_{Person} \equiv (OclIsKindOf_{Person}::(\cdot Person) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Person}$-$Person$: $(x::\cdot Person)$ $.oclIsKindOf(Person) \equiv (x$ $.oclIsTypeOf(Person))$ or $(x$ $.oclIsKindOf(Staff))$ or $(x$ $.oclIsKindOf(Client))$
end
overloading $OclIsKindOf_{Person} \equiv (OclIsKindOf_{Person}::(\cdot OclAny) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Person}$-$OclAny$: $(x::\cdot OclAny)$ $.oclIsKindOf(Person) \equiv (x$ $.oclIsTypeOf(Person))$ or $(x$ $.oclIsKindOf(Staff))$ or $(x$ $.oclIsKindOf(Client))$
end
overloading $OclIsKindOf_{Person} \equiv (OclIsKindOf_{Person}::(\cdot Client) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Person}$-$Client$: $(x::\cdot Client)$ $.oclIsKindOf(Person) \equiv (x$ $.oclIsTypeOf(Person))$ or $(x$ $.oclIsKindOf(Staff))$ or $(x$ $.oclIsKindOf(Client))$
end
overloading $OclIsKindOf_{Person} \equiv (OclIsKindOf_{Person}::(\cdot Staff) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Person}$-$Staff$: $(x::\cdot Staff)$ $.oclIsKindOf(Person) \equiv (x$ $.oclIsTypeOf(Person))$ or $(x$ $.oclIsKindOf(Staff))$ or $(x$ $.oclIsKindOf(Client))$
end
overloading $OclIsKindOf_{Person} \equiv (OclIsKindOf_{Person}::(\cdot Reservation) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Person}$-$Reservation$: $(x::\cdot Reservation)$ $.oclIsKindOf(Person) \equiv (x$ $.oclIsTypeOf(Person))$ or $(x$ $.oclIsKindOf(Staff))$ or $(x$ $.oclIsKindOf(Client))$
end
overloading $OclIsKindOf_{Person} \equiv (OclIsKindOf_{Person}::(\cdot Flight) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Person}$-$Flight$: $(x::\cdot Flight)$ $.oclIsKindOf(Person) \equiv (x$ $.oclIsTypeOf(Person))$ or $(x$ $.oclIsKindOf(Staff))$ or $(x$ $.oclIsKindOf(Client))$
end
overloading $OclIsKindOf_{Reservation} \equiv (OclIsKindOf_{Reservation}::(\cdot Reservation) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Reservation}$-$Reservation$: $(x::\cdot Reservation)$ $.oclIsKindOf(Reservation) \equiv (x$ $.oclIsTypeOf(Reservation))$
end
overloading $OclIsKindOf_{Reservation} \equiv (OclIsKindOf_{Reservation}::(\cdot OclAny) \Rightarrow$ -)
begin
definition $OclIsKindOf_{Reservation}$-$OclAny$: $(x::\cdot OclAny)$ $.oclIsKindOf(Reservation) \equiv (x$ $.oclIsTypeOf(Reservation))$
end
overloading $OclIsKindOf_{Reservation} \equiv (OclIsKindOf_{Reservation}::(\cdot Staff) \Rightarrow$ -)

```
begin
  definition OclIsKindOf_Reservation-Staff : (x::·Staff) .oclIsKindOf(Reservation) ≡ (x .oclIsTypeOf(Reservation))
end
overloading OclIsKindOf_Reservation ≡ (OclIsKindOf_Reservation::(·Person) ⇒ -)
begin
  definition OclIsKindOf_Reservation-Person : (x::·Person) .oclIsKindOf(Reservation) ≡ (x .oclIsTypeOf(Reservation))
end
overloading OclIsKindOf_Reservation ≡ (OclIsKindOf_Reservation::(·Client) ⇒ -)
begin
  definition OclIsKindOf_Reservation-Client : (x::·Client) .oclIsKindOf(Reservation) ≡ (x .oclIsTypeOf(Reservation))
end
overloading OclIsKindOf_Reservation ≡ (OclIsKindOf_Reservation::(·Flight) ⇒ -)
begin
  definition OclIsKindOf_Reservation-Flight : (x::·Flight) .oclIsKindOf(Reservation) ≡ (x .oclIsTypeOf(Reservation))
end
overloading OclIsKindOf_OclAny ≡ (OclIsKindOf_OclAny::(·OclAny) ⇒ -)
begin
    definition OclIsKindOf_OclAny-OclAny : (x::·OclAny) .oclIsKindOf(OclAny) ≡ (x .oclIsTypeOf(OclAny)) or (x
.oclIsKindOf(Reservation)) or (x .oclIsKindOf(Person)) or (x .oclIsKindOf(Flight))
end
overloading OclIsKindOf_OclAny ≡ (OclIsKindOf_OclAny::(·Flight) ⇒ -)
begin
    definition OclIsKindOf_OclAny-Flight : (x::·Flight) .oclIsKindOf(OclAny) ≡ (x .oclIsTypeOf(OclAny)) or (x
.oclIsKindOf(Reservation)) or (x .oclIsKindOf(Person)) or (x .oclIsKindOf(Flight))
end
overloading OclIsKindOf_OclAny ≡ (OclIsKindOf_OclAny::(·Client) ⇒ -)
begin
    definition OclIsKindOf_OclAny-Client : (x::·Client) .oclIsKindOf(OclAny) ≡ (x .oclIsTypeOf(OclAny)) or (x
.oclIsKindOf(Reservation)) or (x .oclIsKindOf(Person)) or (x .oclIsKindOf(Flight))
end
overloading OclIsKindOf_OclAny ≡ (OclIsKindOf_OclAny::(·Staff) ⇒ -)
begin
    definition OclIsKindOf_OclAny-Staff : (x::·Staff) .oclIsKindOf(OclAny) ≡ (x .oclIsTypeOf(OclAny)) or (x
.oclIsKindOf(Reservation)) or (x .oclIsKindOf(Person)) or (x .oclIsKindOf(Flight))
end
overloading OclIsKindOf_OclAny ≡ (OclIsKindOf_OclAny::(·Person) ⇒ -)
begin
    definition OclIsKindOf_OclAny-Person : (x::·Person) .oclIsKindOf(OclAny) ≡ (x .oclIsTypeOf(OclAny)) or (x
.oclIsKindOf(Reservation)) or (x .oclIsKindOf(Person)) or (x .oclIsKindOf(Flight))
end
overloading OclIsKindOf_OclAny ≡ (OclIsKindOf_OclAny::(·Reservation) ⇒ -)
begin
  definition OclIsKindOf_OclAny-Reservation : (x::·Reservation) .oclIsKindOf(OclAny) ≡ (x .oclIsTypeOf(OclAny)) or (x
.oclIsKindOf(Reservation)) or (x .oclIsKindOf(Person)) or (x .oclIsKindOf(Flight))
end


definition OclIsKindOf_Flight-𝔄 = (λ (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsKindOf(Flight))
    | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsKindOf(Flight))
    | (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsKindOf(Flight))
    | (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsKindOf(Flight))
    | (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsKindOf(Flight))
    | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsKindOf(Flight)))
definition OclIsKindOf_Client-𝔄 = (λ (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsKindOf(Client))
    | (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsKindOf(Client))
    | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsKindOf(Client))
    | (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsKindOf(Client))
    | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsKindOf(Client))
    | (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsKindOf(Client)))
definition OclIsKindOf_Staff-𝔄 = (λ (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsKindOf(Staff))
    | (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsKindOf(Staff))
    | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsKindOf(Staff))
    | (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsKindOf(Staff))
    | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsKindOf(Staff))
    | (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsKindOf(Staff)))
definition OclIsKindOf_Person-𝔄 = (λ (in_Person (Person)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Person))::·Person) .oclIsKindOf(Person))
    | (in_OclAny (OclAny)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (OclAny))::·OclAny) .oclIsKindOf(Person))
    | (in_Client (Client)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Client))::·Client) .oclIsKindOf(Person))
    | (in_Staff (Staff)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Staff))::·Staff) .oclIsKindOf(Person))
    | (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation) .oclIsKindOf(Person))
    | (in_Flight (Flight)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Flight))::·Flight) .oclIsKindOf(Person)))
definition OclIsKindOf_Reservation-𝔄 = (λ (in_Reservation (Reservation)) ⇒ (((((λx -. ⌊⌊x⌋⌋)) (Reservation))::·Reservation)
.oclIsKindOf(Reservation))
```

| $(in_{OclAny}\ (OclAny)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (OclAny))::\cdot OclAny)\ .oclIsKindOf(Reservation))$
| $(in_{Staff}\ (Staff)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Staff))::\cdot Staff)\ .oclIsKindOf(Reservation))$
| $(in_{Person}\ (Person)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Person))::\cdot Person)\ .oclIsKindOf(Reservation))$
| $(in_{Client}\ (Client)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Client))::\cdot Client)\ .oclIsKindOf(Reservation))$
| $(in_{Flight}\ (Flight)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Flight))::\cdot Flight)\ .oclIsKindOf(Reservation)))$

definition $OclIsKindOf_{OclAny}$-$\mathfrak{A} = (\lambda\ (in_{OclAny}\ (OclAny)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (OclAny))::\cdot OclAny)\ .oclIsKindOf(OclAny))$
| $(in_{Flight}\ (Flight)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Flight))::\cdot Flight)\ .oclIsKindOf(OclAny))$
| $(in_{Client}\ (Client)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Client))::\cdot Client)\ .oclIsKindOf(OclAny))$
| $(in_{Staff}\ (Staff)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Staff))::\cdot Staff)\ .oclIsKindOf(OclAny))$
| $(in_{Person}\ (Person)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Person))::\cdot Person)\ .oclIsKindOf(OclAny))$
| $(in_{Reservation}\ (Reservation)) \Rightarrow (((((\lambda x\ \text{-}.\ \lfloor\lfloor x \rfloor\rfloor))\ (Reservation))::\cdot Reservation)\ .oclIsKindOf(OclAny)))$

lemmas[*simp,code-unfold*] = $OclIsKindOf_{Flight}$-*Flight*
$OclIsKindOf_{Client}$-*Client*
$OclIsKindOf_{Staff}$-*Staff*
$OclIsKindOf_{Person}$-*Person*
$OclIsKindOf_{Reservation}$-*Reservation*
$OclIsKindOf_{OclAny}$-*OclAny*

Context Passing

lemma *cp-*$OclIsKindOf_{Flight}$*-Flight-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Flight*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Flight-Flight*)
lemma *cp-*$OclIsKindOf_{Flight}$*-OclAny-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Flight*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-OclAny-Flight*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Staff-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Flight*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Staff-Flight*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Person-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Flight*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Person-Flight*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Client-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Flight*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Client-Flight*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Reservation-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Flight*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Reservation-Flight*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Flight-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*OclAny*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Flight-OclAny*)
lemma *cp-*$OclIsKindOf_{Flight}$*-OclAny-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*OclAny*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-OclAny-OclAny*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Staff-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*OclAny*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Staff-OclAny*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Person-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*OclAny*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Person-OclAny*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Client-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*OclAny*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Client-OclAny*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Reservation-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*OclAny*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Reservation-OclAny*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Flight-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Staff*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Flight-Staff*)
lemma *cp-*$OclIsKindOf_{Flight}$*-OclAny-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Staff*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-OclAny-Staff*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Staff-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Staff*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Staff-Staff*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Person-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Staff*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Person-Staff*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Client-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Staff)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Staff*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Client-Staff*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Reservation-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Staff*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Reservation-Staff*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Flight-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Person*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Flight-Person*)
lemma *cp-*$OclIsKindOf_{Flight}$*-OclAny-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Person*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-OclAny-Person*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Staff-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Person*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Staff-Person*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Person-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Person*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Person-Person*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Client-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Person*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Client-Person*)
lemma *cp-*$OclIsKindOf_{Flight}$*-Reservation-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclIsKindOf(Flight)))))$
by(*simp only*: $OclIsKindOf_{Flight}$-*Person*, *simp only*: *cp-*$OclIsTypeOf_{Flight}$*-Reservation-Person*)

lemma *cp-OclIsKindOf*$_{Flight}$*-Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Flight-Client*)
lemma *cp-OclIsKindOf*$_{Flight}$*-OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-OclAny-Client*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Staff-Client*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Person-Client*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Client-Client*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Client*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Reservation-Client*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Reservation*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Flight-Reservation*)
lemma *cp-OclIsKindOf*$_{Flight}$*-OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Reservation*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-OclAny-Reservation*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Reservation*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Staff-Reservation*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Reservation*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Person-Reservation*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Reservation*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Client-Reservation*)
lemma *cp-OclIsKindOf*$_{Flight}$*-Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclIsKindOf*(*Flight*)))))
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Reservation*, *simp only*: *cp-OclIsTypeOf*$_{Flight}$*-Reservation-Reservation*)
lemma *cp-OclIsKindOf*$_{Client}$*-Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Client-Client*)
lemma *cp-OclIsKindOf*$_{Client}$*-Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Person-Client*)
lemma *cp-OclIsKindOf*$_{Client}$*-OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-OclAny-Client*)
lemma *cp-OclIsKindOf*$_{Client}$*-Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Staff-Client*)
lemma *cp-OclIsKindOf*$_{Client}$*-Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Client*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Reservation-Client*)
lemma *cp-OclIsKindOf*$_{Client}$*-Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Flight-Client*)
lemma *cp-OclIsKindOf*$_{Client}$*-Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Client-Person*)
lemma *cp-OclIsKindOf*$_{Client}$*-Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Person-Person*)
lemma *cp-OclIsKindOf*$_{Client}$*-OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-OclAny-Person*)
lemma *cp-OclIsKindOf*$_{Client}$*-Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Staff-Person*)
lemma *cp-OclIsKindOf*$_{Client}$*-Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Reservation-Person*)
lemma *cp-OclIsKindOf*$_{Client}$*-Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Flight-Person*)
lemma *cp-OclIsKindOf*$_{Client}$*-Client-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*OclAny*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-OclAny*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Client-OclAny*)
lemma *cp-OclIsKindOf*$_{Client}$*-Person-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*OclAny*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-OclAny*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Person-OclAny*)
lemma *cp-OclIsKindOf*$_{Client}$*-OclAny-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*OclAny*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-OclAny*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-OclAny-OclAny*)
lemma *cp-OclIsKindOf*$_{Client}$*-Staff-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*OclAny*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-OclAny*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Staff-OclAny*)
lemma *cp-OclIsKindOf*$_{Client}$*-Reservation-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*OclAny*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-OclAny*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Reservation-OclAny*)
lemma *cp-OclIsKindOf*$_{Client}$*-Flight-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*OclAny*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-OclAny*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Flight-OclAny*)
lemma *cp-OclIsKindOf*$_{Client}$*-Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Staff*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Client-Staff*)
lemma *cp-OclIsKindOf*$_{Client}$*-Person-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Staff*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf*$_{Client}$*-Staff*, *simp only*: *cp-OclIsTypeOf*$_{Client}$*-Person-Staff*)

lemma *cp-OclIsKindOf$_{Client}$-OclAny-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Staff*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Client}$-OclAny-Staff*)
lemma *cp-OclIsKindOf$_{Client}$-Staff-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Staff*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Staff-Staff*)
lemma *cp-OclIsKindOf$_{Client}$-Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Staff*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Reservation-Staff*)
lemma *cp-OclIsKindOf$_{Client}$-Flight-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Staff*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Flight-Staff*)
lemma *cp-OclIsKindOf$_{Client}$-Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Reservation*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Client-Reservation*)
lemma *cp-OclIsKindOf$_{Client}$-Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Reservation*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Person-Reservation*)
lemma *cp-OclIsKindOf$_{Client}$-OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Reservation*, *simp only*: *cp-OclIsTypeOf$_{Client}$-OclAny-Reservation*)
lemma *cp-OclIsKindOf$_{Client}$-Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Reservation*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Staff-Reservation*)
lemma *cp-OclIsKindOf$_{Client}$-Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Reservation*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Reservation-Reservation*)
lemma *cp-OclIsKindOf$_{Client}$-Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Reservation*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Flight-Reservation*)
lemma *cp-OclIsKindOf$_{Client}$-Client-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Flight*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Flight*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Client-Flight*)
lemma *cp-OclIsKindOf$_{Client}$-Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Flight*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Person-Flight*)
lemma *cp-OclIsKindOf$_{Client}$-OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Flight*, *simp only*: *cp-OclIsTypeOf$_{Client}$-OclAny-Flight*)
lemma *cp-OclIsKindOf$_{Client}$-Staff-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Flight*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Flight*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Staff-Flight*)
lemma *cp-OclIsKindOf$_{Client}$-Reservation-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Flight*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Flight*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Reservation-Flight*)
lemma *cp-OclIsKindOf$_{Client}$-Flight-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Flight*) .*oclIsKindOf*(*Client*)))))
by(*simp only*: *OclIsKindOf$_{Client}$-Flight*, *simp only*: *cp-OclIsTypeOf$_{Client}$-Flight-Flight*)
lemma *cp-OclIsKindOf$_{Staff}$-Staff-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Staff*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Staff-Staff*)
lemma *cp-OclIsKindOf$_{Staff}$-Person-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Staff*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Person-Staff*)
lemma *cp-OclIsKindOf$_{Staff}$-OclAny-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Staff*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-OclAny-Staff*)
lemma *cp-OclIsKindOf$_{Staff}$-Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Client-Staff*)
lemma *cp-OclIsKindOf$_{Staff}$-Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Staff*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Reservation-Staff*)
lemma *cp-OclIsKindOf$_{Staff}$-Flight-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Staff*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Staff*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Flight-Staff*)
lemma *cp-OclIsKindOf$_{Staff}$-Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Person*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Staff-Person*)
lemma *cp-OclIsKindOf$_{Staff}$-Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Person*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Person-Person*)
lemma *cp-OclIsKindOf$_{Staff}$-OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Person*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-OclAny-Person*)
lemma *cp-OclIsKindOf$_{Staff}$-Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Person*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Client-Person*)
lemma *cp-OclIsKindOf$_{Staff}$-Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Person*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Reservation-Person*)
lemma *cp-OclIsKindOf$_{Staff}$-Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-Person*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Flight-Person*)
lemma *cp-OclIsKindOf$_{Staff}$-Staff-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*OclAny*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-OclAny*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Staff-OclAny*)
lemma *cp-OclIsKindOf$_{Staff}$-Person-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*OclAny*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-OclAny*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Person-OclAny*)
lemma *cp-OclIsKindOf$_{Staff}$-OclAny-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*OclAny*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-OclAny*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-OclAny-OclAny*)
lemma *cp-OclIsKindOf$_{Staff}$-Client-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*OclAny*) .*oclIsKindOf*(*Staff*)))))
by(*simp only*: *OclIsKindOf$_{Staff}$-OclAny*, *simp only*: *cp-OclIsTypeOf$_{Staff}$-Client-OclAny*)
lemma *cp-OclIsKindOf$_{Staff}$-Reservation-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*OclAny*) .*oclIsKindOf*(*Staff*)))))

by(*simp only*: $OclIsKindOf_{Staff}$-*OclAny*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Reservation-OclAny*)
lemma cp-$OclIsKindOf_{Staff}$-*Flight-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*OclAny*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Flight-OclAny*)
lemma cp-$OclIsKindOf_{Staff}$-*Staff-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Client)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Client*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Staff-Client*)
lemma cp-$OclIsKindOf_{Staff}$-*Person-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Client)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Client*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Person-Client*)
lemma cp-$OclIsKindOf_{Staff}$-*OclAny-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Client)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Client*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*OclAny-Client*)
lemma cp-$OclIsKindOf_{Staff}$-*Client-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Client)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Client*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Client-Client*)
lemma cp-$OclIsKindOf_{Staff}$-*Reservation-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Client)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Client*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Reservation-Client*)
lemma cp-$OclIsKindOf_{Staff}$-*Flight-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Client)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Client*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Flight-Client*)
lemma cp-$OclIsKindOf_{Staff}$-*Staff-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Reservation*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Staff-Reservation*)
lemma cp-$OclIsKindOf_{Staff}$-*Person-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Reservation*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Person-Reservation*)
lemma cp-$OclIsKindOf_{Staff}$-*OclAny-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Reservation*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*OclAny-Reservation*)
lemma cp-$OclIsKindOf_{Staff}$-*Client-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Reservation*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Client-Reservation*)
lemma cp-$OclIsKindOf_{Staff}$-*Reservation-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Reservation*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Reservation-Reservation*)
lemma cp-$OclIsKindOf_{Staff}$-*Flight-Reservation* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Reservation*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Flight-Reservation*)
lemma cp-$OclIsKindOf_{Staff}$-*Staff-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Flight*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Staff-Flight*)
lemma cp-$OclIsKindOf_{Staff}$-*Person-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Flight)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Flight*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Person-Flight*)
lemma cp-$OclIsKindOf_{Staff}$-*OclAny-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Flight)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Flight*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*OclAny-Flight*)
lemma cp-$OclIsKindOf_{Staff}$-*Client-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Flight)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Flight*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Client-Flight*)
lemma cp-$OclIsKindOf_{Staff}$-*Reservation-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Flight*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Reservation-Flight*)
lemma cp-$OclIsKindOf_{Staff}$-*Flight-Flight* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclIsKindOf(Staff)))))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Flight*, *simp only*: cp-$OclIsTypeOf_{Staff}$-*Flight-Flight*)
lemma cp-$OclIsKindOf_{Person}$-*Person-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Person)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Person*)
apply((*rule cpI2*[where f = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: cp-$OclIsTypeOf_{Person}$-*Person-Person*)
by(*simp only*: cp-$OclIsKindOf_{Staff}$-*Person-Person*, *simp only*: cp-$OclIsKindOf_{Client}$-*Person-Person*)
lemma cp-$OclIsKindOf_{Person}$-*OclAny-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Person)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Person*)
apply((*rule cpI2*[where f = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: cp-$OclIsTypeOf_{Person}$-*OclAny-Person*)
by(*simp only*: cp-$OclIsKindOf_{Staff}$-*OclAny-Person*, *simp only*: cp-$OclIsKindOf_{Client}$-*OclAny-Person*)
lemma cp-$OclIsKindOf_{Person}$-*Client-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Person)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Person*)
apply((*rule cpI2*[where f = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: cp-$OclIsTypeOf_{Person}$-*Client-Person*)
by(*simp only*: cp-$OclIsKindOf_{Staff}$-*Client-Person*, *simp only*: cp-$OclIsKindOf_{Client}$-*Client-Person*)
lemma cp-$OclIsKindOf_{Person}$-*Staff-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Person)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Person*)
apply((*rule cpI2*[where f = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: cp-$OclIsTypeOf_{Person}$-*Staff-Person*)
by(*simp only*: cp-$OclIsKindOf_{Staff}$-*Staff-Person*, *simp only*: cp-$OclIsKindOf_{Client}$-*Staff-Person*)
lemma cp-$OclIsKindOf_{Person}$-*Reservation-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Person*)
apply((*rule cpI2*[where f = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: cp-$OclIsTypeOf_{Person}$-*Reservation-Person*)
by(*simp only*: cp-$OclIsKindOf_{Staff}$-*Reservation-Person*, *simp only*: cp-$OclIsKindOf_{Client}$-*Reservation-Person*)
lemma cp-$OclIsKindOf_{Person}$-*Flight-Person* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Person)\ .oclIsKindOf(Person)))))$

apply(*simp only*: $OclIsKindOf_{Person}$-*Person*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Flight-Person*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Flight-Person*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Flight-Person*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Person-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Person)))::OclAny)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*OclAny*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Person-OclAny*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Person-OclAny*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Person-OclAny*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*OclAny-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::OclAny)))::OclAny)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*OclAny*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*OclAny-OclAny*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*OclAny-OclAny*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*OclAny-OclAny*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Client-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Client)))::OclAny)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*OclAny*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Client-OclAny*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Client-OclAny*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Client-OclAny*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Staff-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Staff)))::OclAny)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*OclAny*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Staff-OclAny*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Staff-OclAny*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Staff-OclAny*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Reservation-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Reservation)))::OclAny)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*OclAny*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Reservation-OclAny*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Reservation-OclAny*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Reservation-OclAny*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Flight-OclAny* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Flight)))::OclAny)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*OclAny*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Flight-OclAny*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Flight-OclAny*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Flight-OclAny*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Person-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Person)))::Client)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Client*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Person-Client*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Person-Client*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Person-Client*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*OclAny-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::OclAny)))::Client)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Client*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*OclAny-Client*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*OclAny-Client*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*OclAny-Client*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Client-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Client)))::Client)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Client*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Client-Client*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Client-Client*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Client-Client*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Staff-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Staff)))::Client)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Client*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Staff-Client*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Staff-Client*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Staff-Client*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Reservation-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Reservation)))::Client)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Client*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Reservation-Client*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Reservation-Client*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Reservation-Client*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Flight-Client* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Flight)))::Client)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Client*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Flight-Client*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Flight-Client*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Flight-Client*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*Person-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::Person)))::Staff)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Staff*)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}$-*Person-Staff*)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}$-*Person-Staff*, *simp only*: $cp\text{-}OclIsKindOf_{Client}$-*Person-Staff*)
lemma $cp\text{-}OclIsKindOf_{Person}$-*OclAny-Staff* : $(cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::OclAny)))::Staff)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}$-*Staff*)

apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-OclAny-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-OclAny-Staff*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-OclAny-Staff*)
lemma *cp-OclIsKindOf*$_{Person}$*-Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Staff*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Client-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Client-Staff*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Client-Staff*)
lemma *cp-OclIsKindOf*$_{Person}$*-Staff-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Staff*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Staff-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Staff-Staff*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Staff-Staff*)
lemma *cp-OclIsKindOf*$_{Person}$*-Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Staff*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Reservation-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Reservation-Staff*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Reservation-Staff*)
lemma *cp-OclIsKindOf*$_{Person}$*-Flight-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Staff*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Flight-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Flight-Staff*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Flight-Staff*)
lemma *cp-OclIsKindOf*$_{Person}$*-Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Person-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Person-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Person-Reservation*)
lemma *cp-OclIsKindOf*$_{Person}$*-OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-OclAny-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-OclAny-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-OclAny-Reservation*)
lemma *cp-OclIsKindOf*$_{Person}$*-Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Client-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Client-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Client-Reservation*)
lemma *cp-OclIsKindOf*$_{Person}$*-Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Staff-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Staff-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Staff-Reservation*)
lemma *cp-OclIsKindOf*$_{Person}$*-Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Reservation-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Reservation-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Reservation-Reservation*)
lemma *cp-OclIsKindOf*$_{Person}$*-Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Flight-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Flight-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Flight-Reservation*)
lemma *cp-OclIsKindOf*$_{Person}$*-Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-Person-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-Person-Flight*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-Person-Flight*)
lemma *cp-OclIsKindOf*$_{Person}$*-OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{Person}$*-OclAny-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Staff}$*-OclAny-Flight*, *simp only*: *cp-OclIsKindOf*$_{Client}$*-OclAny-Flight*)
lemma *cp-OclIsKindOf*$_{Person}$*-Client-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Flight*) .*oclIsKindOf*(*Person*)))))
apply(*simp only*: *OclIsKindOf*$_{Person}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)

apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}\text{-}Client\text{-}Flight$)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}\text{-}Client\text{-}Flight$, *simp only*: $cp\text{-}OclIsKindOf_{Client}\text{-}Client\text{-}Flight$)
lemma $cp\text{-}OclIsKindOf_{Person}\text{-}Staff\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Flight)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}\text{-}Flight$)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}\text{-}Staff\text{-}Flight$)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}\text{-}Staff\text{-}Flight$, *simp only*: $cp\text{-}OclIsKindOf_{Client}\text{-}Staff\text{-}Flight$)
lemma $cp\text{-}OclIsKindOf_{Person}\text{-}Reservation\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Flight)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}\text{-}Flight$)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}\text{-}Reservation\text{-}Flight$)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}\text{-}Reservation\text{-}Flight$, *simp only*: $cp\text{-}OclIsKindOf_{Client}\text{-}Reservation\text{-}Flight$)
lemma $cp\text{-}OclIsKindOf_{Person}\text{-}Flight\text{-}Flight : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Flight)\ .oclIsKindOf(Person)))))$
apply(*simp only*: $OclIsKindOf_{Person}\text{-}Flight$)
apply((*rule cpI2*[where $f = op\ or$], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: $cp\text{-}OclIsTypeOf_{Person}\text{-}Flight\text{-}Flight$)
by(*simp only*: $cp\text{-}OclIsKindOf_{Staff}\text{-}Flight\text{-}Flight$, *simp only*: $cp\text{-}OclIsKindOf_{Client}\text{-}Flight\text{-}Flight$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Reservation)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Reservation$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Reservation$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Reservation)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Reservation$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Reservation$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Reservation)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Reservation$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Reservation$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Reservation)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Reservation$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Reservation$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Reservation)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Reservation$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Reservation$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}Reservation : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Reservation)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Reservation$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Reservation$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot OclAny)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}OclAny$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}OclAny$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot OclAny)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}OclAny$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}OclAny$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot OclAny)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}OclAny$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}OclAny$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot OclAny)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}OclAny$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}OclAny$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot OclAny)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}OclAny$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}OclAny$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}OclAny : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot OclAny)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}OclAny$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}OclAny$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Staff)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Staff$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Staff$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot OclAny)))::\cdot Staff)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Staff$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}OclAny\text{-}Staff$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Staff)))::\cdot Staff)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Staff$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Staff\text{-}Staff$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Person)))::\cdot Staff)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Staff$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Person\text{-}Staff$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Client)))::\cdot Staff)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Staff$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Client\text{-}Staff$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}Staff : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Flight)))::\cdot Staff)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Staff$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Flight\text{-}Staff$)
lemma $cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Person : (cp\ (p)) \Longrightarrow (cp\ ((\lambda x.\ (((p\ ((x::\cdot Reservation)))::\cdot Person)\ .oclIsKindOf(Reservation)))))$
by(*simp only*: $OclIsKindOf_{Reservation}\text{-}Person$, *simp only*: $cp\text{-}OclIsTypeOf_{Reservation}\text{-}Reservation\text{-}Person$)

lemma *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Person*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-OclAny-Person*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Person*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Staff-Person*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Person-Person*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Person*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Client-Person*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Person*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Person*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Flight-Person*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Client*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Reservation-Client*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-OclAny-Client*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Staff-Client*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Client*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Person-Client*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Client-Client*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Client*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Flight-Client*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Reservation-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Flight*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Flight*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Reservation-Flight*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Flight*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-OclAny-Flight*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Staff-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Flight*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Flight*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Staff-Flight*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Flight*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Person-Flight*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Client-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Flight*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Flight*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Client-Flight*)
lemma *cp-OclIsKindOf*$_{Reservation}$*-Flight-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Flight*) .*oclIsKindOf*(*Reservation*)))))
by(*simp only*: *OclIsKindOf*$_{Reservation}$*-Flight*, *simp only*: *cp-OclIsTypeOf*$_{Reservation}$*-Flight-Flight*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-OclAny-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*OclAny*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-OclAny*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-OclAny-OclAny*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-OclAny-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-OclAny-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-OclAny-OclAny*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Flight-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*OclAny*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-OclAny*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Flight-OclAny*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Flight-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Flight-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Flight-OclAny*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Client-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*OclAny*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-OclAny*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Client-OclAny*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Client-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Client-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Client-OclAny*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Staff-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*OclAny*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-OclAny*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Staff-OclAny*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Staff-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Staff-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Staff-OclAny*)

lemma *cp-OclIsKindOf*$_{OclAny}$*-Person-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*OclAny*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-OclAny*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Person-OclAny*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Person-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Person-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Person-OclAny*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Reservation-OclAny* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*OclAny*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-OclAny*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Reservation-OclAny*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Reservation-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Reservation-OclAny*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Reservation-OclAny*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-OclAny-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Flight*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-OclAny-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Flight*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-OclAny-Flight*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-OclAny-Flight*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Flight-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Flight*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Flight-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Flight-Flight*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Flight-Flight*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Flight-Flight*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Client-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Flight*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Client-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Client-Flight*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Client-Flight*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Client-Flight*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Staff-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Flight*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Staff-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Staff-Flight*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Staff-Flight*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Staff-Flight*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Person-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Flight*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Person-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Person-Flight*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Person-Flight*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Person-Flight*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Reservation-Flight* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Flight*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Flight*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Reservation-Flight*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Reservation-Flight*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Reservation-Flight*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Reservation-Flight*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-OclAny-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Client*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Client*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-OclAny-Client*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Client*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-OclAny-Client*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-OclAny-Client*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Flight-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Client*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Client*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Flight-Client*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Flight-Client*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Flight-Client*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Flight-Client*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Client-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Client*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Client*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Client-Client*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Client-Client*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Client-Client*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Client-Client*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Staff-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Client*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Client*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Staff-Client*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Staff-Client*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Staff-Client*, *simp only*:

cp-OclIsKindOf$_{Flight}$*-Staff-Client*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Person-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Person*)))::*Client*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Client*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Person-Client*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Person-Client*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Person-Client*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Person-Client*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Reservation-Client* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Reservation*)))::*Client*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Client*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Reservation-Client*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Reservation-Client*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Reservation-Client*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Reservation-Client*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-OclAny-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*OclAny*)))::*Staff*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-OclAny-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Staff*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-OclAny-Staff*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-OclAny-Staff*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Flight-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Flight*)))::*Staff*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Flight-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Flight-Staff*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Flight-Staff*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Flight-Staff*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Client-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Client*)))::*Staff*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Client-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Client-Staff*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Client-Staff*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Client-Staff*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Staff-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Staff*)))::*Staff*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Staff-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Staff-Staff*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Staff-Staff*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Staff-Staff*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Person-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Person*)))::*Staff*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Person-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Person-Staff*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Person-Staff*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Person-Staff*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Reservation-Staff* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Reservation*)))::*Staff*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Staff*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Reservation-Staff*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Reservation-Staff*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Reservation-Staff*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Reservation-Staff*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-OclAny-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*OclAny*)))::*Person*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Person*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-OclAny-Person*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Person*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-OclAny-Person*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-OclAny-Person*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Flight-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Flight*)))::*Person*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Person*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Flight-Person*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Flight-Person*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Flight-Person*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Flight-Person*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Client-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Client*)))::*Person*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Person*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Client-Person*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Client-Person*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Client-Person*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Client-Person*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Staff-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::*Staff*)))::*Person*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Person*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Staff-Person*)

by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Staff-Person*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Staff-Person*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Staff-Person*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Person-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Person*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Person*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Person-Person*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Person-Person*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Person-Person*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Person-Person*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Reservation-Person* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Person*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Person*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Reservation-Person*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Reservation-Person*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Reservation-Person*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Reservation-Person*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-OclAny-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*OclAny*)))::·*Reservation*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-OclAny-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-OclAny-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-OclAny-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-OclAny-Reservation*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Flight-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Flight*)))::·*Reservation*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Flight-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Flight-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Flight-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Flight-Reservation*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Client-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Client*)))::·*Reservation*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Client-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Client-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Client-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Client-Reservation*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Staff-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Staff*)))::·*Reservation*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Staff-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Staff-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Staff-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Staff-Reservation*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Person-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Person*)))::·*Reservation*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Person-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Person-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Person-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Person-Reservation*)
lemma *cp-OclIsKindOf*$_{OclAny}$*-Reservation-Reservation* : (*cp* (*p*)) ⟹ (*cp* ((λ*x*. (((*p* ((*x*::·*Reservation*)))::·*Reservation*) .*oclIsKindOf*(*OclAny*)))))
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-Reservation*)
apply((*rule cpI2*[where *f* = *op or*], (*rule allI*)+, *rule cp-OclOr*)+)
apply(*simp only*: *cp-OclIsTypeOf*$_{OclAny}$*-Reservation-Reservation*)
by(*simp only*: *cp-OclIsKindOf*$_{Reservation}$*-Reservation-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Person}$*-Reservation-Reservation*, *simp only*: *cp-OclIsKindOf*$_{Flight}$*-Reservation-Reservation*)

lemmas[*simp*,*code-unfold*] = *cp-OclIsKindOf*$_{Client}$*-Client-Client*
cp-OclIsKindOf$_{Client}$*-Reservation-Client*
cp-OclIsKindOf$_{Client}$*-OclAny-Client*
cp-OclIsKindOf$_{Client}$*-Person-Client*
cp-OclIsKindOf$_{Client}$*-Staff-Client*
cp-OclIsKindOf$_{Client}$*-Flight-Client*
cp-OclIsKindOf$_{Client}$*-Client-Reservation*
cp-OclIsKindOf$_{Client}$*-Reservation-Reservation*
cp-OclIsKindOf$_{Client}$*-OclAny-Reservation*
cp-OclIsKindOf$_{Client}$*-Person-Reservation*
cp-OclIsKindOf$_{Client}$*-Staff-Reservation*
cp-OclIsKindOf$_{Client}$*-Flight-Reservation*
cp-OclIsKindOf$_{Client}$*-Client-OclAny*
cp-OclIsKindOf$_{Client}$*-Reservation-OclAny*

$cp\text{-}OclIsKindOf_{Client}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Client}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Client}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Client}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Client}\text{-}Client\text{-}Person$
$cp\text{-}OclIsKindOf_{Client}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsKindOf_{Client}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsKindOf_{Client}\text{-}Person\text{-}Person$
$cp\text{-}OclIsKindOf_{Client}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsKindOf_{Client}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsKindOf_{Client}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsKindOf_{Client}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsKindOf_{Client}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsKindOf_{Client}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsKindOf_{Client}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsKindOf_{Client}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsKindOf_{Client}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsKindOf_{Client}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsKindOf_{Client}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsKindOf_{Client}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsKindOf_{Client}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsKindOf_{Client}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}Client$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}Client$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}Person$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}Person$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsKindOf_{Reservation}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Client\text{-}Client$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Person\text{-}Client$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}OclAny\text{-}OclAny$

$cp\text{-}OclIsKindOf_{OclAny}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Client\text{-}Person$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Person\text{-}Person$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsKindOf_{OclAny}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsKindOf_{Person}\text{-}Client\text{-}Client$
$cp\text{-}OclIsKindOf_{Person}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsKindOf_{Person}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsKindOf_{Person}\text{-}Person\text{-}Client$
$cp\text{-}OclIsKindOf_{Person}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsKindOf_{Person}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsKindOf_{Person}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Person}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Person}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Person}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Person}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Person}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Person}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Person}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Person}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Person}\text{-}Person\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Person}\text{-}Staff\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Person}\text{-}Flight\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Person}\text{-}Client\text{-}Person$
$cp\text{-}OclIsKindOf_{Person}\text{-}Reservation\text{-}Person$
$cp\text{-}OclIsKindOf_{Person}\text{-}OclAny\text{-}Person$
$cp\text{-}OclIsKindOf_{Person}\text{-}Person\text{-}Person$
$cp\text{-}OclIsKindOf_{Person}\text{-}Staff\text{-}Person$
$cp\text{-}OclIsKindOf_{Person}\text{-}Flight\text{-}Person$
$cp\text{-}OclIsKindOf_{Person}\text{-}Client\text{-}Staff$
$cp\text{-}OclIsKindOf_{Person}\text{-}Reservation\text{-}Staff$
$cp\text{-}OclIsKindOf_{Person}\text{-}OclAny\text{-}Staff$
$cp\text{-}OclIsKindOf_{Person}\text{-}Person\text{-}Staff$
$cp\text{-}OclIsKindOf_{Person}\text{-}Staff\text{-}Staff$
$cp\text{-}OclIsKindOf_{Person}\text{-}Flight\text{-}Staff$
$cp\text{-}OclIsKindOf_{Person}\text{-}Client\text{-}Flight$
$cp\text{-}OclIsKindOf_{Person}\text{-}Reservation\text{-}Flight$
$cp\text{-}OclIsKindOf_{Person}\text{-}OclAny\text{-}Flight$
$cp\text{-}OclIsKindOf_{Person}\text{-}Person\text{-}Flight$
$cp\text{-}OclIsKindOf_{Person}\text{-}Staff\text{-}Flight$
$cp\text{-}OclIsKindOf_{Person}\text{-}Flight\text{-}Flight$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Client\text{-}Client$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Reservation\text{-}Client$
$cp\text{-}OclIsKindOf_{Staff}\text{-}OclAny\text{-}Client$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Person\text{-}Client$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Staff\text{-}Client$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Flight\text{-}Client$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Client\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Reservation\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Staff}\text{-}OclAny\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Person\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Staff\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Flight\text{-}Reservation$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Client\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Reservation\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Staff}\text{-}OclAny\text{-}OclAny$
$cp\text{-}OclIsKindOf_{Staff}\text{-}Person\text{-}OclAny$

cp-OclIsKindOf$_{Staff}$*-Staff-OclAny*
cp-OclIsKindOf$_{Staff}$*-Flight-OclAny*
cp-OclIsKindOf$_{Staff}$*-Client-Person*
cp-OclIsKindOf$_{Staff}$*-Reservation-Person*
cp-OclIsKindOf$_{Staff}$*-OclAny-Person*
cp-OclIsKindOf$_{Staff}$*-Person-Person*
cp-OclIsKindOf$_{Staff}$*-Staff-Person*
cp-OclIsKindOf$_{Staff}$*-Flight-Person*
cp-OclIsKindOf$_{Staff}$*-Client-Staff*
cp-OclIsKindOf$_{Staff}$*-Reservation-Staff*
cp-OclIsKindOf$_{Staff}$*-OclAny-Staff*
cp-OclIsKindOf$_{Staff}$*-Person-Staff*
cp-OclIsKindOf$_{Staff}$*-Staff-Staff*
cp-OclIsKindOf$_{Staff}$*-Flight-Staff*
cp-OclIsKindOf$_{Staff}$*-Client-Flight*
cp-OclIsKindOf$_{Staff}$*-Reservation-Flight*
cp-OclIsKindOf$_{Staff}$*-OclAny-Flight*
cp-OclIsKindOf$_{Staff}$*-Person-Flight*
cp-OclIsKindOf$_{Staff}$*-Staff-Flight*
cp-OclIsKindOf$_{Staff}$*-Flight-Flight*
cp-OclIsKindOf$_{Flight}$*-Client-Client*
cp-OclIsKindOf$_{Flight}$*-Reservation-Client*
cp-OclIsKindOf$_{Flight}$*-OclAny-Client*
cp-OclIsKindOf$_{Flight}$*-Person-Client*
cp-OclIsKindOf$_{Flight}$*-Staff-Client*
cp-OclIsKindOf$_{Flight}$*-Flight-Client*
cp-OclIsKindOf$_{Flight}$*-Client-Reservation*
cp-OclIsKindOf$_{Flight}$*-Reservation-Reservation*
cp-OclIsKindOf$_{Flight}$*-OclAny-Reservation*
cp-OclIsKindOf$_{Flight}$*-Person-Reservation*
cp-OclIsKindOf$_{Flight}$*-Staff-Reservation*
cp-OclIsKindOf$_{Flight}$*-Flight-Reservation*
cp-OclIsKindOf$_{Flight}$*-Client-OclAny*
cp-OclIsKindOf$_{Flight}$*-Reservation-OclAny*
cp-OclIsKindOf$_{Flight}$*-OclAny-OclAny*
cp-OclIsKindOf$_{Flight}$*-Person-OclAny*
cp-OclIsKindOf$_{Flight}$*-Staff-OclAny*
cp-OclIsKindOf$_{Flight}$*-Flight-OclAny*
cp-OclIsKindOf$_{Flight}$*-Client-Person*
cp-OclIsKindOf$_{Flight}$*-Reservation-Person*
cp-OclIsKindOf$_{Flight}$*-OclAny-Person*
cp-OclIsKindOf$_{Flight}$*-Person-Person*
cp-OclIsKindOf$_{Flight}$*-Staff-Person*
cp-OclIsKindOf$_{Flight}$*-Flight-Person*
cp-OclIsKindOf$_{Flight}$*-Client-Staff*
cp-OclIsKindOf$_{Flight}$*-Reservation-Staff*
cp-OclIsKindOf$_{Flight}$*-OclAny-Staff*
cp-OclIsKindOf$_{Flight}$*-Person-Staff*
cp-OclIsKindOf$_{Flight}$*-Staff-Staff*
cp-OclIsKindOf$_{Flight}$*-Flight-Staff*
cp-OclIsKindOf$_{Flight}$*-Client-Flight*
cp-OclIsKindOf$_{Flight}$*-Reservation-Flight*
cp-OclIsKindOf$_{Flight}$*-OclAny-Flight*
cp-OclIsKindOf$_{Flight}$*-Person-Flight*
cp-OclIsKindOf$_{Flight}$*-Staff-Flight*
cp-OclIsKindOf$_{Flight}$*-Flight-Flight*

Execution with Invalid or Null as Argument

lemma *OclIsKindOf*$_{Flight}$*-Flight-invalid* : $((invalid::\cdot Flight)\ .oclIsKindOf(Flight)) = invalid$
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Flight OclIsTypeOf*$_{Flight}$*-Flight-invalid*)
lemma *OclIsKindOf*$_{Flight}$*-Flight-null* : $((null::\cdot Flight)\ .oclIsKindOf(Flight)) = true$
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Flight OclIsTypeOf*$_{Flight}$*-Flight-null*)
lemma *OclIsKindOf*$_{Flight}$*-OclAny-invalid* : $((invalid::\cdot OclAny)\ .oclIsKindOf(Flight)) = invalid$
by(*simp only*: *OclIsKindOf*$_{Flight}$*-OclAny OclIsTypeOf*$_{Flight}$*-OclAny-invalid*)
lemma *OclIsKindOf*$_{Flight}$*-OclAny-null* : $((null::\cdot OclAny)\ .oclIsKindOf(Flight)) = true$
by(*simp only*: *OclIsKindOf*$_{Flight}$*-OclAny OclIsTypeOf*$_{Flight}$*-OclAny-null*)
lemma *OclIsKindOf*$_{Flight}$*-Staff-invalid* : $((invalid::\cdot Staff)\ .oclIsKindOf(Flight)) = invalid$
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Staff OclIsTypeOf*$_{Flight}$*-Staff-invalid*)
lemma *OclIsKindOf*$_{Flight}$*-Staff-null* : $((null::\cdot Staff)\ .oclIsKindOf(Flight)) = true$
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Staff OclIsTypeOf*$_{Flight}$*-Staff-null*)
lemma *OclIsKindOf*$_{Flight}$*-Person-invalid* : $((invalid::\cdot Person)\ .oclIsKindOf(Flight)) = invalid$
by(*simp only*: *OclIsKindOf*$_{Flight}$*-Person OclIsTypeOf*$_{Flight}$*-Person-invalid*)

lemma $OclIsKindOf_{Flight}$-Person-null : $((null::\cdot Person)\ .oclIsKindOf(Flight)) = true$
by(*simp only*: $OclIsKindOf_{Flight}$-Person $OclIsTypeOf_{Flight}$-Person-null)
lemma $OclIsKindOf_{Flight}$-Client-invalid : $((invalid::\cdot Client)\ .oclIsKindOf(Flight)) = invalid$
by(*simp only*: $OclIsKindOf_{Flight}$-Client $OclIsTypeOf_{Flight}$-Client-invalid)
lemma $OclIsKindOf_{Flight}$-Client-null : $((null::\cdot Client)\ .oclIsKindOf(Flight)) = true$
by(*simp only*: $OclIsKindOf_{Flight}$-Client $OclIsTypeOf_{Flight}$-Client-null)
lemma $OclIsKindOf_{Flight}$-Reservation-invalid : $((invalid::\cdot Reservation)\ .oclIsKindOf(Flight)) = invalid$
by(*simp only*: $OclIsKindOf_{Flight}$-Reservation $OclIsTypeOf_{Flight}$-Reservation-invalid)
lemma $OclIsKindOf_{Flight}$-Reservation-null : $((null::\cdot Reservation)\ .oclIsKindOf(Flight)) = true$
by(*simp only*: $OclIsKindOf_{Flight}$-Reservation $OclIsTypeOf_{Flight}$-Reservation-null)
lemma $OclIsKindOf_{Client}$-Client-invalid : $((invalid::\cdot Client)\ .oclIsKindOf(Client)) = invalid$
by(*simp only*: $OclIsKindOf_{Client}$-Client $OclIsTypeOf_{Client}$-Client-invalid)
lemma $OclIsKindOf_{Client}$-Client-null : $((null::\cdot Client)\ .oclIsKindOf(Client)) = true$
by(*simp only*: $OclIsKindOf_{Client}$-Client $OclIsTypeOf_{Client}$-Client-null)
lemma $OclIsKindOf_{Client}$-Person-invalid : $((invalid::\cdot Person)\ .oclIsKindOf(Client)) = invalid$
by(*simp only*: $OclIsKindOf_{Client}$-Person $OclIsTypeOf_{Client}$-Person-invalid)
lemma $OclIsKindOf_{Client}$-Person-null : $((null::\cdot Person)\ .oclIsKindOf(Client)) = true$
by(*simp only*: $OclIsKindOf_{Client}$-Person $OclIsTypeOf_{Client}$-Person-null)
lemma $OclIsKindOf_{Client}$-OclAny-invalid : $((invalid::\cdot OclAny)\ .oclIsKindOf(Client)) = invalid$
by(*simp only*: $OclIsKindOf_{Client}$-OclAny $OclIsTypeOf_{Client}$-OclAny-invalid)
lemma $OclIsKindOf_{Client}$-OclAny-null : $((null::\cdot OclAny)\ .oclIsKindOf(Client)) = true$
by(*simp only*: $OclIsKindOf_{Client}$-OclAny $OclIsTypeOf_{Client}$-OclAny-null)
lemma $OclIsKindOf_{Client}$-Staff-invalid : $((invalid::\cdot Staff)\ .oclIsKindOf(Client)) = invalid$
by(*simp only*: $OclIsKindOf_{Client}$-Staff $OclIsTypeOf_{Client}$-Staff-invalid)
lemma $OclIsKindOf_{Client}$-Staff-null : $((null::\cdot Staff)\ .oclIsKindOf(Client)) = true$
by(*simp only*: $OclIsKindOf_{Client}$-Staff $OclIsTypeOf_{Client}$-Staff-null)
lemma $OclIsKindOf_{Client}$-Reservation-invalid : $((invalid::\cdot Reservation)\ .oclIsKindOf(Client)) = invalid$
by(*simp only*: $OclIsKindOf_{Client}$-Reservation $OclIsTypeOf_{Client}$-Reservation-invalid)
lemma $OclIsKindOf_{Client}$-Reservation-null : $((null::\cdot Reservation)\ .oclIsKindOf(Client)) = true$
by(*simp only*: $OclIsKindOf_{Client}$-Reservation $OclIsTypeOf_{Client}$-Reservation-null)
lemma $OclIsKindOf_{Client}$-Flight-invalid : $((invalid::\cdot Flight)\ .oclIsKindOf(Client)) = invalid$
by(*simp only*: $OclIsKindOf_{Client}$-Flight $OclIsTypeOf_{Client}$-Flight-invalid)
lemma $OclIsKindOf_{Client}$-Flight-null : $((null::\cdot Flight)\ .oclIsKindOf(Client)) = true$
by(*simp only*: $OclIsKindOf_{Client}$-Flight $OclIsTypeOf_{Client}$-Flight-null)
lemma $OclIsKindOf_{Staff}$-Staff-invalid : $((invalid::\cdot Staff)\ .oclIsKindOf(Staff)) = invalid$
by(*simp only*: $OclIsKindOf_{Staff}$-Staff $OclIsTypeOf_{Staff}$-Staff-invalid)
lemma $OclIsKindOf_{Staff}$-Staff-null : $((null::\cdot Staff)\ .oclIsKindOf(Staff)) = true$
by(*simp only*: $OclIsKindOf_{Staff}$-Staff $OclIsTypeOf_{Staff}$-Staff-null)
lemma $OclIsKindOf_{Staff}$-Person-invalid : $((invalid::\cdot Person)\ .oclIsKindOf(Staff)) = invalid$
by(*simp only*: $OclIsKindOf_{Staff}$-Person $OclIsTypeOf_{Staff}$-Person-invalid)
lemma $OclIsKindOf_{Staff}$-Person-null : $((null::\cdot Person)\ .oclIsKindOf(Staff)) = true$
by(*simp only*: $OclIsKindOf_{Staff}$-Person $OclIsTypeOf_{Staff}$-Person-null)
lemma $OclIsKindOf_{Staff}$-OclAny-invalid : $((invalid::\cdot OclAny)\ .oclIsKindOf(Staff)) = invalid$
by(*simp only*: $OclIsKindOf_{Staff}$-OclAny $OclIsTypeOf_{Staff}$-OclAny-invalid)
lemma $OclIsKindOf_{Staff}$-OclAny-null : $((null::\cdot OclAny)\ .oclIsKindOf(Staff)) = true$
by(*simp only*: $OclIsKindOf_{Staff}$-OclAny $OclIsTypeOf_{Staff}$-OclAny-null)
lemma $OclIsKindOf_{Staff}$-Client-invalid : $((invalid::\cdot Client)\ .oclIsKindOf(Staff)) = invalid$
by(*simp only*: $OclIsKindOf_{Staff}$-Client $OclIsTypeOf_{Staff}$-Client-invalid)
lemma $OclIsKindOf_{Staff}$-Client-null : $((null::\cdot Client)\ .oclIsKindOf(Staff)) = true$
by(*simp only*: $OclIsKindOf_{Staff}$-Client $OclIsTypeOf_{Staff}$-Client-null)
lemma $OclIsKindOf_{Staff}$-Reservation-invalid : $((invalid::\cdot Reservation)\ .oclIsKindOf(Staff)) = invalid$
by(*simp only*: $OclIsKindOf_{Staff}$-Reservation $OclIsTypeOf_{Staff}$-Reservation-invalid)
lemma $OclIsKindOf_{Staff}$-Reservation-null : $((null::\cdot Reservation)\ .oclIsKindOf(Staff)) = true$
by(*simp only*: $OclIsKindOf_{Staff}$-Reservation $OclIsTypeOf_{Staff}$-Reservation-null)
lemma $OclIsKindOf_{Staff}$-Flight-invalid : $((invalid::\cdot Flight)\ .oclIsKindOf(Staff)) = invalid$
by(*simp only*: $OclIsKindOf_{Staff}$-Flight $OclIsTypeOf_{Staff}$-Flight-invalid)
lemma $OclIsKindOf_{Staff}$-Flight-null : $((null::\cdot Flight)\ .oclIsKindOf(Staff)) = true$
by(*simp only*: $OclIsKindOf_{Staff}$-Flight $OclIsTypeOf_{Staff}$-Flight-null)
lemma $OclIsKindOf_{Person}$-Person-invalid : $((invalid::\cdot Person)\ .oclIsKindOf(Person)) = invalid$
by(*simp only*: $OclIsKindOf_{Person}$-Person $OclIsTypeOf_{Person}$-Person-invalid $OclIsKindOf_{Staff}$-Person-invalid $OclIsKindOf_{Client}$-Person-invalid, *simp*)
lemma $OclIsKindOf_{Person}$-Person-null : $((null::\cdot Person)\ .oclIsKindOf(Person)) = true$
by(*simp only*: $OclIsKindOf_{Person}$-Person $OclIsTypeOf_{Person}$-Person-null $OclIsKindOf_{Staff}$-Person-null $OclIsKindOf_{Client}$-Person-null, *simp*)
lemma $OclIsKindOf_{Person}$-OclAny-invalid : $((invalid::\cdot OclAny)\ .oclIsKindOf(Person)) = invalid$
by(*simp only*: $OclIsKindOf_{Person}$-OclAny $OclIsTypeOf_{Person}$-OclAny-invalid $OclIsKindOf_{Staff}$-OclAny-invalid $OclIsKindOf_{Client}$-OclAny-invalid, *simp*)
lemma $OclIsKindOf_{Person}$-OclAny-null : $((null::\cdot OclAny)\ .oclIsKindOf(Person)) = true$
by(*simp only*: $OclIsKindOf_{Person}$-OclAny $OclIsTypeOf_{Person}$-OclAny-null $OclIsKindOf_{Staff}$-OclAny-null $OclIsKindOf_{Client}$-OclAny-null, *simp*)
lemma $OclIsKindOf_{Person}$-Client-invalid : $((invalid::\cdot Client)\ .oclIsKindOf(Person)) = invalid$
by(*simp only*: $OclIsKindOf_{Person}$-Client $OclIsTypeOf_{Person}$-Client-invalid $OclIsKindOf_{Staff}$-Client-invalid $OclIsKindOf_{Client}$-Client-invalid, *simp*)

lemma $OclIsKindOf_{Person}$-Client-null : ((null::·Client) .oclIsKindOf(Person)) = true
by(simp only: $OclIsKindOf_{Person}$-Client $OclIsTypeOf_{Person}$-Client-null $OclIsKindOf_{Staff}$-Client-null $OclIsKindOf_{Client}$-Client-null, simp)
lemma $OclIsKindOf_{Person}$-Staff-invalid : ((invalid::·Staff) .oclIsKindOf(Person)) = invalid
by(simp only: $OclIsKindOf_{Person}$-Staff $OclIsTypeOf_{Person}$-Staff-invalid $OclIsKindOf_{Staff}$-Staff-invalid $OclIsKindOf_{Client}$-Staff-invalid, simp)
lemma $OclIsKindOf_{Person}$-Staff-null : ((null::·Staff) .oclIsKindOf(Person)) = true
by(simp only: $OclIsKindOf_{Person}$-Staff $OclIsTypeOf_{Person}$-Staff-null $OclIsKindOf_{Staff}$-Staff-null $OclIsKindOf_{Client}$-Staff-null, simp)
lemma $OclIsKindOf_{Person}$-Reservation-invalid : ((invalid::·Reservation) .oclIsKindOf(Person)) = invalid
by(simp only: $OclIsKindOf_{Person}$-Reservation $OclIsTypeOf_{Person}$-Reservation-invalid $OclIsKindOf_{Staff}$-Reservation-invalid $OclIsKindOf_{Client}$-Reservation-invalid, simp)
lemma $OclIsKindOf_{Person}$-Reservation-null : ((null::·Reservation) .oclIsKindOf(Person)) = true
by(simp only: $OclIsKindOf_{Person}$-Reservation $OclIsTypeOf_{Person}$-Reservation-null $OclIsKindOf_{Staff}$-Reservation-null $OclIsKindOf_{Client}$-Reservation-null, simp)
lemma $OclIsKindOf_{Person}$-Flight-invalid : ((invalid::·Flight) .oclIsKindOf(Person)) = invalid
by(simp only: $OclIsKindOf_{Person}$-Flight $OclIsTypeOf_{Person}$-Flight-invalid $OclIsKindOf_{Staff}$-Flight-invalid $OclIsKindOf_{Client}$-Flight-invalid, simp)
lemma $OclIsKindOf_{Person}$-Flight-null : ((null::·Flight) .oclIsKindOf(Person)) = true
by(simp only: $OclIsKindOf_{Person}$-Flight $OclIsTypeOf_{Person}$-Flight-null $OclIsKindOf_{Staff}$-Flight-null $OclIsKindOf_{Client}$-Flight-null, simp)
lemma $OclIsKindOf_{Reservation}$-Reservation-invalid : ((invalid::·Reservation) .oclIsKindOf(Reservation)) = invalid
by(simp only: $OclIsKindOf_{Reservation}$-Reservation $OclIsTypeOf_{Reservation}$-Reservation-invalid)
lemma $OclIsKindOf_{Reservation}$-Reservation-null : ((null::·Reservation) .oclIsKindOf(Reservation)) = true
by(simp only: $OclIsKindOf_{Reservation}$-Reservation $OclIsTypeOf_{Reservation}$-Reservation-null)
lemma $OclIsKindOf_{Reservation}$-OclAny-invalid : ((invalid::·OclAny) .oclIsKindOf(Reservation)) = invalid
by(simp only: $OclIsKindOf_{Reservation}$-OclAny $OclIsTypeOf_{Reservation}$-OclAny-invalid)
lemma $OclIsKindOf_{Reservation}$-OclAny-null : ((null::·OclAny) .oclIsKindOf(Reservation)) = true
by(simp only: $OclIsKindOf_{Reservation}$-OclAny $OclIsTypeOf_{Reservation}$-OclAny-null)
lemma $OclIsKindOf_{Reservation}$-Staff-invalid : ((invalid::·Staff) .oclIsKindOf(Reservation)) = invalid
by(simp only: $OclIsKindOf_{Reservation}$-Staff $OclIsTypeOf_{Reservation}$-Staff-invalid)
lemma $OclIsKindOf_{Reservation}$-Staff-null : ((null::·Staff) .oclIsKindOf(Reservation)) = true
by(simp only: $OclIsKindOf_{Reservation}$-Staff $OclIsTypeOf_{Reservation}$-Staff-null)
lemma $OclIsKindOf_{Reservation}$-Person-invalid : ((invalid::·Person) .oclIsKindOf(Reservation)) = invalid
by(simp only: $OclIsKindOf_{Reservation}$-Person $OclIsTypeOf_{Reservation}$-Person-invalid)
lemma $OclIsKindOf_{Reservation}$-Person-null : ((null::·Person) .oclIsKindOf(Reservation)) = true
by(simp only: $OclIsKindOf_{Reservation}$-Person $OclIsTypeOf_{Reservation}$-Person-null)
lemma $OclIsKindOf_{Reservation}$-Client-invalid : ((invalid::·Client) .oclIsKindOf(Reservation)) = invalid
by(simp only: $OclIsKindOf_{Reservation}$-Client $OclIsTypeOf_{Reservation}$-Client-invalid)
lemma $OclIsKindOf_{Reservation}$-Client-null : ((null::·Client) .oclIsKindOf(Reservation)) = true
by(simp only: $OclIsKindOf_{Reservation}$-Client $OclIsTypeOf_{Reservation}$-Client-null)
lemma $OclIsKindOf_{Reservation}$-Flight-invalid : ((invalid::·Flight) .oclIsKindOf(Reservation)) = invalid
by(simp only: $OclIsKindOf_{Reservation}$-Flight $OclIsTypeOf_{Reservation}$-Flight-invalid)
lemma $OclIsKindOf_{Reservation}$-Flight-null : ((null::·Flight) .oclIsKindOf(Reservation)) = true
by(simp only: $OclIsKindOf_{Reservation}$-Flight $OclIsTypeOf_{Reservation}$-Flight-null)
lemma $OclIsKindOf_{OclAny}$-OclAny-invalid : ((invalid::·OclAny) .oclIsKindOf(OclAny)) = invalid
by(simp only: $OclIsKindOf_{OclAny}$-OclAny $OclIsTypeOf_{OclAny}$-OclAny-invalid $OclIsKindOf_{Reservation}$-OclAny-invalid $OclIsKindOf_{Person}$-OclAny-invalid $OclIsKindOf_{Flight}$-OclAny-invalid, simp)
lemma $OclIsKindOf_{OclAny}$-OclAny-null : ((null::·OclAny) .oclIsKindOf(OclAny)) = true
by(simp only: $OclIsKindOf_{OclAny}$-OclAny $OclIsTypeOf_{OclAny}$-OclAny-null $OclIsKindOf_{Reservation}$-OclAny-null $OclIsKindOf_{Person}$-OclAny-null $OclIsKindOf_{Flight}$-OclAny-null, simp)
lemma $OclIsKindOf_{OclAny}$-Flight-invalid : ((invalid::·Flight) .oclIsKindOf(OclAny)) = invalid
by(simp only: $OclIsKindOf_{OclAny}$-Flight $OclIsTypeOf_{OclAny}$-Flight-invalid $OclIsKindOf_{Reservation}$-Flight-invalid $OclIsKindOf_{Person}$-Flight-invalid $OclIsKindOf_{Flight}$-Flight-invalid, simp)
lemma $OclIsKindOf_{OclAny}$-Flight-null : ((null::·Flight) .oclIsKindOf(OclAny)) = true
by(simp only: $OclIsKindOf_{OclAny}$-Flight $OclIsTypeOf_{OclAny}$-Flight-null $OclIsKindOf_{Reservation}$-Flight-null $OclIsKindOf_{Person}$-Flight-null $OclIsKindOf_{Flight}$-Flight-null, simp)
lemma $OclIsKindOf_{OclAny}$-Client-invalid : ((invalid::·Client) .oclIsKindOf(OclAny)) = invalid
by(simp only: $OclIsKindOf_{OclAny}$-Client $OclIsTypeOf_{OclAny}$-Client-invalid $OclIsKindOf_{Reservation}$-Client-invalid $OclIsKindOf_{Person}$-Client-invalid $OclIsKindOf_{Flight}$-Client-invalid, simp)
lemma $OclIsKindOf_{OclAny}$-Client-null : ((null::·Client) .oclIsKindOf(OclAny)) = true
by(simp only: $OclIsKindOf_{OclAny}$-Client $OclIsTypeOf_{OclAny}$-Client-null $OclIsKindOf_{Reservation}$-Client-null $OclIsKindOf_{Person}$-Client-null $OclIsKindOf_{Flight}$-Client-null, simp)
lemma $OclIsKindOf_{OclAny}$-Staff-invalid : ((invalid::·Staff) .oclIsKindOf(OclAny)) = invalid
by(simp only: $OclIsKindOf_{OclAny}$-Staff $OclIsTypeOf_{OclAny}$-Staff-invalid $OclIsKindOf_{Reservation}$-Staff-invalid $OclIsKindOf_{Person}$-Staff-invalid $OclIsKindOf_{Flight}$-Staff-invalid, simp)
lemma $OclIsKindOf_{OclAny}$-Staff-null : ((null::·Staff) .oclIsKindOf(OclAny)) = true
by(simp only: $OclIsKindOf_{OclAny}$-Staff $OclIsTypeOf_{OclAny}$-Staff-null $OclIsKindOf_{Reservation}$-Staff-null $OclIsKindOf_{Person}$-Staff-null $OclIsKindOf_{Flight}$-Staff-null, simp)
lemma $OclIsKindOf_{OclAny}$-Person-invalid : ((invalid::·Person) .oclIsKindOf(OclAny)) = invalid
by(simp only: $OclIsKindOf_{OclAny}$-Person $OclIsTypeOf_{OclAny}$-Person-invalid $OclIsKindOf_{Reservation}$-Person-invalid $OclIsKindOf_{Person}$-Person-invalid $OclIsKindOf_{Flight}$-Person-invalid, simp)
lemma $OclIsKindOf_{OclAny}$-Person-null : ((null::·Person) .oclIsKindOf(OclAny)) = true

by(*simp only*: $OclIsKindOf_{OclAny}$-*Person* $OclIsTypeOf_{OclAny}$-*Person-null* $OclIsKindOf_{Reservation}$-*Person-null* $OclIsKindOf_{Person}$-*Person-null* $OclIsKindOf_{Flight}$-*Person-null*, *simp*)
lemma $OclIsKindOf_{OclAny}$-*Reservation-invalid* : ((*invalid*::·*Reservation*) .*oclIsKindOf*(*OclAny*)) = *invalid*
by(*simp only*: $OclIsKindOf_{OclAny}$-*Reservation* $OclIsTypeOf_{OclAny}$-*Reservation-invalid* $OclIsKindOf_{Reservation}$-*Reservation-invalid* $OclIsKindOf_{Person}$-*Reservation-invalid* $OclIsKindOf_{Flight}$-*Reservation-invalid*, *simp*)
lemma $OclIsKindOf_{OclAny}$-*Reservation-null* : ((*null*::·*Reservation*) .*oclIsKindOf*(*OclAny*)) = *true*
by(*simp only*: $OclIsKindOf_{OclAny}$-*Reservation* $OclIsTypeOf_{OclAny}$-*Reservation-null* $OclIsKindOf_{Reservation}$-*Reservation-null* $OclIsKindOf_{Person}$-*Reservation-null* $OclIsKindOf_{Flight}$-*Reservation-null*, *simp*)

lemmas[*simp*,*code-unfold*] = $OclIsKindOf_{Client}$-*Client-invalid*
$OclIsKindOf_{Client}$-*Reservation-invalid*
$OclIsKindOf_{Client}$-*OclAny-invalid*
$OclIsKindOf_{Client}$-*Person-invalid*
$OclIsKindOf_{Client}$-*Staff-invalid*
$OclIsKindOf_{Client}$-*Flight-invalid*
$OclIsKindOf_{Client}$-*Client-null*
$OclIsKindOf_{Client}$-*Reservation-null*
$OclIsKindOf_{Client}$-*OclAny-null*
$OclIsKindOf_{Client}$-*Person-null*
$OclIsKindOf_{Client}$-*Staff-null*
$OclIsKindOf_{Client}$-*Flight-null*
$OclIsKindOf_{Reservation}$-*Client-invalid*
$OclIsKindOf_{Reservation}$-*Reservation-invalid*
$OclIsKindOf_{Reservation}$-*OclAny-invalid*
$OclIsKindOf_{Reservation}$-*Person-invalid*
$OclIsKindOf_{Reservation}$-*Staff-invalid*
$OclIsKindOf_{Reservation}$-*Flight-invalid*
$OclIsKindOf_{Reservation}$-*Client-null*
$OclIsKindOf_{Reservation}$-*Reservation-null*
$OclIsKindOf_{Reservation}$-*OclAny-null*
$OclIsKindOf_{Reservation}$-*Person-null*
$OclIsKindOf_{Reservation}$-*Staff-null*
$OclIsKindOf_{Reservation}$-*Flight-null*
$OclIsKindOf_{OclAny}$-*Client-invalid*
$OclIsKindOf_{OclAny}$-*Reservation-invalid*
$OclIsKindOf_{OclAny}$-*OclAny-invalid*
$OclIsKindOf_{OclAny}$-*Person-invalid*
$OclIsKindOf_{OclAny}$-*Staff-invalid*
$OclIsKindOf_{OclAny}$-*Flight-invalid*
$OclIsKindOf_{OclAny}$-*Client-null*
$OclIsKindOf_{OclAny}$-*Reservation-null*
$OclIsKindOf_{OclAny}$-*OclAny-null*
$OclIsKindOf_{OclAny}$-*Person-null*
$OclIsKindOf_{OclAny}$-*Staff-null*
$OclIsKindOf_{OclAny}$-*Flight-null*
$OclIsKindOf_{Person}$-*Client-invalid*
$OclIsKindOf_{Person}$-*Reservation-invalid*
$OclIsKindOf_{Person}$-*OclAny-invalid*
$OclIsKindOf_{Person}$-*Person-invalid*
$OclIsKindOf_{Person}$-*Staff-invalid*
$OclIsKindOf_{Person}$-*Flight-invalid*
$OclIsKindOf_{Person}$-*Client-null*
$OclIsKindOf_{Person}$-*Reservation-null*
$OclIsKindOf_{Person}$-*OclAny-null*
$OclIsKindOf_{Person}$-*Person-null*
$OclIsKindOf_{Person}$-*Staff-null*
$OclIsKindOf_{Person}$-*Flight-null*
$OclIsKindOf_{Staff}$-*Client-invalid*
$OclIsKindOf_{Staff}$-*Reservation-invalid*
$OclIsKindOf_{Staff}$-*OclAny-invalid*
$OclIsKindOf_{Staff}$-*Person-invalid*
$OclIsKindOf_{Staff}$-*Staff-invalid*
$OclIsKindOf_{Staff}$-*Flight-invalid*
$OclIsKindOf_{Staff}$-*Client-null*
$OclIsKindOf_{Staff}$-*Reservation-null*
$OclIsKindOf_{Staff}$-*OclAny-null*
$OclIsKindOf_{Staff}$-*Person-null*
$OclIsKindOf_{Staff}$-*Staff-null*
$OclIsKindOf_{Staff}$-*Flight-null*
$OclIsKindOf_{Flight}$-*Client-invalid*
$OclIsKindOf_{Flight}$-*Reservation-invalid*
$OclIsKindOf_{Flight}$-*OclAny-invalid*

OclIsKindOf$_{Flight}$-Person-invalid
OclIsKindOf$_{Flight}$-Staff-invalid
OclIsKindOf$_{Flight}$-Flight-invalid
OclIsKindOf$_{Flight}$-Client-null
OclIsKindOf$_{Flight}$-Reservation-null
OclIsKindOf$_{Flight}$-OclAny-null
OclIsKindOf$_{Flight}$-Person-null
OclIsKindOf$_{Flight}$-Staff-null
OclIsKindOf$_{Flight}$-Flight-null

Validity and Definedness Properties

lemma *OclIsKindOf$_{Flight}$-Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsKindOf(Flight)))$
by(*simp only*: *OclIsKindOf$_{Flight}$-Flight*, *rule OclIsTypeOf$_{Flight}$-Flight-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Flight}$-OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsKindOf(Flight)))$
by(*simp only*: *OclIsKindOf$_{Flight}$-OclAny*, *rule OclIsTypeOf$_{Flight}$-OclAny-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Flight}$-Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsKindOf(Flight)))$
by(*simp only*: *OclIsKindOf$_{Flight}$-Staff*, *rule OclIsTypeOf$_{Flight}$-Staff-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Flight}$-Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsKindOf(Flight)))$
by(*simp only*: *OclIsKindOf$_{Flight}$-Person*, *rule OclIsTypeOf$_{Flight}$-Person-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Flight}$-Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsKindOf(Flight)))$
by(*simp only*: *OclIsKindOf$_{Flight}$-Client*, *rule OclIsTypeOf$_{Flight}$-Client-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Flight}$-Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsKindOf(Flight)))$
by(*simp only*: *OclIsKindOf$_{Flight}$-Reservation*, *rule OclIsTypeOf$_{Flight}$-Reservation-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Client}$-Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsKindOf(Client)))$
by(*simp only*: *OclIsKindOf$_{Client}$-Client*, *rule OclIsTypeOf$_{Client}$-Client-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Client}$-Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsKindOf(Client)))$
by(*simp only*: *OclIsKindOf$_{Client}$-Person*, *rule OclIsTypeOf$_{Client}$-Person-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Client}$-OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsKindOf(Client)))$
by(*simp only*: *OclIsKindOf$_{Client}$-OclAny*, *rule OclIsTypeOf$_{Client}$-OclAny-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Client}$-Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsKindOf(Client)))$
by(*simp only*: *OclIsKindOf$_{Client}$-Staff*, *rule OclIsTypeOf$_{Client}$-Staff-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Client}$-Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsKindOf(Client)))$
by(*simp only*: *OclIsKindOf$_{Client}$-Reservation*, *rule OclIsTypeOf$_{Client}$-Reservation-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Client}$-Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsKindOf(Client)))$
by(*simp only*: *OclIsKindOf$_{Client}$-Flight*, *rule OclIsTypeOf$_{Client}$-Flight-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Staff}$-Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsKindOf(Staff)))$
by(*simp only*: *OclIsKindOf$_{Staff}$-Staff*, *rule OclIsTypeOf$_{Staff}$-Staff-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Staff}$-Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsKindOf(Staff)))$
by(*simp only*: *OclIsKindOf$_{Staff}$-Person*, *rule OclIsTypeOf$_{Staff}$-Person-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Staff}$-OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsKindOf(Staff)))$
by(*simp only*: *OclIsKindOf$_{Staff}$-OclAny*, *rule OclIsTypeOf$_{Staff}$-OclAny-defined*[*OF isdef*])
lemma *OclIsKindOf$_{Staff}$-Client-defined* :

assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsKindOf(Staff)))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Client*, *rule* $OclIsTypeOf_{Staff}$-*Client-defined*[*OF isdef*])
lemma $OclIsKindOf_{Staff}$-*Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsKindOf(Staff)))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Reservation*, *rule* $OclIsTypeOf_{Staff}$-*Reservation-defined*[*OF isdef*])
lemma $OclIsKindOf_{Staff}$-*Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(Staff)))$
by(*simp only*: $OclIsKindOf_{Staff}$-*Flight*, *rule* $OclIsTypeOf_{Staff}$-*Flight-defined*[*OF isdef*])
lemma $OclIsKindOf_{Person}$-*Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsKindOf(Person)))$
by(*simp only*: $OclIsKindOf_{Person}$-*Person*, *rule defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{Person}$-*Person-defined*[*OF isdef*], *OF* $OclIsKindOf_{Staff}$-*Person-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Client}$-*Person-defined*[*OF isdef*]])
lemma $OclIsKindOf_{Person}$-*OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsKindOf(Person)))$
by(*simp only*: $OclIsKindOf_{Person}$-*OclAny*, *rule defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{Person}$-*OclAny-defined*[*OF isdef*], *OF* $OclIsKindOf_{Staff}$-*OclAny-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Client}$-*OclAny-defined*[*OF isdef*]])
lemma $OclIsKindOf_{Person}$-*Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsKindOf(Person)))$
by(*simp only*: $OclIsKindOf_{Person}$-*Client*, *rule defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{Person}$-*Client-defined*[*OF isdef*], *OF* $OclIsKindOf_{Staff}$-*Client-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Client}$-*Client-defined*[*OF isdef*]])
lemma $OclIsKindOf_{Person}$-*Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsKindOf(Person)))$
by(*simp only*: $OclIsKindOf_{Person}$-*Staff*, *rule defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{Person}$-*Staff-defined*[*OF isdef*], *OF* $OclIsKindOf_{Staff}$-*Staff-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Client}$-*Staff-defined*[*OF isdef*]])
lemma $OclIsKindOf_{Person}$-*Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsKindOf(Person)))$
by(*simp only*: $OclIsKindOf_{Person}$-*Reservation*, *rule defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{Person}$-*Reservation-defined*[*OF isdef*], *OF* $OclIsKindOf_{Staff}$-*Reservation-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Client}$-*Reservation-defined*[*OF isdef*]])
lemma $OclIsKindOf_{Person}$-*Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(Person)))$
by(*simp only*: $OclIsKindOf_{Person}$-*Flight*, *rule defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{Person}$-*Flight-defined*[*OF isdef*], *OF* $OclIsKindOf_{Staff}$-*Flight-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Client}$-*Flight-defined*[*OF isdef*]])
lemma $OclIsKindOf_{Reservation}$-*Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsKindOf(Reservation)))$
by(*simp only*: $OclIsKindOf_{Reservation}$-*Reservation*, *rule* $OclIsTypeOf_{Reservation}$-*Reservation-defined*[*OF isdef*])
lemma $OclIsKindOf_{Reservation}$-*OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsKindOf(Reservation)))$
by(*simp only*: $OclIsKindOf_{Reservation}$-*OclAny*, *rule* $OclIsTypeOf_{Reservation}$-*OclAny-defined*[*OF isdef*])
lemma $OclIsKindOf_{Reservation}$-*Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsKindOf(Reservation)))$
by(*simp only*: $OclIsKindOf_{Reservation}$-*Staff*, *rule* $OclIsTypeOf_{Reservation}$-*Staff-defined*[*OF isdef*])
lemma $OclIsKindOf_{Reservation}$-*Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsKindOf(Reservation)))$
by(*simp only*: $OclIsKindOf_{Reservation}$-*Person*, *rule* $OclIsTypeOf_{Reservation}$-*Person-defined*[*OF isdef*])
lemma $OclIsKindOf_{Reservation}$-*Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsKindOf(Reservation)))$
by(*simp only*: $OclIsKindOf_{Reservation}$-*Client*, *rule* $OclIsTypeOf_{Reservation}$-*Client-defined*[*OF isdef*])
lemma $OclIsKindOf_{Reservation}$-*Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(Reservation)))$
by(*simp only*: $OclIsKindOf_{Reservation}$-*Flight*, *rule* $OclIsTypeOf_{Reservation}$-*Flight-defined*[*OF isdef*])
lemma $OclIsKindOf_{OclAny}$-*OclAny-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsKindOf(OclAny)))$
by(*simp only*: $OclIsKindOf_{OclAny}$-*OclAny*, *rule defined-or-I*[*OF defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{OclAny}$-*OclAny-defined*[*OF isdef*], *OF* $OclIsKindOf_{Reservation}$-*OclAny-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Person}$-*OclAny-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Flight}$-*OclAny-defined*[*OF isdef*]])
lemma $OclIsKindOf_{OclAny}$-*Flight-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$

shows $\tau \models (\delta\ ((X::Flight)\ .oclIsKindOf(OclAny)))$
by(*simp only*: $OclIsKindOf_{OclAny}$-*Flight*, *rule defined-or-I*[*OF defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{OclAny}$-*Flight-defined*[*OF isdef*], *OF* $OclIsKindOf_{Reservation}$-*Flight-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Person}$-*Flight-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Flight}$-*Flight-defined*[*OF isdef*]])
lemma $OclIsKindOf_{OclAny}$-*Client-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsKindOf(OclAny)))$
by(*simp only*: $OclIsKindOf_{OclAny}$-*Client*, *rule defined-or-I*[*OF defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{OclAny}$-*Client-defined*[*OF isdef*], *OF* $OclIsKindOf_{Reservation}$-*Client-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Person}$-*Client-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Flight}$-*Client-defined*[*OF isdef*]])
lemma $OclIsKindOf_{OclAny}$-*Staff-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsKindOf(OclAny)))$
by(*simp only*: $OclIsKindOf_{OclAny}$-*Staff*, *rule defined-or-I*[*OF defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{OclAny}$-*Staff-defined*[*OF isdef*], *OF* $OclIsKindOf_{Reservation}$-*Staff-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Person}$-*Staff-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Flight}$-*Staff-defined*[*OF isdef*]])
lemma $OclIsKindOf_{OclAny}$-*Person-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsKindOf(OclAny)))$
by(*simp only*: $OclIsKindOf_{OclAny}$-*Person*, *rule defined-or-I*[*OF defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{OclAny}$-*Person-defined*[*OF isdef*], *OF* $OclIsKindOf_{Reservation}$-*Person-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Person}$-*Person-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Flight}$-*Person-defined*[*OF isdef*]])
lemma $OclIsKindOf_{OclAny}$-*Reservation-defined* :
assumes *isdef*: $\tau \models (\upsilon\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsKindOf(OclAny)))$
by(*simp only*: $OclIsKindOf_{OclAny}$-*Reservation*, *rule defined-or-I*[*OF defined-or-I*[*OF defined-or-I*[*OF* $OclIsTypeOf_{OclAny}$-*Reservation-defined*[*OF isdef*], *OF* $OclIsKindOf_{Reservation}$-*Reservation-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Person}$-*Reservation-defined*[*OF isdef*]], *OF* $OclIsKindOf_{Flight}$-*Reservation-defined*[*OF isdef*]])

lemma $OclIsKindOf_{Flight}$-*Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Flight)\ .oclIsKindOf(Flight)))$
by(*rule* $OclIsKindOf_{Flight}$-*Flight-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Flight}$-*OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsKindOf(Flight)))$
by(*rule* $OclIsKindOf_{Flight}$-*OclAny-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Flight}$-*Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsKindOf(Flight)))$
by(*rule* $OclIsKindOf_{Flight}$-*Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Flight}$-*Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsKindOf(Flight)))$
by(*rule* $OclIsKindOf_{Flight}$-*Person-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Flight}$-*Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsKindOf(Flight)))$
by(*rule* $OclIsKindOf_{Flight}$-*Client-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Flight}$-*Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsKindOf(Flight)))$
by(*rule* $OclIsKindOf_{Flight}$-*Reservation-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Client}$-*Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Client)\ .oclIsKindOf(Client)))$
by(*rule* $OclIsKindOf_{Client}$-*Client-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Client}$-*Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Person)\ .oclIsKindOf(Client)))$
by(*rule* $OclIsKindOf_{Client}$-*Person-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Client}$-*OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::OclAny)\ .oclIsKindOf(Client)))$
by(*rule* $OclIsKindOf_{Client}$-*OclAny-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Client}$-*Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Staff)\ .oclIsKindOf(Client)))$
by(*rule* $OclIsKindOf_{Client}$-*Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma $OclIsKindOf_{Client}$-*Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::Reservation)\ .oclIsKindOf(Client)))$

by(*rule OclIsKindOf$_{Client}$-Reservation-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Client}$-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(Client)))$
by(*rule OclIsKindOf$_{Client}$-Flight-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Staff}$-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsKindOf(Staff)))$
by(*rule OclIsKindOf$_{Staff}$-Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Staff}$-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsKindOf(Staff)))$
by(*rule OclIsKindOf$_{Staff}$-Person-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Staff}$-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsKindOf(Staff)))$
by(*rule OclIsKindOf$_{Staff}$-OclAny-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Staff}$-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsKindOf(Staff)))$
by(*rule OclIsKindOf$_{Staff}$-Client-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Staff}$-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsKindOf(Staff)))$
by(*rule OclIsKindOf$_{Staff}$-Reservation-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Staff}$-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(Staff)))$
by(*rule OclIsKindOf$_{Staff}$-Flight-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Person}$-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsKindOf(Person)))$
by(*rule OclIsKindOf$_{Person}$-Person-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Person}$-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsKindOf(Person)))$
by(*rule OclIsKindOf$_{Person}$-OclAny-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Person}$-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsKindOf(Person)))$
by(*rule OclIsKindOf$_{Person}$-Client-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Person}$-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsKindOf(Person)))$
by(*rule OclIsKindOf$_{Person}$-Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Person}$-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsKindOf(Person)))$
by(*rule OclIsKindOf$_{Person}$-Reservation-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Person}$-Flight-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(Person)))$
by(*rule OclIsKindOf$_{Person}$-Flight-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Reservation}$-Reservation-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsKindOf(Reservation)))$
by(*rule OclIsKindOf$_{Reservation}$-Reservation-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Reservation}$-OclAny-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsKindOf(Reservation)))$
by(*rule OclIsKindOf$_{Reservation}$-OclAny-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Reservation}$-Staff-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsKindOf(Reservation)))$
by(*rule OclIsKindOf$_{Reservation}$-Staff-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Reservation}$-Person-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsKindOf(Reservation)))$
by(*rule OclIsKindOf$_{Reservation}$-Person-defined*[*OF isdef*[*THEN foundation20*]])
lemma *OclIsKindOf$_{Reservation}$-Client-defined'* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsKindOf(Reservation)))$
by(*rule OclIsKindOf$_{Reservation}$-Client-defined*[*OF isdef*[*THEN foundation20*]])

lemma $OclIsKindOf_{Reservation}$-Flight-defined' :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(Reservation)))$
by(rule $OclIsKindOf_{Reservation}$-Flight-defined[OF isdef[THEN foundation20]])
lemma $OclIsKindOf_{OclAny}$-OclAny-defined' :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot OclAny)\ .oclIsKindOf(OclAny)))$
by(rule $OclIsKindOf_{OclAny}$-OclAny-defined[OF isdef[THEN foundation20]])
lemma $OclIsKindOf_{OclAny}$-Flight-defined' :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Flight)\ .oclIsKindOf(OclAny)))$
by(rule $OclIsKindOf_{OclAny}$-Flight-defined[OF isdef[THEN foundation20]])
lemma $OclIsKindOf_{OclAny}$-Client-defined' :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Client)\ .oclIsKindOf(OclAny)))$
by(rule $OclIsKindOf_{OclAny}$-Client-defined[OF isdef[THEN foundation20]])
lemma $OclIsKindOf_{OclAny}$-Staff-defined' :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Staff)\ .oclIsKindOf(OclAny)))$
by(rule $OclIsKindOf_{OclAny}$-Staff-defined[OF isdef[THEN foundation20]])
lemma $OclIsKindOf_{OclAny}$-Person-defined' :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Person)\ .oclIsKindOf(OclAny)))$
by(rule $OclIsKindOf_{OclAny}$-Person-defined[OF isdef[THEN foundation20]])
lemma $OclIsKindOf_{OclAny}$-Reservation-defined' :
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models (\delta\ ((X::\cdot Reservation)\ .oclIsKindOf(OclAny)))$
by(rule $OclIsKindOf_{OclAny}$-Reservation-defined[OF isdef[THEN foundation20]])

Up Down Casting

lemma $actual\text{-}eq\text{-}static_{Flight}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Flight)\ .oclIsKindOf(Flight))$
apply(simp only: OclValid-def, insert isdef)
apply(simp only: $OclIsKindOf_{Flight}$-Flight)
apply(auto simp: foundation16 bot-option-def split: option.split $ty\mathcal{EXT}_{Flight}$.split ty_{Flight}.split)
by((simp-all add: false-def true-def OclOr-def OclAnd-def OclNot-def)?)
lemma $actual\text{-}eq\text{-}static_{Client}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Client)\ .oclIsKindOf(Client))$
apply(simp only: OclValid-def, insert isdef)
apply(simp only: $OclIsKindOf_{Client}$-Client)
apply(auto simp: foundation16 bot-option-def split: option.split $ty\mathcal{EXT}_{Client}$.split ty_{Client}.split)
by((simp-all add: false-def true-def OclOr-def OclAnd-def OclNot-def)?)
lemma $actual\text{-}eq\text{-}static_{Staff}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Staff)\ .oclIsKindOf(Staff))$
apply(simp only: OclValid-def, insert isdef)
apply(simp only: $OclIsKindOf_{Staff}$-Staff)
apply(auto simp: foundation16 bot-option-def split: option.split $ty\mathcal{EXT}_{Staff}$.split ty_{Staff}.split)
by((simp-all add: false-def true-def OclOr-def OclAnd-def OclNot-def)?)
lemma $actual\text{-}eq\text{-}static_{Person}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Person)\ .oclIsKindOf(Person))$
apply(simp only: OclValid-def, insert isdef)
apply(simp only: $OclIsKindOf_{Person}$-Person, subst (1) cp-OclOr, subst (2 1) cp-OclOr, simp only: $OclIsKindOf_{Staff}$-Person, simp only: $OclIsKindOf_{Client}$-Person)
apply(auto simp: cp-OclOr[symmetric] foundation16 bot-option-def $OclIsTypeOf_{Client}$-Person $OclIsTypeOf_{Staff}$-Person split: option.split $ty\mathcal{EXT}_{Person}$.split ty_{Person}.split $ty\mathcal{EXT}_{Client}$.split ty_{Client}.split $ty\mathcal{EXT}_{Staff}$.split ty_{Staff}.split)
by((simp-all add: false-def true-def OclOr-def OclAnd-def OclNot-def)?)
lemma $actual\text{-}eq\text{-}static_{Reservation}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Reservation)\ .oclIsKindOf(Reservation))$
apply(simp only: OclValid-def, insert isdef)
apply(simp only: $OclIsKindOf_{Reservation}$-Reservation)
apply(auto simp: foundation16 bot-option-def split: option.split $ty\mathcal{EXT}_{Reservation}$.split $ty_{Reservation}$.split)
by((simp-all add: false-def true-def OclOr-def OclAnd-def OclNot-def)?)
lemma $actual\text{-}eq\text{-}static_{OclAny}$:
assumes isdef: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(OclAny))$
apply(simp only: OclValid-def, insert isdef)
apply(simp only: $OclIsKindOf_{OclAny}$-OclAny, subst (1) cp-OclOr, subst (2 1) cp-OclOr, subst (3 2 1) cp-OclOr, simp only:

OclIsKindOf$_{Reservation}$-OclAny, simp only: OclIsKindOf$_{Person}$-OclAny, subst (4 3 2 1) cp-OclOr, subst (5 4 3 2 1) cp-OclOr, simp only: OclIsKindOf$_{Staff}$-OclAny, simp only: OclIsKindOf$_{Client}$-OclAny, simp only: OclIsKindOf$_{Flight}$-OclAny)
apply(*auto simp: cp-OclOr[symmetric] foundation16 bot-option-def OclIsTypeOf$_{Flight}$-OclAny OclIsTypeOf$_{Client}$-OclAny OclIsTypeOf$_{Staff}$-OclAny OclIsTypeOf$_{Person}$-OclAny OclIsTypeOf$_{Reservation}$-OclAny split: option.split ty$\mathcal{EXT}_{OclAny}$.split ty$_{OclAny}$.split ty$\mathcal{EXT}_{Flight}$.split ty$_{Flight}$.split ty$\mathcal{EXT}_{Client}$.split ty$_{Client}$.split ty$\mathcal{EXT}_{Staff}$.split ty$_{Staff}$.split ty$\mathcal{EXT}_{Person}$.split ty$_{Person}$.split ty$\mathcal{EXT}_{Reservation}$.split ty$_{Reservation}$.split*)
by((*simp-all add: false-def true-def OclOr-def OclAnd-def OclNot-def*)*?*)

lemma *actualKind$_{Flight}$-larger-staticKind$_{OclAny}$* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Flight)\ .oclIsKindOf(OclAny))$
apply(*simp only: OclIsKindOf$_{OclAny}$-Flight*)
by(*rule foundation25′, rule actual-eq-static$_{Flight}$[OF isdef]*)
lemma *actualKind$_{Client}$-larger-staticKind$_{Person}$* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Client)\ .oclIsKindOf(Person))$
apply(*simp only: OclIsKindOf$_{Person}$-Client*)
by(*rule foundation25′, rule actual-eq-static$_{Client}$[OF isdef]*)
lemma *actualKind$_{Client}$-larger-staticKind$_{OclAny}$* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Client)\ .oclIsKindOf(OclAny))$
apply(*simp only: OclIsKindOf$_{OclAny}$-Client*)
by(*rule foundation25, rule foundation25′, rule actualKind$_{Client}$-larger-staticKind$_{Person}$[OF isdef]*)
lemma *actualKind$_{Staff}$-larger-staticKind$_{Person}$* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Staff)\ .oclIsKindOf(Person))$
apply(*simp only: OclIsKindOf$_{Person}$-Staff*)
by(*rule foundation25, rule foundation25′, rule actual-eq-static$_{Staff}$[OF isdef]*)
lemma *actualKind$_{Staff}$-larger-staticKind$_{OclAny}$* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Staff)\ .oclIsKindOf(OclAny))$
apply(*simp only: OclIsKindOf$_{OclAny}$-Staff*)
by(*rule foundation25, rule foundation25′, rule actualKind$_{Staff}$-larger-staticKind$_{Person}$[OF isdef]*)
lemma *actualKind$_{Person}$-larger-staticKind$_{OclAny}$* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Person)\ .oclIsKindOf(OclAny))$
apply(*simp only: OclIsKindOf$_{OclAny}$-Person*)
by(*rule foundation25, rule foundation25′, rule actual-eq-static$_{Person}$[OF isdef]*)
lemma *actualKind$_{Reservation}$-larger-staticKind$_{OclAny}$* :
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models ((X::\cdot Reservation)\ .oclIsKindOf(OclAny))$
apply(*simp only: OclIsKindOf$_{OclAny}$-Reservation*)
by(*rule foundation25, rule foundation25, rule foundation25′, rule actual-eq-static$_{Reservation}$[OF isdef]*)

lemma *not-OclIsKindOf$_{Flight}$-then-OclAny-OclIsTypeOf-others-unfold* :
assumes *isdef*: $(\tau \models (\delta\ (X)))$
assumes *iskin*: $(\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Flight)))$
shows $(\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Flight)))$
using *iskin*
apply(*simp only: OclIsKindOf$_{Flight}$-OclAny*)
done
lemma *not-OclIsKindOf$_{Client}$-then-Person-OclIsTypeOf-others-unfold* :
assumes *isdef*: $(\tau \models (\delta\ (X)))$
assumes *iskin*: $(\tau \models ((X::\cdot Person)\ .oclIsKindOf(Client)))$
shows $(\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Client)))$
using *iskin*
apply(*simp only: OclIsKindOf$_{Client}$-Person*)
done
lemma *not-OclIsKindOf$_{Client}$-then-OclAny-OclIsTypeOf-others-unfold* :
assumes *isdef*: $(\tau \models (\delta\ (X)))$
assumes *iskin*: $(\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Client)))$
shows $(\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Client)))$
using *iskin*
apply(*simp only: OclIsKindOf$_{Client}$-OclAny*)
done
lemma *not-OclIsKindOf$_{Staff}$-then-Person-OclIsTypeOf-others-unfold* :
assumes *isdef*: $(\tau \models (\delta\ (X)))$
assumes *iskin*: $(\tau \models ((X::\cdot Person)\ .oclIsKindOf(Staff)))$
shows $(\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Staff)))$
using *iskin*
apply(*simp only: OclIsKindOf$_{Staff}$-Person*)

done
lemma *not-OclIsKindOf$_{Staff}$-then-OclAny-OclIsTypeOf-others-unfold* :
assumes *isdef*: $(\tau \models (\delta\ (X)))$
assumes *iskin*: $(\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Staff)))$
shows $(\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Staff)))$
using *iskin*
apply(*simp only*: *OclIsKindOf$_{Staff}$-OclAny*)
done
lemma *not-OclIsKindOf$_{Person}$-then-OclAny-OclIsTypeOf-others-unfold* :
assumes *isdef*: $(\tau \models (\delta\ (X)))$
assumes *iskin*: $(\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Person)))$
shows $((\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Person))) \vee ((\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Client))) \vee (\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Staff)))))$
using *iskin*
apply(*simp only*: *OclIsKindOf$_{Person}$-OclAny*)
apply(*erule foundation26*[*OF defined-or-I*[*OF OclIsTypeOf$_{Person}$-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf$_{Staff}$-OclAny-defined'*[*OF isdef*]], *OF OclIsKindOf$_{Client}$-OclAny-defined'*[*OF isdef*]])
apply(*erule foundation26*[*OF OclIsTypeOf$_{Person}$-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf$_{Staff}$-OclAny-defined'*[*OF isdef*]])
apply(*simp*)
apply(*drule not-OclIsKindOf$_{Staff}$-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*], *blast*)
apply(*drule not-OclIsKindOf$_{Client}$-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*], *blast*)
done
lemma *not-OclIsKindOf$_{Reservation}$-then-OclAny-OclIsTypeOf-others-unfold* :
assumes *isdef*: $(\tau \models (\delta\ (X)))$
assumes *iskin*: $(\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Reservation)))$
shows $(\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Reservation)))$
using *iskin*
apply(*simp only*: *OclIsKindOf$_{Reservation}$-OclAny*)
done

lemma *not-OclIsKindOf$_{Flight}$-then-OclAny-OclIsTypeOf-others* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $(\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny)) \vee (\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Person)) \vee \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Reservation))))$
using *actual-eq-static$_{OclAny}$*[*OF isdef*]
apply(*simp only*: *OclIsKindOf$_{OclAny}$-OclAny*)
apply(*erule foundation26*[*OF defined-or-I*[*OF defined-or-I*[*OF OclIsTypeOf$_{OclAny}$-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf$_{Reservation}$-OclAny-defined'*[*OF isdef*]], *OF OclIsKindOf$_{Person}$-OclAny-defined'*[*OF isdef*]], *OF OclIsKindOf$_{Flight}$-OclAny-defined'*[*OF isdef*]])
apply(*erule foundation26*[*OF defined-or-I*[*OF OclIsTypeOf$_{OclAny}$-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf$_{Reservation}$-OclAny-defined'*[*OF isdef*]], *OF OclIsKindOf$_{Person}$-OclAny-defined'*[*OF isdef*]])
apply(*erule foundation26*[*OF OclIsTypeOf$_{OclAny}$-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf$_{Reservation}$-OclAny-defined'*[*OF isdef*]])
apply(*simp*)
apply(*simp*)
apply(*simp*)
apply(*simp add*: *iskin*)
done
lemma *not-OclIsKindOf$_{Client}$-then-Person-OclIsTypeOf-others* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot Person)\ .oclIsKindOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $(\tau \models ((X::\cdot Person)\ .oclIsTypeOf(Person)) \vee \tau \models ((X::\cdot Person)\ .oclIsKindOf(Staff)))$
using *actual-eq-static$_{Person}$*[*OF isdef*]
apply(*simp only*: *OclIsKindOf$_{Person}$-Person*)
apply(*erule foundation26*[*OF defined-or-I*[*OF OclIsTypeOf$_{Person}$-Person-defined'*[*OF isdef*], *OF OclIsKindOf$_{Staff}$-Person-defined'*[*OF isdef*]], *OF OclIsKindOf$_{Client}$-Person-defined'*[*OF isdef*]])
apply(*erule foundation26*[*OF OclIsTypeOf$_{Person}$-Person-defined'*[*OF isdef*], *OF OclIsKindOf$_{Staff}$-Person-defined'*[*OF isdef*]])
apply(*simp*)
apply(*simp*)
apply(*simp add*: *iskin*)
done
lemma *not-OclIsKindOf$_{Client}$-then-OclAny-OclIsTypeOf-others* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $(\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny)) \vee (\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(Person)) \vee (\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Reservation)) \vee (\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Flight)) \vee \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Staff))))))$
using *actual-eq-static$_{OclAny}$*[*OF isdef*]
apply(*simp only*: *OclIsKindOf$_{OclAny}$-OclAny*)
apply(*erule foundation26*[*OF defined-or-I*[*OF defined-or-I*[*OF OclIsTypeOf$_{OclAny}$-OclAny-defined'*[*OF isdef*],

OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Person}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Flight}$-OclAny-defined'[OF isdef]])
apply(*erule foundation26[OF defined-or-I[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Person}$-OclAny-defined'[OF isdef]])*
apply(*erule foundation26[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]]*)
apply(*simp*)
apply(*simp*)
apply(*simp only: OclIsKindOf$_{Person}$-OclAny*)
apply(*erule foundation26[OF defined-or-I[OF OclIsTypeOf$_{Person}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Staff}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Client}$-OclAny-defined'[OF isdef]]*)
apply(*erule foundation26[OF OclIsTypeOf$_{Person}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Staff}$-OclAny-defined'[OF isdef]]*)
apply(*simp*)
apply(*simp*)
apply(*simp add: iskin*)
apply(*simp*)
done
lemma *not-OclIsKindOf$_{Staff}$-then-Person-OclIsTypeOf-others* :
assumes *iskin*: $\neg\ \tau \models ((X::Person)\ .oclIsKindOf(Staff))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $(\tau \models ((X::Person)\ .oclIsTypeOf(Person)) \vee \tau \models ((X::Person)\ .oclIsKindOf(Client)))$
using *actual-eq-static$_{Person}$[OF isdef]*
apply(*simp only: OclIsKindOf$_{Person}$-Person*)
apply(*erule foundation26[OF defined-or-I[OF OclIsTypeOf$_{Person}$-Person-defined'[OF isdef], OF OclIsKindOf$_{Staff}$-Person-defined'[OF isdef]], OF OclIsKindOf$_{Client}$-Person-defined'[OF isdef]]*)
apply(*erule foundation26[OF OclIsTypeOf$_{Person}$-Person-defined'[OF isdef], OF OclIsKindOf$_{Staff}$-Person-defined'[OF isdef]]*)
apply(*simp*)
apply(*simp add: iskin*)
apply(*simp*)
done
lemma *not-OclIsKindOf$_{Staff}$-then-OclAny-OclIsTypeOf-others* :
assumes *iskin*: $\neg\ \tau \models ((X::OclAny)\ .oclIsKindOf(Staff))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $(\tau \models ((X::OclAny)\ .oclIsTypeOf(OclAny)) \vee (\tau \models ((X::OclAny)\ .oclIsTypeOf(Person)) \vee (\tau \models ((X::OclAny)\ .oclIsKindOf(Reservation)) \vee (\tau \models ((X::OclAny)\ .oclIsKindOf(Flight)) \vee \tau \models ((X::OclAny)\ .oclIsKindOf(Client))))))$
using *actual-eq-static$_{OclAny}$[OF isdef]*
apply(*simp only: OclIsKindOf$_{OclAny}$-OclAny*)
apply(*erule foundation26[OF defined-or-I[OF defined-or-I[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Person}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Flight}$-OclAny-defined'[OF isdef]]*)
apply(*erule foundation26[OF defined-or-I[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Person}$-OclAny-defined'[OF isdef]]*)
apply(*erule foundation26[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]]*)
apply(*simp*)
apply(*simp*)
apply(*simp only: OclIsKindOf$_{Person}$-OclAny*)
apply(*erule foundation26[OF defined-or-I[OF OclIsTypeOf$_{Person}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Staff}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Client}$-OclAny-defined'[OF isdef]]*)
apply(*erule foundation26[OF OclIsTypeOf$_{Person}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Staff}$-OclAny-defined'[OF isdef]]*)
apply(*simp*)
apply(*simp add: iskin*)
apply(*simp*)
apply(*simp*)
done
lemma *not-OclIsKindOf$_{Person}$-then-OclAny-OclIsTypeOf-others* :
assumes *iskin*: $\neg\ \tau \models ((X::OclAny)\ .oclIsKindOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $(\tau \models ((X::OclAny)\ .oclIsTypeOf(OclAny)) \vee (\tau \models ((X::OclAny)\ .oclIsKindOf(Reservation)) \vee \tau \models ((X::OclAny)\ .oclIsKindOf(Flight))))$
using *actual-eq-static$_{OclAny}$[OF isdef]*
apply(*simp only: OclIsKindOf$_{OclAny}$-OclAny*)
apply(*erule foundation26[OF defined-or-I[OF defined-or-I[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Person}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Flight}$-OclAny-defined'[OF isdef]]*)
apply(*erule foundation26[OF defined-or-I[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]], OF OclIsKindOf$_{Person}$-OclAny-defined'[OF isdef]]*)
apply(*erule foundation26[OF OclIsTypeOf$_{OclAny}$-OclAny-defined'[OF isdef], OF OclIsKindOf$_{Reservation}$-OclAny-defined'[OF isdef]]*)
apply(*simp*)

apply(*simp*)
apply(*simp add*: *iskin*)
apply(*simp*)
done
lemma *not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $(\tau \models ((X::\cdot OclAny)\ .oclIsTypeOf(OclAny)) \vee (\tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Person)) \vee \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Flight))))$
using *actual-eq-static*$_{OclAny}$[*OF isdef*]
apply(*simp only*: *OclIsKindOf*$_{OclAny}$*-OclAny*)
apply(*erule foundation26*[*OF defined-or-I*[*OF defined-or-I*[*OF OclIsTypeOf*$_{OclAny}$*-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf*$_{Reservation}$*-OclAny-defined'*[*OF isdef*]], *OF OclIsKindOf*$_{Person}$*-OclAny-defined'*[*OF isdef*]], *OF OclIsKindOf*$_{Flight}$*-OclAny-defined'*[*OF isdef*]])
apply(*erule foundation26*[*OF defined-or-I*[*OF OclIsTypeOf*$_{OclAny}$*-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf*$_{Reservation}$*-OclAny-defined'*[*OF isdef*]], *OF OclIsKindOf*$_{Person}$*-OclAny-defined'*[*OF isdef*]])
apply(*erule foundation26*[*OF OclIsTypeOf*$_{OclAny}$*-OclAny-defined'*[*OF isdef*], *OF OclIsKindOf*$_{Reservation}$*-OclAny-defined'*[*OF isdef*]])
apply(*simp*)
apply(*simp add*: *iskin*)
apply(*simp*)
apply(*simp*)
done

lemma *down-cast-kind*$_{Flight}$*-from-OclAny-to-Flight* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Flight))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Flight)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Flight}$*-then-OclAny-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{OclAny}$*-from-OclAny-to-Flight*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Person}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*auto simp*: *isdef down-cast-type*$_{Staff}$*-from-OclAny-to-Flight down-cast-type*$_{Person}$*-from-OclAny-to-Flight down-cast-type*$_{Client}$*-from-OclAny-to-Flight*)
apply(*drule not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Reservation}$*-from-OclAny-to-Flight*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Client}$*-from-Person-to-Client* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot Person)\ .oclIsKindOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Client}$*-then-Person-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{Person}$*-from-Person-to-Client*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Staff}$*-then-Person-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Staff}$*-from-Person-to-Client*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Client}$*-from-OclAny-to-Client* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Client))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Client}$*-then-OclAny-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{OclAny}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
apply(*rule down-cast-type*$_{Person}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Reservation}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Flight}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Flight}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Staff}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Staff}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Staff}$*-from-Person-to-Staff* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot Person)\ .oclIsKindOf(Staff))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Staff}$*-then-Person-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{Person}$*-from-Person-to-Staff*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Client}$*-then-Person-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Client}$*-from-Person-to-Staff*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Staff}$*-from-OclAny-to-Staff* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Staff))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$

apply(*insert not-OclIsKindOf*$_{Staff}$*-then-OclAny-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{OclAny}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
apply(*rule down-cast-type*$_{Person}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Reservation}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Flight}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Flight}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Client}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Client}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Person}$*-from-OclAny-to-Person* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Person)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Person}$*-then-OclAny-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{OclAny}$*-from-OclAny-to-Person*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Reservation}$*-from-OclAny-to-Person*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Flight}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Flight}$*-from-OclAny-to-Person*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Person}$*-from-OclAny-to-Client* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Client)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Person}$*-then-OclAny-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{OclAny}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Reservation}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Flight}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Flight}$*-from-OclAny-to-Client*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Person}$*-from-OclAny-to-Staff* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Person))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Staff)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Person}$*-then-OclAny-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{OclAny}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Reservation}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Flight}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Flight}$*-from-OclAny-to-Staff*, *simp only*: , *simp only*: *isdef*)
done
lemma *down-cast-kind*$_{Reservation}$*-from-OclAny-to-Reservation* :
assumes *iskin*: $\neg\ \tau \models ((X::\cdot OclAny)\ .oclIsKindOf(Reservation))$
assumes *isdef*: $\tau \models (\delta\ (X))$
shows $\tau \models (X\ .oclAsType(Reservation)) \triangleq invalid$
apply(*insert not-OclIsKindOf*$_{Reservation}$*-then-OclAny-OclIsTypeOf-others*[*OF iskin*, *OF isdef*], *elim disjE*)
apply(*rule down-cast-type*$_{OclAny}$*-from-OclAny-to-Reservation*, *simp only*: , *simp only*: *isdef*)
apply(*drule not-OclIsKindOf*$_{Person}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*auto simp*: *isdef down-cast-type*$_{Staff}$*-from-OclAny-to-Reservation down-cast-type*$_{Person}$*-from-OclAny-to-Reservation down-cast-type*$_{Client}$*-from-OclAny-to-Reservation*)
apply(*drule not-OclIsKindOf*$_{Flight}$*-then-OclAny-OclIsTypeOf-others-unfold*[*OF isdef*])
apply(*rule down-cast-type*$_{Flight}$*-from-OclAny-to-Reservation*, *simp only*: , *simp only*: *isdef*)
done

Const

B.7 Class Model: OclAllInstances

definition *Flight* = *OclAsType*$_{Flight}$-$\mathfrak{A}$
definition *Client* = *OclAsType*$_{Client}$-$\mathfrak{A}$
definition *Staff* = *OclAsType*$_{Staff}$-$\mathfrak{A}$
definition *Person* = *OclAsType*$_{Person}$-$\mathfrak{A}$
definition *Reservation* = *OclAsType*$_{Reservation}$-$\mathfrak{A}$
definition *OclAny* = *OclAsType*$_{OclAny}$-$\mathfrak{A}$

lemmas[*simp*,*code-unfold*] = *Flight-def*
Client-def
Staff-def
Person-def

Reservation-def
OclAny-def

lemma $OclAsType_{OclAny}$-$\mathfrak{A}$-*some* : ($OclAsType_{OclAny}$-$\mathfrak{A}$ (*x*)) ≠ *None*
by(*simp add*: $OclAsType_{OclAny}$-$\mathfrak{A}$-*def*)

lemma $OclAllInstances\text{-}generic_{OclAny}$-*exec* :
shows (*OclAllInstances-generic* (*pre-post*) (*OclAny*)) = (λτ. (*Abs-Set*$_{base}$ (⌊⌊*Some* ‘ *OclAny* ‘ (*ran* ((*heap* ((*pre-post* (τ))))))⌋⌋)))
proof − let *?S1* = (λτ. *OclAny* ‘ (*ran* ((*heap* ((*pre-post* (τ))))))) show *?thesis*
proof − let *?S2* = (λτ. ((*?S1*) (τ)) − {*None*}) show *?thesis*
proof − have *B*: (⋀τ. ((*?S2*) (τ)) ⊆ ((*?S1*) (τ))) by(*auto*) show *?thesis*
proof − have *C*: (⋀τ. ((*?S1*) (τ)) ⊆ ((*?S2*) (τ))) by(*auto simp*: $OclAsType_{OclAny}$-$\mathfrak{A}$-*some*) show *?thesis*
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def* $OclIsKindOf_{OclAny}$-*OclAny*)
by(*insert equalityI*[*OF B*, *OF C*], *simp*) qed qed qed qed
lemma $OclAllInstances\text{-}at\text{-}post_{OclAny}$-*exec* :
shows (*OclAllInstances-at-post* (*OclAny*)) = (λτ. (*Abs-Set*$_{base}$ (⌊⌊*Some* ‘ *OclAny* ‘ (*ran* ((*heap* ((*snd* (τ))))))⌋⌋)))
unfolding *OclAllInstances-at-post-def*
by(*rule* $OclAllInstances\text{-}generic_{OclAny}$-*exec*)
lemma $OclAllInstances\text{-}at\text{-}pre_{OclAny}$-*exec* :
shows (*OclAllInstances-at-pre* (*OclAny*)) = (λτ. (*Abs-Set*$_{base}$ (⌊⌊*Some* ‘ *OclAny* ‘ (*ran* ((*heap* ((*fst* (τ))))))⌋⌋)))
unfolding *OclAllInstances-at-pre-def*
by(*rule* $OclAllInstances\text{-}generic_{OclAny}$-*exec*)

OclIsTypeOf

lemma *ex-ssubst* : (∀ *x* ∈ *B*. (*s* (*x*)) = (*t* (*x*))) ⟹ (∃ *x* ∈ *B*. (*P* ((*s* (*x*))))) = (∃ *x* ∈ *B*. (*P* ((*t* (*x*)))))
by(*simp*)
lemma *ex-def* : *x* ∈ ⌈⌈⌊⌊*Some* ‘ (*X* − {*None*})⌋⌋⌉⌉ ⟹ (∃ *y*. *x* = ⌊⌊*y*⌋⌋)
by(*auto*)

lemma *Flight-OclAllInstances-generic-OclIsTypeOf*$_{Flight}$: τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Flight*))) ($OclIsTypeOf_{Flight}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$-*inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where *s* = (λ*x*. (((λ-. *x*) .*oclIsTypeOf*(*Flight*)) (τ))) and *t* = (λ-. (*true* (τ)))])
apply(*intro ballI actual-eq-static*$_{Flight}$[*simplified OclValid-def*, *simplified* $OclIsKindOf_{Flight}$-*Flight*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Flight-OclAllInstances-at-post-OclIsTypeOf*$_{Flight}$:
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Flight*))) ($OclIsTypeOf_{Flight}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Flight-OclAllInstances-generic-OclIsTypeOf*$_{Flight}$)
lemma *Flight-OclAllInstances-at-pre-OclIsTypeOf*$_{Flight}$:
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Flight*))) ($OclIsTypeOf_{Flight}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Flight-OclAllInstances-generic-OclIsTypeOf*$_{Flight}$)
lemma *Client-OclAllInstances-generic-OclIsTypeOf*$_{Client}$: τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Client*))) ($OclIsTypeOf_{Client}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$-*inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where *s* = (λ*x*. (((λ-. *x*) .*oclIsTypeOf*(*Client*)) (τ))) and *t* = (λ-. (*true* (τ)))])
apply(*intro ballI actual-eq-static*$_{Client}$[*simplified OclValid-def*, *simplified* $OclIsKindOf_{Client}$-*Client*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Client-OclAllInstances-at-post-OclIsTypeOf*$_{Client}$:
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Client*))) ($OclIsTypeOf_{Client}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Client-OclAllInstances-generic-OclIsTypeOf*$_{Client}$)
lemma *Client-OclAllInstances-at-pre-OclIsTypeOf*$_{Client}$:
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Client*))) ($OclIsTypeOf_{Client}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Client-OclAllInstances-generic-OclIsTypeOf*$_{Client}$)
lemma *Staff-OclAllInstances-generic-OclIsTypeOf*$_{Staff}$: τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Staff*))) ($OclIsTypeOf_{Staff}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])

apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsTypeOf(Staff))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actual-eq-static*$_{Staff}$[*simplified OclValid-def*, *simplified OclIsKindOf*$_{Staff}$*-Staff*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Staff-OclAllInstances-at-post-OclIsTypeOf*$_{Staff}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Staff*))) (*OclIsTypeOf*$_{Staff}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Staff-OclAllInstances-generic-OclIsTypeOf*$_{Staff}$)
lemma *Staff-OclAllInstances-at-pre-OclIsTypeOf*$_{Staff}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Staff*))) (*OclIsTypeOf*$_{Staff}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Staff-OclAllInstances-generic-OclIsTypeOf*$_{Staff}$)
lemma *Person-OclAllInstances-generic-OclIsTypeOf*$_{Person}$*1* :
assumes [*simp*]: $(\bigwedge x.\ (pre\text{-}post\ ((x\ ,\ x))) = x)$
shows $(\exists \tau.\ \tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Person*))) (*OclIsTypeOf*$_{Person}$)))
apply(*rule exI*[where $x = \tau_0$], *simp add*: τ_0*-def OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
by(*simp*)
lemma *Person-OclAllInstances-at-post-OclIsTypeOf*$_{Person}$*1* :
shows $(\exists \tau.\ \tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Person*))) (*OclIsTypeOf*$_{Person}$)))
unfolding *OclAllInstances-at-post-def*
by(*rule Person-OclAllInstances-generic-OclIsTypeOf*$_{Person}$*1*, *simp*)
lemma *Person-OclAllInstances-at-pre-OclIsTypeOf*$_{Person}$*1* :
shows $(\exists \tau.\ \tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Person*))) (*OclIsTypeOf*$_{Person}$)))
unfolding *OclAllInstances-at-pre-def*
by(*rule Person-OclAllInstances-generic-OclIsTypeOf*$_{Person}$*1*, *simp*)
lemma *Person-OclAllInstances-generic-OclIsTypeOf*$_{Person}$*2* :
assumes [*simp*]: $(\bigwedge x.\ (pre\text{-}post\ ((x\ ,\ x))) = x)$
shows $(\exists \tau.\ \tau \models$ (*not* ((*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Person*))) (*OclIsTypeOf*$_{Person}$)))))
proof − fix *oid a* show *?thesis*
proof − let *?t0* = (*state.make* ((*Map.empty* (*oid* ↦ (*in*$_{Person}$ ((*mk*$_{Person}$ ((*mk*$\mathcal{EXT}_{Person\text{-}Staff}$ (*a*))) (*None*)))))))
(*Map.empty*)) show *?thesis*
apply(*rule exI*[where $x = (?t0\ ,\ ?t0)$], *simp add*: *OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def OclAsType*$_{Person}$-$\mathfrak{A}$*-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
by(*simp add*: *state.make-def OclNot-def*) qed qed
lemma *Person-OclAllInstances-at-post-OclIsTypeOf*$_{Person}$*2* :
shows $(\exists \tau.\ \tau \models$ (*not* ((*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Person*))) (*OclIsTypeOf*$_{Person}$)))))
unfolding *OclAllInstances-at-post-def*
by(*rule Person-OclAllInstances-generic-OclIsTypeOf*$_{Person}$*2*, *simp*)
lemma *Person-OclAllInstances-at-pre-OclIsTypeOf*$_{Person}$*2* :
shows $(\exists \tau.\ \tau \models$ (*not* ((*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Person*))) (*OclIsTypeOf*$_{Person}$)))))
unfolding *OclAllInstances-at-pre-def*
by(*rule Person-OclAllInstances-generic-OclIsTypeOf*$_{Person}$*2*, *simp*)
lemma *Reservation-OclAllInstances-generic-OclIsTypeOf*$_{Reservation}$: $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Reservation*))) (*OclIsTypeOf*$_{Reservation}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsTypeOf(Reservation))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actual-eq-static*$_{Reservation}$[*simplified OclValid-def*, *simplified OclIsKindOf*$_{Reservation}$*-Reservation*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Reservation-OclAllInstances-at-post-OclIsTypeOf*$_{Reservation}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Reservation*))) (*OclIsTypeOf*$_{Reservation}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Reservation-OclAllInstances-generic-OclIsTypeOf*$_{Reservation}$)
lemma *Reservation-OclAllInstances-at-pre-OclIsTypeOf*$_{Reservation}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Reservation*))) (*OclIsTypeOf*$_{Reservation}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Reservation-OclAllInstances-generic-OclIsTypeOf*$_{Reservation}$)
lemma *OclAny-OclAllInstances-generic-OclIsTypeOf*$_{OclAny}$*1* :
assumes [*simp*]: $(\bigwedge x.\ (pre\text{-}post\ ((x\ ,\ x))) = x)$
shows $(\exists \tau.\ \tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*OclAny*))) (*OclIsTypeOf*$_{OclAny}$)))
apply(*rule exI*[where $x = \tau_0$], *simp add*: τ_0*-def OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)

apply(*subst* (*1 2 3*) *Abs-Set$_{base}$-inverse*, *simp add*: *bot-option-def*)
by(*simp*)
lemma *OclAny-OclAllInstances-at-post-OclIsTypeOf$_{OclAny}$1* :
shows (∃ τ. τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*OclAny*))) (*OclIsTypeOf$_{OclAny}$*)))
unfolding *OclAllInstances-at-post-def*
by(*rule OclAny-OclAllInstances-generic-OclIsTypeOf$_{OclAny}$1*, *simp*)
lemma *OclAny-OclAllInstances-at-pre-OclIsTypeOf$_{OclAny}$1* :
shows (∃ τ. τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*OclAny*))) (*OclIsTypeOf$_{OclAny}$*)))
unfolding *OclAllInstances-at-pre-def*
by(*rule OclAny-OclAllInstances-generic-OclIsTypeOf$_{OclAny}$1*, *simp*)
lemma *OclAny-OclAllInstances-generic-OclIsTypeOf$_{OclAny}$2* :
assumes [*simp*]: (⋀x. (*pre-post* ((x , x))) = x)
shows (∃ τ. τ ⊨ (*not* ((*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*OclAny*))) (*OclIsTypeOf$_{OclAny}$*)))))
proof − fix *oid a* show *?thesis*
proof − let *?t0* = (*state.make* ((*Map.empty* (*oid* ↦ (*in$_{OclAny}$* ((*mk$_{OclAny}$* ((*mk$\mathcal{EXT}_{OclAny\text{-}Reservation}$* (*a*)))))))))
(*Map.empty*)) show *?thesis*
apply(*rule exI*[where x = (*?t0* , *?t0*)], *simp add*: *OclValid-def del*: *OclAllInstances-generic-def*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def OclAsType$_{OclAny}$-$\mathfrak{A}$-def*)
apply(*subst* (*1 2 3*) *Abs-Set$_{base}$-inverse*, *simp add*: *bot-option-def*)
by(*simp add*: *state.make-def OclNot-def*) qed qed
lemma *OclAny-OclAllInstances-at-post-OclIsTypeOf$_{OclAny}$2* :
shows (∃ τ. τ ⊨ (*not* ((*UML-Set.OclForall* ((*OclAllInstances-at-post* (*OclAny*))) (*OclIsTypeOf$_{OclAny}$*)))))
unfolding *OclAllInstances-at-post-def*
by(*rule OclAny-OclAllInstances-generic-OclIsTypeOf$_{OclAny}$2*, *simp*)
lemma *OclAny-OclAllInstances-at-pre-OclIsTypeOf$_{OclAny}$2* :
shows (∃ τ. τ ⊨ (*not* ((*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*OclAny*))) (*OclIsTypeOf$_{OclAny}$*)))))
unfolding *OclAllInstances-at-pre-def*
by(*rule OclAny-OclAllInstances-generic-OclIsTypeOf$_{OclAny}$2*, *simp*)

OclIsKindOf

lemma *Flight-OclAllInstances-generic-OclIsKindOf$_{Flight}$* : τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Flight*))) (*OclIsKindOf$_{Flight}$*))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf$_{Flight}$-Flight*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set$_{base}$-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where s = (λx. (((λ-. x) .*oclIsKindOf*(*Flight*)) (τ))) and t = (λ-. (*true* (τ)))])
apply(*intro ballI actual-eq-static$_{Flight}$*[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Flight-OclAllInstances-at-post-OclIsKindOf$_{Flight}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Flight*))) (*OclIsKindOf$_{Flight}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Flight-OclAllInstances-generic-OclIsKindOf$_{Flight}$*)
lemma *Flight-OclAllInstances-at-pre-OclIsKindOf$_{Flight}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Flight*))) (*OclIsKindOf$_{Flight}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Flight-OclAllInstances-generic-OclIsKindOf$_{Flight}$*)
lemma *Client-OclAllInstances-generic-OclIsKindOf$_{Client}$* : τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Client*))) (*OclIsKindOf$_{Client}$*))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf$_{Client}$-Client*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set$_{base}$-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where s = (λx. (((λ-. x) .*oclIsKindOf*(*Client*)) (τ))) and t = (λ-. (*true* (τ)))])
apply(*intro ballI actual-eq-static$_{Client}$*[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Client-OclAllInstances-at-post-OclIsKindOf$_{Client}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Client*))) (*OclIsKindOf$_{Client}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Client-OclAllInstances-generic-OclIsKindOf$_{Client}$*)
lemma *Client-OclAllInstances-at-pre-OclIsKindOf$_{Client}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Client*))) (*OclIsKindOf$_{Client}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Client-OclAllInstances-generic-OclIsKindOf$_{Client}$*)
lemma *Staff-OclAllInstances-generic-OclIsKindOf$_{Staff}$* : τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Staff*))) (*OclIsKindOf$_{Staff}$*))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf$_{Staff}$-Staff*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)

apply(*subst (1 2 3) Abs-Set$_{base}$-inverse, simp add: bot-option-def*)
apply(*subst (1 2 3) ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(Staff))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actual-eq-static$_{Staff}$*[*simplified OclValid-def*])
apply(*drule ex-def, erule exE, simp*)
by(*simp*)
lemma *Staff-OclAllInstances-at-post-OclIsKindOf$_{Staff}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Staff*))) (*OclIsKindOf$_{Staff}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Staff-OclAllInstances-generic-OclIsKindOf$_{Staff}$*)
lemma *Staff-OclAllInstances-at-pre-OclIsKindOf$_{Staff}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Staff*))) (*OclIsKindOf$_{Staff}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Staff-OclAllInstances-generic-OclIsKindOf$_{Staff}$*)
lemma *Person-OclAllInstances-generic-OclIsKindOf$_{Person}$* : $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Person*))) (*OclIsKindOf$_{Person}$*))
apply(*simp add: OclValid-def del: OclAllInstances-generic-def OclIsKindOf$_{Person}$-Person*)
apply(*simp only: UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only: OclAllInstances-generic-def*)
apply(*subst (1 2 3) Abs-Set$_{base}$-inverse, simp add: bot-option-def*)
apply(*subst (1 2 3) ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(Person))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actual-eq-static$_{Person}$*[*simplified OclValid-def*])
apply(*drule ex-def, erule exE, simp*)
by(*simp*)
lemma *Person-OclAllInstances-at-post-OclIsKindOf$_{Person}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Person*))) (*OclIsKindOf$_{Person}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Person-OclAllInstances-generic-OclIsKindOf$_{Person}$*)
lemma *Person-OclAllInstances-at-pre-OclIsKindOf$_{Person}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Person*))) (*OclIsKindOf$_{Person}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Person-OclAllInstances-generic-OclIsKindOf$_{Person}$*)
lemma *Reservation-OclAllInstances-generic-OclIsKindOf$_{Reservation}$* : $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Reservation*))) (*OclIsKindOf$_{Reservation}$*))
apply(*simp add: OclValid-def del: OclAllInstances-generic-def OclIsKindOf$_{Reservation}$-Reservation*)
apply(*simp only: UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only: OclAllInstances-generic-def*)
apply(*subst (1 2 3) Abs-Set$_{base}$-inverse, simp add: bot-option-def*)
apply(*subst (1 2 3) ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(Reservation))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actual-eq-static$_{Reservation}$*[*simplified OclValid-def*])
apply(*drule ex-def, erule exE, simp*)
by(*simp*)
lemma *Reservation-OclAllInstances-at-post-OclIsKindOf$_{Reservation}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Reservation*))) (*OclIsKindOf$_{Reservation}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Reservation-OclAllInstances-generic-OclIsKindOf$_{Reservation}$*)
lemma *Reservation-OclAllInstances-at-pre-OclIsKindOf$_{Reservation}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Reservation*))) (*OclIsKindOf$_{Reservation}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Reservation-OclAllInstances-generic-OclIsKindOf$_{Reservation}$*)
lemma *OclAny-OclAllInstances-generic-OclIsKindOf$_{OclAny}$* : $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*OclAny*))) (*OclIsKindOf$_{OclAny}$*))
apply(*simp add: OclValid-def del: OclAllInstances-generic-def OclIsKindOf$_{OclAny}$-OclAny*)
apply(*simp only: UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only: OclAllInstances-generic-def*)
apply(*subst (1 2 3) Abs-Set$_{base}$-inverse, simp add: bot-option-def*)
apply(*subst (1 2 3) ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(OclAny))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actual-eq-static$_{OclAny}$*[*simplified OclValid-def*])
apply(*drule ex-def, erule exE, simp*)
by(*simp*)
lemma *OclAny-OclAllInstances-at-post-OclIsKindOf$_{OclAny}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*OclAny*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule OclAny-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)
lemma *OclAny-OclAllInstances-at-pre-OclIsKindOf$_{OclAny}$* :
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*OclAny*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule OclAny-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)

lemma *Flight-OclAllInstances-generic-OclIsKindOf$_{OclAny}$* : $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Flight*))) (*OclIsKindOf$_{OclAny}$*))
apply(*simp add: OclValid-def del: OclAllInstances-generic-def OclIsKindOf$_{OclAny}$-Flight*)
apply(*simp only: UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])

apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(OclAny))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actualKind*$_{Flight}$*-larger-staticKind*$_{OclAny}$[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Flight-OclAllInstances-at-post-OclIsKindOf*$_{OclAny}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Flight*))) (*OclIsKindOf*$_{OclAny}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Flight-OclAllInstances-generic-OclIsKindOf*$_{OclAny}$)
lemma *Flight-OclAllInstances-at-pre-OclIsKindOf*$_{OclAny}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Flight*))) (*OclIsKindOf*$_{OclAny}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Flight-OclAllInstances-generic-OclIsKindOf*$_{OclAny}$)
lemma *Client-OclAllInstances-generic-OclIsKindOf*$_{Person}$: $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Client*))) (*OclIsKindOf*$_{Person}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf*$_{Person}$*-Client*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(Person))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actualKind*$_{Client}$*-larger-staticKind*$_{Person}$[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Client-OclAllInstances-at-post-OclIsKindOf*$_{Person}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Client*))) (*OclIsKindOf*$_{Person}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Client-OclAllInstances-generic-OclIsKindOf*$_{Person}$)
lemma *Client-OclAllInstances-at-pre-OclIsKindOf*$_{Person}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Client*))) (*OclIsKindOf*$_{Person}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Client-OclAllInstances-generic-OclIsKindOf*$_{Person}$)
lemma *Client-OclAllInstances-generic-OclIsKindOf*$_{OclAny}$: $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Client*))) (*OclIsKindOf*$_{OclAny}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf*$_{OclAny}$*-Client*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(OclAny))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actualKind*$_{Client}$*-larger-staticKind*$_{OclAny}$[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Client-OclAllInstances-at-post-OclIsKindOf*$_{OclAny}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Client*))) (*OclIsKindOf*$_{OclAny}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Client-OclAllInstances-generic-OclIsKindOf*$_{OclAny}$)
lemma *Client-OclAllInstances-at-pre-OclIsKindOf*$_{OclAny}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Client*))) (*OclIsKindOf*$_{OclAny}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Client-OclAllInstances-generic-OclIsKindOf*$_{OclAny}$)
lemma *Staff-OclAllInstances-generic-OclIsKindOf*$_{Person}$: $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Staff*))) (*OclIsKindOf*$_{Person}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf*$_{Person}$*-Staff*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst* (*1 2 3*) *Abs-Set*$_{base}$*-inverse*, *simp add*: *bot-option-def*)
apply(*subst* (*1 2 3*) *ex-ssubst*[where $s = (\lambda x.\ (((\lambda\text{-}.\ x)\ .oclIsKindOf(Person))\ (\tau)))$ and $t = (\lambda\text{-}.\ (true\ (\tau)))$])
apply(*intro ballI actualKind*$_{Staff}$*-larger-staticKind*$_{Person}$[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Staff-OclAllInstances-at-post-OclIsKindOf*$_{Person}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Staff*))) (*OclIsKindOf*$_{Person}$))
unfolding *OclAllInstances-at-post-def*
by(*rule Staff-OclAllInstances-generic-OclIsKindOf*$_{Person}$)
lemma *Staff-OclAllInstances-at-pre-OclIsKindOf*$_{Person}$:
shows $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Staff*))) (*OclIsKindOf*$_{Person}$))
unfolding *OclAllInstances-at-pre-def*
by(*rule Staff-OclAllInstances-generic-OclIsKindOf*$_{Person}$)
lemma *Staff-OclAllInstances-generic-OclIsKindOf*$_{OclAny}$: $\tau \models$ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Staff*))) (*OclIsKindOf*$_{OclAny}$))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf*$_{OclAny}$*-Staff*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)

apply(*subst (1 2 3) Abs-Set$_{base}$-inverse, simp add: bot-option-def*)
apply(*subst (1 2 3) ex-ssubst*[where *s* = (λ*x*. (((λ-. *x*) .*oclIsKindOf*(*OclAny*)) (τ))) and *t* = (λ-. (*true* (τ)))])
apply(*intro ballI actualKind$_{Staff}$-larger-staticKind$_{OclAny}$*[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Staff-OclAllInstances-at-post-OclIsKindOf$_{OclAny}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Staff*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Staff-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)
lemma *Staff-OclAllInstances-at-pre-OclIsKindOf$_{OclAny}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Staff*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Staff-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)
lemma *Person-OclAllInstances-generic-OclIsKindOf$_{OclAny}$* : τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Person*))) (*OclIsKindOf$_{OclAny}$*))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf$_{OclAny}$-Person*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst (1 2 3) Abs-Set$_{base}$-inverse, simp add*: *bot-option-def*)
apply(*subst (1 2 3) ex-ssubst*[where *s* = (λ*x*. (((λ-. *x*) .*oclIsKindOf*(*OclAny*)) (τ))) and *t* = (λ-. (*true* (τ)))])
apply(*intro ballI actualKind$_{Person}$-larger-staticKind$_{OclAny}$*[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Person-OclAllInstances-at-post-OclIsKindOf$_{OclAny}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Person*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Person-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)
lemma *Person-OclAllInstances-at-pre-OclIsKindOf$_{OclAny}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Person*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Person-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)
lemma *Reservation-OclAllInstances-generic-OclIsKindOf$_{OclAny}$* : τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-generic* (*pre-post*) (*Reservation*))) (*OclIsKindOf$_{OclAny}$*))
apply(*simp add*: *OclValid-def del*: *OclAllInstances-generic-def OclIsKindOf$_{OclAny}$-Reservation*)
apply(*simp only*: *UML-Set.OclForall-def refl if-True OclAllInstances-generic-defined*[*simplified OclValid-def*])
apply(*simp only*: *OclAllInstances-generic-def*)
apply(*subst (1 2 3) Abs-Set$_{base}$-inverse, simp add*: *bot-option-def*)
apply(*subst (1 2 3) ex-ssubst*[where *s* = (λ*x*. (((λ-. *x*) .*oclIsKindOf*(*OclAny*)) (τ))) and *t* = (λ-. (*true* (τ)))])
apply(*intro ballI actualKind$_{Reservation}$-larger-staticKind$_{OclAny}$*[*simplified OclValid-def*])
apply(*drule ex-def*, *erule exE*, *simp*)
by(*simp*)
lemma *Reservation-OclAllInstances-at-post-OclIsKindOf$_{OclAny}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-post* (*Reservation*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-post-def*
by(*rule Reservation-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)
lemma *Reservation-OclAllInstances-at-pre-OclIsKindOf$_{OclAny}$* :
shows τ ⊨ (*UML-Set.OclForall* ((*OclAllInstances-at-pre* (*Reservation*))) (*OclIsKindOf$_{OclAny}$*))
unfolding *OclAllInstances-at-pre-def*
by(*rule Reservation-OclAllInstances-generic-OclIsKindOf$_{OclAny}$*)

B.8 Class Model: The Accessors

Definition

ML ⟨*val oidFlight-1-passengers = 2*⟩
ML ⟨*val oidClient-0-cl-res = 1*⟩
ML ⟨*val oidClient-0-flights = 2*⟩
ML ⟨*val oidStaff-0-flights = 2*⟩
ML ⟨*val oidPerson-0-flights = 2*⟩
ML ⟨*val oidReservation-0-prev = 0*⟩
ML ⟨*val oidReservation-1-next = 0*⟩
ML ⟨*val oidReservation-1-client = 1*⟩

definition *oid$_{Flight}$-1---passengers = 2*
definition *oid$_{Client}$-0---cl-res = 1*
definition *oid$_{Client}$-0---flights = 2*
definition *oid$_{Staff}$-0---flights = 2*
definition *oid$_{Person}$-0---flights = 2*
definition *oid$_{Reservation}$-0---prev = 0*
definition *oid$_{Reservation}$-1---next = 0*
definition *oid$_{Reservation}$-1---client = 1*

definition *eval-extract x f* = (λτ. (*case x τ of* ⌊⌊*obj*⌋⌋ ⇒ (*f* ((*oid-of* (*obj*))) (τ))
| - ⇒ *invalid* τ))
definition *in-pre-state* = *fst*
definition *in-post-state* = *snd*
definition *reconst-basetype* = (λ*x* -. ⌊⌊*x*⌋⌋)
definition *reconst-basetype*$_{Void}$ *x* = *Abs-Void*$_{base}$ *o* (*reconst-basetype* (*x*))

ML ‹*val switch2-01* = (*fn* [*x0* , *x1*] => (*x0* , *x1*))›
ML ‹*val switch2-10* = (*fn* [*x0* , *x1*] => (*x1* , *x0*))›

definition *switch*$_2$*-01* = (λ [*x0* , *x1*] ⇒ (*x0* , *x1*))
definition *switch*$_2$*-10* = (λ [*x0* , *x1*] ⇒ (*x1* , *x0*))
definition *deref-assocs pre-post to-from assoc-oid f oid* = (λτ. (*case* (*assocs* ((*pre-post* (τ))) (*assoc-oid*)) *of* ⌊*S*⌋ ⇒ (*f* ((*deref-assocs-list* (*to-from*) (*oid*) (*S*))) (τ))
| - ⇒ (*invalid* (τ))))

definition *deref-oid*$_{Flight}$ *fst-snd f oid* = (λτ. (*case* (*heap* (*fst-snd* τ) (*oid*)) *of* ⌊*in*$_{Flight}$ *obj*⌋ ⇒ *f obj* τ
| - ⇒ *invalid* τ))
definition *deref-oid*$_{Client}$ *fst-snd f oid* = (λτ. (*case* (*heap* (*fst-snd* τ) (*oid*)) *of* ⌊*in*$_{Client}$ *obj*⌋ ⇒ *f obj* τ
| - ⇒ *invalid* τ))
definition *deref-oid*$_{Staff}$ *fst-snd f oid* = (λτ. (*case* (*heap* (*fst-snd* τ) (*oid*)) *of* ⌊*in*$_{Staff}$ *obj*⌋ ⇒ *f obj* τ
| - ⇒ *invalid* τ))
definition *deref-oid*$_{Person}$ *fst-snd f oid* = (λτ. (*case* (*heap* (*fst-snd* τ) (*oid*)) *of* ⌊*in*$_{Person}$ *obj*⌋ ⇒ *f obj* τ
| - ⇒ *invalid* τ))
definition *deref-oid*$_{Reservation}$ *fst-snd f oid* = (λτ. (*case* (*heap* (*fst-snd* τ) (*oid*)) *of* ⌊*in*$_{Reservation}$ *obj*⌋ ⇒ *f obj* τ
| - ⇒ *invalid* τ))
definition *deref-oid*$_{OclAny}$ *fst-snd f oid* = (λτ. (*case* (*heap* (*fst-snd* τ) (*oid*)) *of* ⌊*in*$_{OclAny}$ *obj*⌋ ⇒ *f obj* τ
| - ⇒ *invalid* τ))

definition *deref-assocs*$_{Flight}$*-1---passengers fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-10*) (*oid*$_{Flight}$*-1---passengers*) (*f*)) ∘ *oid-of*
definition *deref-assocs*$_{Client}$*-0---cl-res fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-01*) (*oid*$_{Client}$*-0---cl-res*) (*f*)) ∘ *oid-of*
definition *deref-assocs*$_{Client}$*-0---flights fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-01*) (*oid*$_{Client}$*-0---flights*) (*f*)) ∘ *oid-of*
definition *deref-assocs*$_{Staff}$*-0---flights fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-01*) (*oid*$_{Staff}$*-0---flights*) (*f*)) ∘ *oid-of*
definition *deref-assocs*$_{Person}$*-0---flights fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-01*) (*oid*$_{Person}$*-0---flights*) (*f*)) ∘ *oid-of*
definition *deref-assocs*$_{Reservation}$*-0---prev fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-01*) (*oid*$_{Reservation}$*-0---prev*) (*f*)) ∘ *oid-of*
definition *deref-assocs*$_{Reservation}$*-1---next fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-10*) (*oid*$_{Reservation}$*-1---next*) (*f*)) ∘ *oid-of*
definition *deref-assocs*$_{Reservation}$*-1---client fst-snd f* = (*deref-assocs* (*fst-snd*) (*switch*$_2$*-10*) (*oid*$_{Reservation}$*-1---client*) (*f*)) ∘ *oid-of*

definition *select*$_{Flight}$*--seats f* = (λ (*mk*$_{Flight}$ (-) (⊥) (-) (-) (-)) ⇒ *null*
| (*mk*$_{Flight}$ (-) (⌊*x---seats*⌋) (-) (-) (-)) ⇒ (*f* (*x---seats*)))
definition *select*$_{Flight}$*--from f* = (λ (*mk*$_{Flight}$ (-) (-) (⊥) (-) (-)) ⇒ *null*
| (*mk*$_{Flight}$ (-) (-) (⌊*x---from*⌋) (-) (-)) ⇒ (*f* (*x---from*)))
definition *select*$_{Flight}$*--to f* = (λ (*mk*$_{Flight}$ (-) (-) (-) (⊥) (-)) ⇒ *null*
| (*mk*$_{Flight}$ (-) (-) (-) (⌊*x---to*⌋) (-)) ⇒ (*f* (*x---to*)))
definition *select*$_{Flight}$*--fl-res f* = (λ (*mk*$_{Flight}$ (-) (-) (-) (-) (⊥)) ⇒ *null*
| (*mk*$_{Flight}$ (-) (-) (-) (-) (⌊*x---fl-res*⌋)) ⇒ (*f* (*x---fl-res*)))
definition *select*$_{Client}$*--address f* = (λ (*mk*$_{Client}$ (-) (⊥)) ⇒ *null*
| (*mk*$_{Client}$ (-) (⌊*x---address*⌋)) ⇒ (*f* (*x---address*)))
definition *select*$_{Person}$*--name f* = (λ (*mk*$_{Person}$ (-) (⊥)) ⇒ *null*
| (*mk*$_{Person}$ (-) (⌊*x---name*⌋)) ⇒ (*f* (*x---name*)))
definition *select*$_{Reservation}$*--id f* = (λ (*mk*$_{Reservation}$ (-) (⊥) (-) (-)) ⇒ *null*
| (*mk*$_{Reservation}$ (-) (⌊*x---id*⌋) (-) (-)) ⇒ (*f* (*x---id*)))
definition *select*$_{Reservation}$*--date f* = (λ (*mk*$_{Reservation}$ (-) (-) (⊥) (-)) ⇒ *null*
| (*mk*$_{Reservation}$ (-) (-) (⌊*x---date*⌋) (-)) ⇒ (*f* (*x---date*)))
definition *select*$_{Reservation}$*--flight f* = (λ (*mk*$_{Reservation}$ (-) (-) (-) (⊥)) ⇒ *null*
| (*mk*$_{Reservation}$ (-) (-) (-) (⌊*x---flight*⌋)) ⇒ (*f* (*x---flight*)))
definition *select*$_{Client}$*--name f* = (λ (*mk*$_{Client}$ ((*mk*$\mathcal{EXT}_{Client}$ (-) (⊥))) (-)) ⇒ *null*
| (*mk*$_{Client}$ ((*mk*$\mathcal{EXT}_{Client}$ (-) (⌊*x---name*⌋))) (-)) ⇒ (*f* (*x---name*)))
definition *select*$_{Staff}$*--name f* = (λ (*mk*$_{Staff}$ ((*mk*$\mathcal{EXT}_{Staff}$ (-) (⊥)))) ⇒ *null*
| (*mk*$_{Staff}$ ((*mk*$\mathcal{EXT}_{Staff}$ (-) (⌊*x---name*⌋)))) ⇒ (*f* (*x---name*)))

definition $select_{Flight}$--*passengers* = *select-object*$_{Set}$
definition $select_{Client}$--*cl-res* = *select-object*$_{Set}$
definition $select_{Client}$--*flights* = *select-object*$_{Set}$
definition $select_{Staff}$--*flights* = *select-object*$_{Set}$
definition $select_{Person}$--*flights* = *select-object*$_{Set}$
definition $select_{Reservation}$--*prev* = *select-object-any*$_{Set}$
definition $select_{Reservation}$--*next* = *select-object-any*$_{Set}$
definition $select_{Reservation}$--*client* = *select-object-any*$_{Set}$

consts *dot-1---passengers* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Set-Person* ((-) *.passengers*)
consts *dot-1---passengersat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Set-Person* ((-) *.passengers@pre*)
consts *dot--seats* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Integer* ((-) *.seats*)
consts *dot--seatsat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Integer* ((-) *.seats@pre*)
consts *dot--from* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.from*)
consts *dot--fromat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.from@pre*)
consts *dot--to* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.to*)
consts *dot--toat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.to@pre*)
consts *dot--fl-res* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Sequence-Reservation* ((-) *.fl′-res*)
consts *dot--fl-resat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Sequence-Reservation* ((-) *.fl′-res@pre*)
consts *dot-0---cl-res* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Set-Reservation* ((-) *.cl′-res*)
consts *dot-0---cl-resat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Set-Reservation* ((-) *.cl′-res@pre*)
consts *dot--address* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.address*)
consts *dot--addressat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.address@pre*)
consts *dot-0---flights* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Set-Flight* ((-) *.flights*)
consts *dot-0---flightsat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Set-Flight* ((-) *.flights@pre*)
consts *dot--name* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.name*)
consts *dot--nameat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *String* ((-) *.name@pre*)
consts *dot-0---prev* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Reservation* ((-) *.prev*)
consts *dot-0---prevat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Reservation* ((-) *.prev@pre*)
consts *dot-1---next* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Reservation* ((-) *.next*)
consts *dot-1---nextat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Reservation* ((-) *.next@pre*)
consts *dot-1---client* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Client* ((-) *.client*)
consts *dot-1---clientat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Client* ((-) *.client@pre*)
consts *dot--id* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Integer* ((-) *.id*)
consts *dot--idat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Integer* ((-) *.id@pre*)
consts *dot--date* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Week* ((-) *.date*)
consts *dot--dateat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ *Week* ((-) *.date@pre*)
consts *dot--flight* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Flight* ((-) *.flight*)
consts *dot--flightat-pre* :: $(\mathfrak{A}, 'a)$ *val* ⇒ ·*Flight* ((-) *.flight@pre*)

overloading *dot-1---passengers* ≡ (*dot-1---passengers*::(·*Flight*) ⇒ -)
begin
definition dot_{Flight}-*1---passengers* : (*x*::·*Flight*) *.passengers* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Flight}$ (*in-post-state*) ((*deref-assocs*$_{Flight}$-*1---passengers* (*in-post-state*) ((*select*$_{Flight}$--*passengers* ((*deref-oid*$_{Person}$ (*in-post-state*) (*reconst-basetype*))))))))))
end
overloading *dot--seats* ≡ (*dot--seats*::(·*Flight*) ⇒ -)
begin
definition dot_{Flight}--*seats* : (*x*::·*Flight*) *.seats* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Flight}$ (*in-post-state*) ((*select*$_{Flight}$--*seats* (*reconst-basetype*))))))
end
overloading *dot--from* ≡ (*dot--from*::(·*Flight*) ⇒ -)
begin
definition dot_{Flight}--*from* : (*x*::·*Flight*) *.from* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Flight}$ (*in-post-state*) ((*select*$_{Flight}$--*from* (*reconst-basetype*))))))
end
overloading *dot--to* ≡ (*dot--to*::(·*Flight*) ⇒ -)
begin
definition dot_{Flight}--*to* : (*x*::·*Flight*) *.to* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Flight}$ (*in-post-state*) ((*select*$_{Flight}$--*to* (*reconst-basetype*))))))
end
overloading *dot--fl-res* ≡ (*dot--fl-res*::(·*Flight*) ⇒ -)
begin
definition dot_{Flight}--*fl-res* : (*x*::·*Flight*) *.fl-res* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Flight}$ (*in-post-state*) ((*select*$_{Flight}$--*fl-res* ((*select-object*$_{Seq}$ ((*deref-oid*$_{Reservation}$ (*in-post-state*) (*reconst-basetype*)))))))))
end
overloading *dot-1---passengersat-pre* ≡ (*dot-1---passengersat-pre*::(·*Flight*) ⇒ -)
begin
definition dot_{Flight}-*1---passengersat-pre* : (*x*::·*Flight*) *.passengers@pre* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Flight}$ (*in-pre-state*) ((*deref-assocs*$_{Flight}$-*1---passengers* (*in-pre-state*) ((*select*$_{Flight}$--*passengers* ((*deref-oid*$_{Person}$ (*in-pre-state*) (*reconst-basetype*))))))))))

```
end
overloading dot--seatsat-pre ≡ (dot--seatsat-pre::(·Flight) ⇒ -)
begin
  definition dot_Flight--seatsat-pre : (x::·Flight) .seats@pre ≡ (eval-extract (x) ((deref-oid_Flight (in-pre-state) ((select_Flight--seats (reconst-basetype))))))
end
overloading dot--fromat-pre ≡ (dot--fromat-pre::(·Flight) ⇒ -)
begin
  definition dot_Flight--fromat-pre : (x::·Flight) .from@pre ≡ (eval-extract (x) ((deref-oid_Flight (in-pre-state) ((select_Flight--from (reconst-basetype))))))
end
overloading dot--toat-pre ≡ (dot--toat-pre::(·Flight) ⇒ -)
begin
  definition dot_Flight--toat-pre : (x::·Flight) .to@pre ≡ (eval-extract (x) ((deref-oid_Flight (in-pre-state) ((select_Flight--to (reconst-basetype))))))
end
overloading dot--fl-resat-pre ≡ (dot--fl-resat-pre::(·Flight) ⇒ -)
begin
  definition dot_Flight--fl-resat-pre : (x::·Flight) .fl-res@pre ≡ (eval-extract (x) ((deref-oid_Flight (in-pre-state) ((select_Flight--fl-res ((select-object_Seq ((deref-oid_Reservation (in-pre-state) (reconst-basetype))))))))))
end
overloading dot-0---cl-res ≡ (dot-0---cl-res::(·Client) ⇒ -)
begin
  definition dot_Client-0---cl-res : (x::·Client) .cl-res ≡ (eval-extract (x) ((deref-oid_Client (in-post-state) ((deref-assocs_Client-0---cl-res (in-post-state) ((select_Client--cl-res ((deref-oid_Reservation (in-post-state) (reconst-basetype))))))))))
end
overloading dot--address ≡ (dot--address::(·Client) ⇒ -)
begin
  definition dot_Client--address : (x::·Client) .address ≡ (eval-extract (x) ((deref-oid_Client (in-post-state) ((select_Client--address (reconst-basetype))))))
end
overloading dot-0---cl-resat-pre ≡ (dot-0---cl-resat-pre::(·Client) ⇒ -)
begin
  definition dot_Client-0---cl-resat-pre : (x::·Client) .cl-res@pre ≡ (eval-extract (x) ((deref-oid_Client (in-pre-state) ((deref-assocs_Client-0---cl-res (in-pre-state) ((select_Client--cl-res ((deref-oid_Reservation (in-pre-state) (reconst-basetype))))))))))
end
overloading dot--addressat-pre ≡ (dot--addressat-pre::(·Client) ⇒ -)
begin
  definition dot_Client--addressat-pre : (x::·Client) .address@pre ≡ (eval-extract (x) ((deref-oid_Client (in-pre-state) ((select_Client--address (reconst-basetype))))))
end
overloading dot-0---flights ≡ (dot-0---flights::(·Person) ⇒ -)
begin
  definition dot_Person-0---flights : (x::·Person) .flights ≡ (eval-extract (x) ((deref-oid_Person (in-post-state) ((deref-assocs_Person-0---flights (in-post-state) ((select_Person--flights ((deref-oid_Flight (in-post-state) (reconst-basetype))))))))))
end
overloading dot--name ≡ (dot--name::(·Person) ⇒ -)
begin
  definition dot_Person--name : (x::·Person) .name ≡ (eval-extract (x) ((deref-oid_Person (in-post-state) ((select_Person--name (reconst-basetype))))))
end
overloading dot-0---flightsat-pre ≡ (dot-0---flightsat-pre::(·Person) ⇒ -)
begin
  definition dot_Person-0---flightsat-pre : (x::·Person) .flights@pre ≡ (eval-extract (x) ((deref-oid_Person (in-pre-state) ((deref-assocs_Person-0---flights (in-pre-state) ((select_Person--flights ((deref-oid_Flight (in-pre-state) (reconst-basetype))))))))))
end
overloading dot--nameat-pre ≡ (dot--nameat-pre::(·Person) ⇒ -)
begin
  definition dot_Person--nameat-pre : (x::·Person) .name@pre ≡ (eval-extract (x) ((deref-oid_Person (in-pre-state) ((select_Person--name (reconst-basetype))))))
end
overloading dot-0---prev ≡ (dot-0---prev::(·Reservation) ⇒ -)
begin
  definition dot_Reservation-0---prev : (x::·Reservation) .prev ≡ (eval-extract (x) ((deref-oid_Reservation (in-post-state) ((deref-assocs_Reservation-0---prev (in-post-state) ((select_Reservation--prev ((deref-oid_Reservation (in-post-state) (reconst-basetype))))))))))
end
overloading dot-1---next ≡ (dot-1---next::(·Reservation) ⇒ -)
begin
  definition dot_Reservation-1---next : (x::·Reservation) .next ≡ (eval-extract (x) ((deref-oid_Reservation (in-post-state) ((deref-assocs_Reservation-1---next (in-post-state) ((select_Reservation--next ((deref-oid_Reservation (in-post-state) (reconst-basetype))))))))))
```

```
end
overloading dot-1---client ≡ (dot-1---client::(·Reservation) ⇒ -)
begin
  definition dot_Reservation-1---client : (x::·Reservation) .client ≡ (eval-extract (x) ((deref-oid_Reservation
(in-post-state) ((deref-assocs_Reservation-1---client (in-post-state) ((select_Reservation--client ((deref-oid_Client (in-post-state)
(reconst-basetype))))))))))
end
overloading dot--id ≡ (dot--id::(·Reservation) ⇒ -)
begin
  definition dot_Reservation--id : (x::·Reservation) .id ≡ (eval-extract (x) ((deref-oid_Reservation (in-post-state)
((select_Reservation--id (reconst-basetype))))))
end
overloading dot--date ≡ (dot--date::(·Reservation) ⇒ -)
begin
  definition dot_Reservation--date : (x::·Reservation) .date ≡ (eval-extract (x) ((deref-oid_Reservation (in-post-state)
((select_Reservation--date (reconst-basetype))))))
end
overloading dot--flight ≡ (dot--flight::(·Reservation) ⇒ -)
begin
  definition dot_Reservation--flight : (x::·Reservation) .flight ≡ (eval-extract (x) ((deref-oid_Reservation (in-post-state)
((select_Reservation--flight ((deref-oid_Flight (in-post-state) (reconst-basetype))))))))
end
overloading dot-0---prevat-pre ≡ (dot-0---prevat-pre::(·Reservation) ⇒ -)
begin
  definition dot_Reservation-0---prevat-pre : (x::·Reservation) .prev@pre ≡ (eval-extract (x) ((deref-oid_Reservation
(in-pre-state) ((deref-assocs_Reservation-0---prev (in-pre-state) ((select_Reservation--prev ((deref-oid_Reservation (in-pre-state)
(reconst-basetype))))))))))
end
overloading dot-1---nextat-pre ≡ (dot-1---nextat-pre::(·Reservation) ⇒ -)
begin
  definition dot_Reservation-1---nextat-pre : (x::·Reservation) .next@pre ≡ (eval-extract (x) ((deref-oid_Reservation
(in-pre-state) ((deref-assocs_Reservation-1---next (in-pre-state) ((select_Reservation--next ((deref-oid_Reservation (in-pre-state)
(reconst-basetype))))))))))
end
overloading dot-1---clientat-pre ≡ (dot-1---clientat-pre::(·Reservation) ⇒ -)
begin
  definition dot_Reservation-1---clientat-pre : (x::·Reservation) .client@pre ≡ (eval-extract (x) ((deref-oid_Reservation
(in-pre-state) ((deref-assocs_Reservation-1---client (in-pre-state) ((select_Reservation--client ((deref-oid_Client (in-pre-state)
(reconst-basetype))))))))))
end
overloading dot--idat-pre ≡ (dot--idat-pre::(·Reservation) ⇒ -)
begin
  definition dot_Reservation--idat-pre : (x::·Reservation) .id@pre ≡ (eval-extract (x) ((deref-oid_Reservation (in-pre-state)
((select_Reservation--id (reconst-basetype))))))
end
overloading dot--dateat-pre ≡ (dot--dateat-pre::(·Reservation) ⇒ -)
begin
  definition dot_Reservation--dateat-pre : (x::·Reservation) .date@pre ≡ (eval-extract (x) ((deref-oid_Reservation (in-pre-state)
((select_Reservation--date (reconst-basetype))))))
end
overloading dot--flightat-pre ≡ (dot--flightat-pre::(·Reservation) ⇒ -)
begin
  definition dot_Reservation--flightat-pre : (x::·Reservation) .flight@pre ≡ (eval-extract (x) ((deref-oid_Reservation (in-pre-state)
((select_Reservation--flight ((deref-oid_Flight (in-pre-state) (reconst-basetype))))))))
end
overloading dot-0---flights ≡ (dot-0---flights::(·Client) ⇒ -)
begin
  definition dot_Client-0---flights : (x::·Client) .flights ≡ (eval-extract (x) ((deref-oid_Client (in-post-state)
((deref-assocs_Client-0---flights (in-post-state) ((select_Client--flights ((deref-oid_Flight (in-post-state) (reconst-basetype))))))))))
end
overloading dot--name ≡ (dot--name::(·Client) ⇒ -)
begin
  definition dot_Client--name : (x::·Client) .name ≡ (eval-extract (x) ((deref-oid_Client (in-post-state) ((select_Client--name
(reconst-basetype))))))
end
overloading dot-0---flightsat-pre ≡ (dot-0---flightsat-pre::(·Client) ⇒ -)
begin
  definition dot_Client-0---flightsat-pre : (x::·Client) .flights@pre ≡ (eval-extract (x) ((deref-oid_Client (in-pre-state)
((deref-assocs_Client-0---flights (in-pre-state) ((select_Client--flights ((deref-oid_Flight (in-pre-state) (reconst-basetype))))))))))
end
overloading dot--nameat-pre ≡ (dot--nameat-pre::(·Client) ⇒ -)
begin
  definition dot_Client--nameat-pre : (x::·Client) .name@pre ≡ (eval-extract (x) ((deref-oid_Client (in-pre-state)
```

((*select*$_{Client}$*--name* (*reconst-basetype*))))))
end
overloading *dot-0---flights* ≡ (*dot-0---flights*::(·*Staff*) ⇒ -)
begin
definition *dot*$_{Staff}$*-0---flights* : (*x*::·*Staff*) .*flights* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Staff}$ (*in-post-state*) ((*deref-assocs*$_{Staff}$*-0---flights* (*in-post-state*) ((*select*$_{Staff}$*--flights* ((*deref-oid*$_{Flight}$ (*in-post-state*) (*reconst-basetype*))))))))))
end
overloading *dot--name* ≡ (*dot--name*::(·*Staff*) ⇒ -)
begin
definition *dot*$_{Staff}$*--name* : (*x*::·*Staff*) .*name* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Staff}$ (*in-post-state*) ((*select*$_{Staff}$*--name* (*reconst-basetype*))))))
end
overloading *dot-0---flightsat-pre* ≡ (*dot-0---flightsat-pre*::(·*Staff*) ⇒ -)
begin
definition *dot*$_{Staff}$*-0---flightsat-pre* : (*x*::·*Staff*) .*flights*@*pre* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Staff}$ (*in-pre-state*) ((*deref-assocs*$_{Staff}$*-0---flights* (*in-pre-state*) ((*select*$_{Staff}$*--flights* ((*deref-oid*$_{Flight}$ (*in-pre-state*) (*reconst-basetype*))))))))))
end
overloading *dot--nameat-pre* ≡ (*dot--nameat-pre*::(·*Staff*) ⇒ -)
begin
definition *dot*$_{Staff}$*--nameat-pre* : (*x*::·*Staff*) .*name*@*pre* ≡ (*eval-extract* (*x*) ((*deref-oid*$_{Staff}$ (*in-pre-state*) ((*select*$_{Staff}$*--name* (*reconst-basetype*))))))
end

lemmas *dot-accessor* = *dot*$_{Flight}$*-1---passengers*
dot$_{Flight}$*--seats*
dot$_{Flight}$*--from*
dot$_{Flight}$*--to*
dot$_{Flight}$*--fl-res*
dot$_{Flight}$*-1---passengersat-pre*
dot$_{Flight}$*--seatsat-pre*
dot$_{Flight}$*--fromat-pre*
dot$_{Flight}$*--toat-pre*
dot$_{Flight}$*--fl-resat-pre*
dot$_{Client}$*-0---cl-res*
dot$_{Client}$*--address*
dot$_{Client}$*-0---cl-resat-pre*
dot$_{Client}$*--addressat-pre*
dot$_{Person}$*-0---flights*
dot$_{Person}$*--name*
dot$_{Person}$*-0---flightsat-pre*
dot$_{Person}$*--nameat-pre*
dot$_{Reservation}$*-0---prev*
dot$_{Reservation}$*-1---next*
dot$_{Reservation}$*-1---client*
dot$_{Reservation}$*--id*
dot$_{Reservation}$*--date*
dot$_{Reservation}$*--flight*
dot$_{Reservation}$*-0---prevat-pre*
dot$_{Reservation}$*-1---nextat-pre*
dot$_{Reservation}$*-1---clientat-pre*
dot$_{Reservation}$*--idat-pre*
dot$_{Reservation}$*--dateat-pre*
dot$_{Reservation}$*--flightat-pre*
dot$_{Client}$*-0---flights*
dot$_{Client}$*--name*
dot$_{Client}$*-0---flightsat-pre*
dot$_{Client}$*--nameat-pre*
dot$_{Staff}$*-0---flights*
dot$_{Staff}$*--name*
dot$_{Staff}$*-0---flightsat-pre*
dot$_{Staff}$*--nameat-pre*

Context Passing

lemmas[*simp*,*code-unfold*] = *eval-extract-def*

lemma *cp-dot*$_{Flight}$*-1---passengers* : (*cp* ((λ*X*. (*X*::·*Flight*) .*passengers*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--seats* : (*cp* ((λ*X*. (*X*::·*Flight*) .*seats*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--from* : (*cp* ((λ*X*. (*X*::·*Flight*) .*from*)))

by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--to* : (*cp* ((λ*X*. (*X*::·*Flight*) *.to*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--fl-res* : (*cp* ((λ*X*. (*X*::·*Flight*) *.fl-res*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*-1---passengersat-pre* : (*cp* ((λ*X*. (*X*::·*Flight*) *.passengers@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--seatsat-pre* : (*cp* ((λ*X*. (*X*::·*Flight*) *.seats@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--fromat-pre* : (*cp* ((λ*X*. (*X*::·*Flight*) *.from@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--toat-pre* : (*cp* ((λ*X*. (*X*::·*Flight*) *.to@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Flight}$*--fl-resat-pre* : (*cp* ((λ*X*. (*X*::·*Flight*) *.fl-res@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*-0---cl-res* : (*cp* ((λ*X*. (*X*::·*Client*) *.cl-res*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*--address* : (*cp* ((λ*X*. (*X*::·*Client*) *.address*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*-0---cl-resat-pre* : (*cp* ((λ*X*. (*X*::·*Client*) *.cl-res@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*--addressat-pre* : (*cp* ((λ*X*. (*X*::·*Client*) *.address@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Person}$*-0---flights* : (*cp* ((λ*X*. (*X*::·*Person*) *.flights*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Person}$*--name* : (*cp* ((λ*X*. (*X*::·*Person*) *.name*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Person}$*-0---flightsat-pre* : (*cp* ((λ*X*. (*X*::·*Person*) *.flights@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Person}$*--nameat-pre* : (*cp* ((λ*X*. (*X*::·*Person*) *.name@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*-0---prev* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.prev*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*-1---next* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.next*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*-1---client* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.client*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*--id* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.id*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*--date* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.date*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*--flight* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.flight*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*-0---prevat-pre* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.prev@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*-1---nextat-pre* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.next@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*-1---clientat-pre* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.client@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*--idat-pre* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.id@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*--dateat-pre* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.date@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Reservation}$*--flightat-pre* : (*cp* ((λ*X*. (*X*::·*Reservation*) *.flight@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*-0---flights* : (*cp* ((λ*X*. (*X*::·*Client*) *.flights*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*--name* : (*cp* ((λ*X*. (*X*::·*Client*) *.name*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*-0---flightsat-pre* : (*cp* ((λ*X*. (*X*::·*Client*) *.flights@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Client}$*--nameat-pre* : (*cp* ((λ*X*. (*X*::·*Client*) *.name@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Staff}$*-0---flights* : (*cp* ((λ*X*. (*X*::·*Staff*) *.flights*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Staff}$*--name* : (*cp* ((λ*X*. (*X*::·*Staff*) *.name*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Staff}$*-0---flightsat-pre* : (*cp* ((λ*X*. (*X*::·*Staff*) *.flights@pre*)))
by(*auto simp*: *dot-accessor cp-def*)
lemma *cp-dot*$_{Staff}$*--nameat-pre* : (*cp* ((λ*X*. (*X*::·*Staff*) *.name@pre*)))
by(*auto simp*: *dot-accessor cp-def*)

lemmas[*simp,code-unfold*] = *cp-dot$_{Flight}$-1---passengers*
cp-dot$_{Flight}$--seats
cp-dot$_{Flight}$--from
cp-dot$_{Flight}$--to
cp-dot$_{Flight}$--fl-res
cp-dot$_{Flight}$-1---passengersat-pre
cp-dot$_{Flight}$--seatsat-pre
cp-dot$_{Flight}$--fromat-pre
cp-dot$_{Flight}$--toat-pre
cp-dot$_{Flight}$--fl-resat-pre
cp-dot$_{Client}$-0---cl-res
cp-dot$_{Client}$--address
cp-dot$_{Client}$-0---cl-resat-pre
cp-dot$_{Client}$--addressat-pre
cp-dot$_{Person}$-0---flights
cp-dot$_{Person}$--name
cp-dot$_{Person}$-0---flightsat-pre
cp-dot$_{Person}$--nameat-pre
cp-dot$_{Reservation}$-0---prev
cp-dot$_{Reservation}$-1---next
cp-dot$_{Reservation}$-1---client
cp-dot$_{Reservation}$--id
cp-dot$_{Reservation}$--date
cp-dot$_{Reservation}$--flight
cp-dot$_{Reservation}$-0---prevat-pre
cp-dot$_{Reservation}$-1---nextat-pre
cp-dot$_{Reservation}$-1---clientat-pre
cp-dot$_{Reservation}$--idat-pre
cp-dot$_{Reservation}$--dateat-pre
cp-dot$_{Reservation}$--flightat-pre
cp-dot$_{Client}$-0---flights
cp-dot$_{Client}$--name
cp-dot$_{Client}$-0---flightsat-pre
cp-dot$_{Client}$--nameat-pre
cp-dot$_{Staff}$-0---flights
cp-dot$_{Staff}$--name
cp-dot$_{Staff}$-0---flightsat-pre
cp-dot$_{Staff}$--nameat-pre

Execution with Invalid or Null as Argument

lemma *dot$_{Flight}$-1---passengers-invalid* : (*invalid*::*Flight*) *.passengers* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$-1---passengers-null* : (*null*::*Flight*) *.passengers* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma *dot$_{Flight}$--seats-invalid* : (*invalid*::*Flight*) *.seats* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$--seats-null* : (*null*::*Flight*) *.seats* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma *dot$_{Flight}$--from-invalid* : (*invalid*::*Flight*) *.from* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$--from-null* : (*null*::*Flight*) *.from* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma *dot$_{Flight}$--to-invalid* : (*invalid*::*Flight*) *.to* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$--to-null* : (*null*::*Flight*) *.to* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma *dot$_{Flight}$--fl-res-invalid* : (*invalid*::*Flight*) *.fl-res* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$--fl-res-null* : (*null*::*Flight*) *.fl-res* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma *dot$_{Flight}$-1---passengersat-pre-invalid* : (*invalid*::*Flight*) *.passengers@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$-1---passengersat-pre-null* : (*null*::*Flight*) *.passengers@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma *dot$_{Flight}$--seatsat-pre-invalid* : (*invalid*::*Flight*) *.seats@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$--seatsat-pre-null* : (*null*::*Flight*) *.seats@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma *dot$_{Flight}$--fromat-pre-invalid* : (*invalid*::*Flight*) *.from@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma *dot$_{Flight}$--fromat-pre-null* : (*null*::*Flight*) *.from@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)

lemma dot_{Flight}--toat-pre-invalid : (*invalid*::·*Flight*) .*to*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Flight}--toat-pre-null : (*null*::·*Flight*) .*to*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Flight}--fl-resat-pre-invalid : (*invalid*::·*Flight*) .*fl-res*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Flight}--fl-resat-pre-null : (*null*::·*Flight*) .*fl-res*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}-0---cl-res-invalid : (*invalid*::·*Client*) .*cl-res* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}-0---cl-res-null : (*null*::·*Client*) .*cl-res* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}--address-invalid : (*invalid*::·*Client*) .*address* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}--address-null : (*null*::·*Client*) .*address* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}-0---cl-resat-pre-invalid : (*invalid*::·*Client*) .*cl-res*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}-0---cl-resat-pre-null : (*null*::·*Client*) .*cl-res*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}--addressat-pre-invalid : (*invalid*::·*Client*) .*address*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}--addressat-pre-null : (*null*::·*Client*) .*address*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Person}-0---flights-invalid : (*invalid*::·*Person*) .*flights* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Person}-0---flights-null : (*null*::·*Person*) .*flights* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Person}--name-invalid : (*invalid*::·*Person*) .*name* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Person}--name-null : (*null*::·*Person*) .*name* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Person}-0---flightsat-pre-invalid : (*invalid*::·*Person*) .*flights*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Person}-0---flightsat-pre-null : (*null*::·*Person*) .*flights*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Person}--nameat-pre-invalid : (*invalid*::·*Person*) .*name*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Person}--nameat-pre-null : (*null*::·*Person*) .*name*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$-0---prev-invalid : (*invalid*::·*Reservation*) .*prev* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$-0---prev-null : (*null*::·*Reservation*) .*prev* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$-1---next-invalid : (*invalid*::·*Reservation*) .*next* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$-1---next-null : (*null*::·*Reservation*) .*next* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$-1---client-invalid : (*invalid*::·*Reservation*) .*client* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$-1---client-null : (*null*::·*Reservation*) .*client* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$--id-invalid : (*invalid*::·*Reservation*) .*id* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$--id-null : (*null*::·*Reservation*) .*id* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$--date-invalid : (*invalid*::·*Reservation*) .*date* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$--date-null : (*null*::·*Reservation*) .*date* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$--flight-invalid : (*invalid*::·*Reservation*) .*flight* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$--flight-null : (*null*::·*Reservation*) .*flight* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$-0---prevat-pre-invalid : (*invalid*::·*Reservation*) .*prev*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$-0---prevat-pre-null : (*null*::·*Reservation*) .*prev*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$-1---nextat-pre-invalid : (*invalid*::·*Reservation*) .*next*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$-1---nextat-pre-null : (*null*::·*Reservation*) .*next*@*pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$-1---clientat-pre-invalid : (*invalid*::·*Reservation*) .*client*@*pre* = *invalid*

by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$*-1---clientat-pre-null* : (*null*::·*Reservation*) *.client@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$*--idat-pre-invalid* : (*invalid*::·*Reservation*) *.id@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$*--idat-pre-null* : (*null*::·*Reservation*) *.id@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$*--dateat-pre-invalid* : (*invalid*::·*Reservation*) *.date@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$*--dateat-pre-null* : (*null*::·*Reservation*) *.date@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma $dot_{Reservation}$*--flightat-pre-invalid* : (*invalid*::·*Reservation*) *.flight@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma $dot_{Reservation}$*--flightat-pre-null* : (*null*::·*Reservation*) *.flight@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}*-0---flights-invalid* : (*invalid*::·*Client*) *.flights* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}*-0---flights-null* : (*null*::·*Client*) *.flights* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}*--name-invalid* : (*invalid*::·*Client*) *.name* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}*--name-null* : (*null*::·*Client*) *.name* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}*-0---flightsat-pre-invalid* : (*invalid*::·*Client*) *.flights@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}*-0---flightsat-pre-null* : (*null*::·*Client*) *.flights@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Client}*--nameat-pre-invalid* : (*invalid*::·*Client*) *.name@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Client}*--nameat-pre-null* : (*null*::·*Client*) *.name@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Staff}*-0---flights-invalid* : (*invalid*::·*Staff*) *.flights* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Staff}*-0---flights-null* : (*null*::·*Staff*) *.flights* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Staff}*--name-invalid* : (*invalid*::·*Staff*) *.name* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Staff}*--name-null* : (*null*::·*Staff*) *.name* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Staff}*-0---flightsat-pre-invalid* : (*invalid*::·*Staff*) *.flights@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Staff}*-0---flightsat-pre-null* : (*null*::·*Staff*) *.flights@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)
lemma dot_{Staff}*--nameat-pre-invalid* : (*invalid*::·*Staff*) *.name@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def invalid-def*)
lemma dot_{Staff}*--nameat-pre-null* : (*null*::·*Staff*) *.name@pre* = *invalid*
by(*rule ext*, *simp add*: *dot-accessor bot-option-def null-fun-def null-option-def*)

Representation in States

lemma *defined-mono-dot*$_{Flight}$*-1---passengers* : $\tau \models (\delta\ ((X::\cdot Flight)\ .passengers)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .passengers)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* dot_{Flight}*-1---passengers-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .passengers)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* dot_{Flight}*-1---passengers-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--seats* : $\tau \models (\delta\ ((X::\cdot Flight)\ .seats)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .seats)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* dot_{Flight}*--seats-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .seats)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* dot_{Flight}*--seats-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--from* : $\tau \models (\delta\ ((X::\cdot Flight)\ .from)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .from)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* dot_{Flight}*--from-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .from)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* dot_{Flight}*--from-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--to* : $\tau \models (\delta\ ((X::\cdot Flight)\ .to)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .to)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* dot_{Flight}*--to-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .to)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp*

add: *foundation16′ dot*$_{Flight}$*--to-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--fl-res* : $\tau \models (\delta\ ((X::\cdot Flight)\ .fl\text{-}res)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .fl\text{-}res)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Flight}$*--fl-res-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .fl\text{-}res)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Flight}$*--fl-res-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*-1---passengersat-pre* : $\tau \models (\delta\ ((X::\cdot Flight)\ .passengers@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .passengers@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Flight}$*-1---passengersat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .passengers@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Flight}$*-1---passengersat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--seatsat-pre* : $\tau \models (\delta\ ((X::\cdot Flight)\ .seats@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .seats@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Flight}$*--seatsat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .seats@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Flight}$*--seatsat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--fromat-pre* : $\tau \models (\delta\ ((X::\cdot Flight)\ .from@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .from@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Flight}$*--fromat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .from@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Flight}$*--fromat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--toat-pre* : $\tau \models (\delta\ ((X::\cdot Flight)\ .to@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .to@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Flight}$*--toat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .to@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Flight}$*--toat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Flight}$*--fl-resat-pre* : $\tau \models (\delta\ ((X::\cdot Flight)\ .fl\text{-}res@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .fl\text{-}res@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Flight}$*--fl-resat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .fl\text{-}res@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Flight}$*--fl-resat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*-0---cl-res* : $\tau \models (\delta\ ((X::\cdot Client)\ .cl\text{-}res)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .cl\text{-}res)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Client}$*-0---cl-res-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .cl\text{-}res)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Client}$*-0---cl-res-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*--address* : $\tau \models (\delta\ ((X::\cdot Client)\ .address)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .address)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Client}$*--address-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .address)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Client}$*--address-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*-0---cl-resat-pre* : $\tau \models (\delta\ ((X::\cdot Client)\ .cl\text{-}res@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .cl\text{-}res@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Client}$*-0---cl-resat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .cl\text{-}res@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Client}$*-0---cl-resat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*--addressat-pre* : $\tau \models (\delta\ ((X::\cdot Client)\ .address@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .address@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Client}$*--addressat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .address@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Client}$*--addressat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Person}$*-0---flights* : $\tau \models (\delta\ ((X::\cdot Person)\ .flights)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Person}$*-0---flights-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot*$_{Person}$*-0---flights-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Person}$*--name* : $\tau \models (\delta\ ((X::\cdot Person)\ .name)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot*$_{Person}$*--name-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name)))$ and $\tau = \tau$ and $x = X$ and $y = null$],

simp add: *foundation16′ dot$_{Person}$--name-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Person}$-0---flightsat-pre* : $\tau \models (\delta\ ((X::\cdot Person)\ .flights@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Person}$-0---flightsat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Person}$-0---flightsat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Person}$--nameat-pre* : $\tau \models (\delta\ ((X::\cdot Person)\ .name@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Person}$--nameat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Person}$--nameat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$-0---prev* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .prev)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .prev)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$-0---prev-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .prev)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$-0---prev-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$-1---next* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .next)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .next)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$-1---next-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .next)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$-1---next-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$-1---client* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .client)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .client)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$-1---client-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .client)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$-1---client-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$--id* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .id)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .id)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$--id-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .id)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$--id-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$--date* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .date)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .date)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$--date-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .date)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$--date-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$--flight* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .flight)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flight)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$--flight-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flight)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$--flight-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$-0---prevat-pre* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .prev@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .prev@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$-0---prevat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .prev@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$-0---prevat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$-1---nextat-pre* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .next@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .next@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$-1---nextat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .next@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$-1---nextat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$-1---clientat-pre* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .client@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .client@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$-1---clientat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .client@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$-1---clientat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot$_{Reservation}$--idat-pre* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .id@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .id@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16′ dot$_{Reservation}$--idat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .id@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16′ dot$_{Reservation}$--idat-pre-null*)

by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Reservation}$*--dateat-pre* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .date@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .date@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Reservation}$*--dateat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .date@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Reservation}$*--dateat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Reservation}$*--flightat-pre* : $\tau \models (\delta\ ((X::\cdot Reservation)\ .flight@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flight@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Reservation}$*--flightat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flight@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Reservation}$*--flightat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*-0---flights* : $\tau \models (\delta\ ((X::\cdot Client)\ .flights)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Client}$*-0---flights-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Client}$*-0---flights-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*--name* : $\tau \models (\delta\ ((X::\cdot Client)\ .name)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Client}$*--name-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Client}$*--name-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*-0---flightsat-pre* : $\tau \models (\delta\ ((X::\cdot Client)\ .flights@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Client}$*-0---flightsat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Client}$*-0---flightsat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Client}$*--nameat-pre* : $\tau \models (\delta\ ((X::\cdot Client)\ .name@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Client}$*--nameat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Client}$*--nameat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Staff}$*-0---flights* : $\tau \models (\delta\ ((X::\cdot Staff)\ .flights)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Staff}$*-0---flights-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Staff}$*-0---flights-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Staff}$*--name* : $\tau \models (\delta\ ((X::\cdot Staff)\ .name)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Staff}$*--name-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Staff}$*--name-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Staff}$*-0---flightsat-pre* : $\tau \models (\delta\ ((X::\cdot Staff)\ .flights@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Staff}$*-0---flightsat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .flights@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Staff}$*-0---flightsat-pre-null*)
by(*simp add*: *defined-split*)
lemma *defined-mono-dot*$_{Staff}$*--nameat-pre* : $\tau \models (\delta\ ((X::\cdot Staff)\ .name@pre)) \Longrightarrow \tau \models (\delta\ (X))$
apply(*case-tac* $\tau \models (X \triangleq invalid)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name@pre)))$ and $\tau = \tau$ and $x = X$ and $y = invalid$], *simp add*: *foundation16'* *dot*$_{Staff}$*--nameat-pre-invalid*)
apply(*case-tac* $\tau \models (X \triangleq null)$, *insert StrongEq-L-subst2*[where $P = (\lambda x.\ (\delta\ (x\ .name@pre)))$ and $\tau = \tau$ and $x = X$ and $y = null$], *simp add*: *foundation16'* *dot*$_{Staff}$*--nameat-pre-null*)
by(*simp add*: *defined-split*)

lemma *is-repr-dot*$_{Reservation}$*-0---prev* :
assumes *def-dot*: $\tau \models (\delta\ ((X::\cdot Reservation)\ .prev))$
shows (*is-represented-in-state* (*in-post-state*) (*X* .*prev*) (*Reservation*) (τ))
apply(*insert defined-mono-dot*$_{Reservation}$*-0---prev*[*OF def-dot*, *simplified foundation16*])
apply(*case-tac* (*X* (τ)), *simp add*: *bot-option-def*)
proof − fix *a0* show (*X* (τ)) = (*Some* (*a0*)) $\Longrightarrow$ *?thesis* when (*X* (τ)) $\neq$ *null*
apply(*insert that*, *case-tac a0*, *simp add*: *null-option-def bot-option-def*, *clarify*)
proof − fix *a* show (*X* (τ)) = (*Some* ((*Some* (*a*)))) $\Longrightarrow$ *?thesis*
apply(*case-tac* (*heap* ((*in-post-state* (τ))) ((*oid-of* (*a*)))), *simp add*: *invalid-def bot-option-def*)

apply(*insert def-dot, simp add: dot$_{Reservation}$-0---prev is-represented-in-state-def select$_{Reservation}$--prev-def deref-oid$_{Reservation}$-def in-post-state-def defined-def OclValid-def false-def true-def invalid-def bot-fun-def split: split-if-asm*)
proof − fix *b* show (*X* (*τ*)) = (*Some* ((*Some* (*a*)))) ⟹ (*heap* ((*in-post-state* (*τ*))) ((*oid-of* (*a*)))) = (*Some* (*b*)) ⟹ *?thesis*
apply(*insert def-dot*[*simplified foundation16*], *auto simp: dot$_{Reservation}$-0---prev is-represented-in-state-def deref-oid$_{Reservation}$-def bot-option-def null-option-def*)
apply(*case-tac b, simp-all add: invalid-def bot-option-def*)
apply(*simp add: deref-assocs$_{Reservation}$-0---prev-def deref-assocs-def*)
apply(*case-tac* (*assocs* ((*in-post-state* (*τ*))) (*oid$_{Reservation}$-0---prev*)), *simp add: invalid-def bot-option-def, simp add: select$_{Reservation}$--prev-def*)
proof − fix *r typeoid* let *?t* = (*Some* ((*Some* (*r*)))) ∈ (*Some o OclAsType$_{Reservation}$-𝔄*) ‘ (*ran* ((*heap* ((*in-post-state* (*τ*))))))
let *?sel-any* = (*select-object-any$_{Set}$* ((*deref-oid$_{Reservation}$* (*in-post-state*) (*reconst-basetype*)))) show ((*?sel-any*) (*typeoid*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t*
proof − fix *aa* show ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t* when *τ* ⊨ (*δ* (((*?sel-any*) (*aa*))))
apply(*insert that, drule select-object-any-exec$_{Set}$*[*simplified foundation22*], *erule exE*)
proof − fix *e* show *?t* when ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) (((*?sel-any*) (*aa*) (*τ*)) = (*deref-oid$_{Reservation}$* (*in-post-state*) (*reconst-basetype*) (*e*) (*τ*))
apply(*insert that, simp add: deref-oid$_{Reservation}$-def*)
apply(*case-tac* (*heap* ((*in-post-state* (*τ*))) (*e*)), *simp add: invalid-def bot-option-def, simp*)
proof − fix *aaa* show (*case aaa of* (*in$_{Reservation}$* (*obj*)) ⇒ (*reconst-basetype* (*obj*) (*τ*))
| - ⇒ (*invalid* (*τ*))) = (*Some* ((*Some* (*r*)))) ⟹ (*heap* ((*in-post-state* (*τ*))) (*e*)) = (*Some* (*aaa*)) ⟹ *?t*
apply(*case-tac aaa, auto simp: invalid-def bot-option-def image-def ran-def*)
apply(*rule exI*[where *x* = (*in$_{Reservation}$* (*r*))], *simp add: OclAsType$_{Reservation}$-𝔄-def Let-def reconst-basetype-def split: split-if-asm*)
by(*rule*) qed
apply-end((*blast*)+)
qed
apply-end(*simp add: foundation16 bot-option-def null-option-def*)
qed qed qed qed
apply-end(*simp-all*)
qed
lemma *is-repr-dot$_{Reservation}$-1---next* :
assumes *def-dot*: *τ* ⊨ (*δ* ((*X*::*Reservation*) *.next*))
shows (*is-represented-in-state* (*in-post-state*) (*X .next*) (*Reservation*) (*τ*))
apply(*insert defined-mono-dot$_{Reservation}$-1---next*[*OF def-dot, simplified foundation16*])
apply(*case-tac* (*X* (*τ*)), *simp add: bot-option-def*)
proof − fix *a0* show (*X* (*τ*)) = (*Some* (*a0*)) ⟹ *?thesis* when (*X* (*τ*)) ≠ *null*
apply(*insert that, case-tac a0, simp add: null-option-def bot-option-def, clarify*)
proof − fix *a* show (*X* (*τ*)) = (*Some* ((*Some* (*a*)))) ⟹ *?thesis*
apply(*case-tac* (*heap* ((*in-post-state* (*τ*))) ((*oid-of* (*a*)))), *simp add: invalid-def bot-option-def*)
apply(*insert def-dot, simp add: dot$_{Reservation}$-1---next is-represented-in-state-def select$_{Reservation}$--next-def deref-oid$_{Reservation}$-def in-post-state-def defined-def OclValid-def false-def true-def invalid-def bot-fun-def split: split-if-asm*)
proof − fix *b* show (*X* (*τ*)) = (*Some* ((*Some* (*a*)))) ⟹ (*heap* ((*in-post-state* (*τ*))) ((*oid-of* (*a*)))) = (*Some* (*b*)) ⟹ *?thesis*
apply(*insert def-dot*[*simplified foundation16*], *auto simp: dot$_{Reservation}$-1---next is-represented-in-state-def deref-oid$_{Reservation}$-def bot-option-def null-option-def*)
apply(*case-tac b, simp-all add: invalid-def bot-option-def*)
apply(*simp add: deref-assocs$_{Reservation}$-1---next-def deref-assocs-def*)
apply(*case-tac* (*assocs* ((*in-post-state* (*τ*))) (*oid$_{Reservation}$-1---next*)), *simp add: invalid-def bot-option-def, simp add: select$_{Reservation}$--next-def*)
proof − fix *r typeoid* let *?t* = (*Some* ((*Some* (*r*)))) ∈ (*Some o OclAsType$_{Reservation}$-𝔄*) ‘ (*ran* ((*heap* ((*in-post-state* (*τ*))))))
let *?sel-any* = (*select-object-any$_{Set}$* ((*deref-oid$_{Reservation}$* (*in-post-state*) (*reconst-basetype*)))) show ((*?sel-any*) (*typeoid*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t*
proof − fix *aa* show ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t* when *τ* ⊨ (*δ* (((*?sel-any*) (*aa*))))
apply(*insert that, drule select-object-any-exec$_{Set}$*[*simplified foundation22*], *erule exE*)
proof − fix *e* show *?t* when ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) (((*?sel-any*) (*aa*) (*τ*)) = (*deref-oid$_{Reservation}$* (*in-post-state*) (*reconst-basetype*) (*e*) (*τ*))
apply(*insert that, simp add: deref-oid$_{Reservation}$-def*)
apply(*case-tac* (*heap* ((*in-post-state* (*τ*))) (*e*)), *simp add: invalid-def bot-option-def, simp*)
proof − fix *aaa* show (*case aaa of* (*in$_{Reservation}$* (*obj*)) ⇒ (*reconst-basetype* (*obj*) (*τ*))
| - ⇒ (*invalid* (*τ*))) = (*Some* ((*Some* (*r*)))) ⟹ (*heap* ((*in-post-state* (*τ*))) (*e*)) = (*Some* (*aaa*)) ⟹ *?t*
apply(*case-tac aaa, auto simp: invalid-def bot-option-def image-def ran-def*)
apply(*rule exI*[where *x* = (*in$_{Reservation}$* (*r*))], *simp add: OclAsType$_{Reservation}$-𝔄-def Let-def reconst-basetype-def split: split-if-asm*)
by(*rule*) qed
apply-end((*blast*)+)
qed
apply-end(*simp add: foundation16 bot-option-def null-option-def*)
qed qed qed qed
apply-end(*simp-all*)
qed
lemma *is-repr-dot$_{Reservation}$-1---client* :
assumes *def-dot*: *τ* ⊨ (*δ* ((*X*::*Reservation*) *.client*))

shows (*is-represented-in-state* (*in-post-state*) (*X .client*) (*Client*) (*τ*))
apply(*insert defined-mono-dot*$_{Reservation}$*-1---client*[*OF def-dot*, *simplified foundation16*])
apply(*case-tac* (*X* (*τ*)), *simp add*: *bot-option-def*)
proof − fix *a0* show (*X* (*τ*)) = (*Some* (*a0*)) ⟹ *?thesis* when (*X* (*τ*)) ≠ *null*
apply(*insert that*, *case-tac a0*, *simp add*: *null-option-def bot-option-def*, *clarify*)
proof − fix *a* show (*X* (*τ*)) = (*Some* ((*Some* (*a*)))) ⟹ *?thesis*
apply(*case-tac* (*heap* ((*in-post-state* (*τ*))) ((*oid-of* (*a*)))), *simp add*: *invalid-def bot-option-def*)
apply(*insert def-dot*, *simp add*: *dot*$_{Reservation}$*-1---client is-represented-in-state-def select*$_{Reservation}$*--client-def deref-oid*$_{Reservation}$*-def in-post-state-def defined-def OclValid-def false-def true-def invalid-def bot-fun-def split*: *split-if-asm*)
proof − fix *b* show (*X* (*τ*)) = (*Some* ((*Some* (*a*)))) ⟹ (*heap* ((*in-post-state* (*τ*))) ((*oid-of* (*a*)))) = (*Some* (*b*)) ⟹ *?thesis*
apply(*insert def-dot*[*simplified foundation16*], *auto simp*: *dot*$_{Reservation}$*-1---client is-represented-in-state-def deref-oid*$_{Reservation}$*-def bot-option-def null-option-def*)
apply(*case-tac b*, *simp-all add*: *invalid-def bot-option-def*)
apply(*simp add*: *deref-assocs*$_{Reservation}$*-1---client-def deref-assocs-def*)
apply(*case-tac* (*assocs* ((*in-post-state* (*τ*))) (*oid*$_{Reservation}$*-1---client*)), *simp add*: *invalid-def bot-option-def*, *simp add*: *select*$_{Reservation}$*--client-def*)
proof − fix *r typeoid* let *?t* = (*Some* ((*Some* (*r*)))) ∈ (*Some o OclAsType*$_{Client}$-𝔄) ‘ (*ran* ((*heap* ((*in-post-state* (*τ*))))))
let *?sel-any* = (*select-object-any*$_{Set}$ ((*deref-oid*$_{Client}$ (*in-post-state*) (*reconst-basetype*)))) show ((*?sel-any*) (*typeoid*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t*
proof − fix *aa* show ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t* when *τ* ⊨ (*δ* (((*?sel-any*) (*aa*))))
apply(*insert that*, *drule select-object-any-exec*$_{Set}$[*simplified foundation22*], *erule exE*)
proof − fix *e* show *?t* when ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) ((*?sel-any*) (*aa*) (*τ*)) = (*deref-oid*$_{Client}$ (*in-post-state*) (*reconst-basetype*) (*e*) (*τ*))
apply(*insert that*, *simp add*: *deref-oid*$_{Client}$*-def*)
apply(*case-tac* (*heap* ((*in-post-state* (*τ*))) (*e*)), *simp add*: *invalid-def bot-option-def*, *simp*)
proof − fix *aaa* show (*case aaa of* (*in*$_{Client}$ (*obj*)) ⇒ (*reconst-basetype* (*obj*) (*τ*))
| - ⇒ (*invalid* (*τ*))) = (*Some* ((*Some* (*r*)))) ⟹ (*heap* ((*in-post-state* (*τ*))) (*e*)) = (*Some* (*aaa*)) ⟹ *?t*
apply(*case-tac aaa*, *auto simp*: *invalid-def bot-option-def image-def ran-def*)
apply(*rule exI*[where *x* = (*in*$_{Client}$ (*r*))], *simp add*: *OclAsType*$_{Client}$*-𝔄-def Let-def reconst-basetype-def split*: *split-if-asm*)
by(*rule*) qed
apply-end((*blast*)+)
qed
apply-end(*simp add*: *foundation16 bot-option-def null-option-def*)
qed qed qed qed
apply-end(*simp-all*)
qed
lemma *is-repr-dot*$_{Reservation}$*-0---prevat-pre* :
assumes *def-dot*: *τ* ⊨ (*δ* ((*X*::*Reservation*) *.prev@pre*))
shows (*is-represented-in-state* (*in-pre-state*) (*X .prev@pre*) (*Reservation*) (*τ*))
apply(*insert defined-mono-dot*$_{Reservation}$*-0---prevat-pre*[*OF def-dot*, *simplified foundation16*])
apply(*case-tac* (*X* (*τ*)), *simp add*: *bot-option-def*)
proof − fix *a0* show (*X* (*τ*)) = (*Some* (*a0*)) ⟹ *?thesis* when (*X* (*τ*)) ≠ *null*
apply(*insert that*, *case-tac a0*, *simp add*: *null-option-def bot-option-def*, *clarify*)
proof − fix *a* show (*X* (*τ*)) = (*Some* ((*Some* (*a*)))) ⟹ *?thesis*
apply(*case-tac* (*heap* ((*in-pre-state* (*τ*))) ((*oid-of* (*a*)))), *simp add*: *invalid-def bot-option-def*)
apply(*insert def-dot*, *simp add*: *dot*$_{Reservation}$*-0---prevat-pre is-represented-in-state-def select*$_{Reservation}$*--prev-def deref-oid*$_{Reservation}$*-def in-pre-state-def defined-def OclValid-def false-def true-def invalid-def bot-fun-def split*: *split-if-asm*)
proof − fix *b* show (*X* (*τ*)) = (*Some* ((*Some* (*a*)))) ⟹ (*heap* ((*in-pre-state* (*τ*))) ((*oid-of* (*a*)))) = (*Some* (*b*)) ⟹ *?thesis*
apply(*insert def-dot*[*simplified foundation16*], *auto simp*: *dot*$_{Reservation}$*-0---prevat-pre is-represented-in-state-def deref-oid*$_{Reservation}$*-def bot-option-def null-option-def*)
apply(*case-tac b*, *simp-all add*: *invalid-def bot-option-def*)
apply(*simp add*: *deref-assocs*$_{Reservation}$*-0---prev-def deref-assocs-def*)
apply(*case-tac* (*assocs* ((*in-pre-state* (*τ*))) (*oid*$_{Reservation}$*-0---prev*)), *simp add*: *invalid-def bot-option-def*, *simp add*: *select*$_{Reservation}$*--prev-def*)
proof − fix *r typeoid* let *?t* = (*Some* ((*Some* (*r*)))) ∈ (*Some o OclAsType*$_{Reservation}$-𝔄) ‘ (*ran* ((*heap* ((*in-pre-state* (*τ*))))))
let *?sel-any* = (*select-object-any*$_{Set}$ ((*deref-oid*$_{Reservation}$ (*in-pre-state*) (*reconst-basetype*)))) show ((*?sel-any*) (*typeoid*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t*
proof − fix *aa* show ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) ⟹ *?t* when *τ* ⊨ (*δ* (((*?sel-any*) (*aa*))))
apply(*insert that*, *drule select-object-any-exec*$_{Set}$[*simplified foundation22*], *erule exE*)
proof − fix *e* show *?t* when ((*?sel-any*) (*aa*) (*τ*)) = (*Some* ((*Some* (*r*)))) ((*?sel-any*) (*aa*) (*τ*)) = (*deref-oid*$_{Reservation}$ (*in-pre-state*) (*reconst-basetype*) (*e*) (*τ*))
apply(*insert that*, *simp add*: *deref-oid*$_{Reservation}$*-def*)
apply(*case-tac* (*heap* ((*in-pre-state* (*τ*))) (*e*)), *simp add*: *invalid-def bot-option-def*, *simp*)
proof − fix *aaa* show (*case aaa of* (*in*$_{Reservation}$ (*obj*)) ⇒ (*reconst-basetype* (*obj*) (*τ*))
| - ⇒ (*invalid* (*τ*))) = (*Some* ((*Some* (*r*)))) ⟹ (*heap* ((*in-pre-state* (*τ*))) (*e*)) = (*Some* (*aaa*)) ⟹ *?t*
apply(*case-tac aaa*, *auto simp*: *invalid-def bot-option-def image-def ran-def*)
apply(*rule exI*[where *x* = (*in*$_{Reservation}$ (*r*))], *simp add*: *OclAsType*$_{Reservation}$*-𝔄-def Let-def reconst-basetype-def split*: *split-if-asm*)
by(*rule*) qed
apply-end((*blast*)+)
qed
apply-end(*simp add*: *foundation16 bot-option-def null-option-def*)

```
 qed qed qed qed
  apply-end(simp-all)
 qed
lemma is-repr-dot_Reservation-1---nextat-pre :
assumes def-dot: τ ⊨ (δ ((X::·Reservation) .next@pre))
shows (is-represented-in-state (in-pre-state) (X .next@pre) (Reservation) (τ))
  apply(insert defined-mono-dot_Reservation-1---nextat-pre[OF def-dot, simplified foundation16])
  apply(case-tac (X (τ)), simp add: bot-option-def)
  proof − fix a0 show (X (τ)) = (Some (a0)) ⟹ ?thesis when (X (τ)) ≠ null
  apply(insert that, case-tac a0, simp add: null-option-def bot-option-def, clarify)
  proof − fix a show (X (τ)) = (Some ((Some (a)))) ⟹ ?thesis
  apply(case-tac (heap ((in-pre-state (τ))) ((oid-of (a)))), simp add: invalid-def bot-option-def)
    apply(insert def-dot, simp add: dot_Reservation-1---nextat-pre is-represented-in-state-def select_Reservation--next-def
deref-oid_Reservation-def in-pre-state-def defined-def OclValid-def false-def true-def invalid-def bot-fun-def split: split-if-asm)
  proof − fix b show (X (τ)) = (Some ((Some (a)))) ⟹ (heap ((in-pre-state (τ))) ((oid-of (a)))) = (Some (b)) ⟹ ?thesis
    apply(insert def-dot[simplified foundation16], auto simp: dot_Reservation-1---nextat-pre is-represented-in-state-def
deref-oid_Reservation-def bot-option-def null-option-def)
  apply(case-tac b, simp-all add: invalid-def bot-option-def)
  apply(simp add: deref-assocs_Reservation-1---next-def deref-assocs-def)
   apply(case-tac (assocs ((in-pre-state (τ))) (oid_Reservation-1---next)), simp add: invalid-def bot-option-def, simp add: se-
lect_Reservation--next-def)
  proof − fix r typeid  let ?t = (Some ((Some (r)))) ∈ (Some o OclAsType_Reservation-𝔄) ‘ (ran ((heap ((in-pre-state
(τ))))))
      let ?sel-any = (select-object-any_Set ((deref-oid_Reservation (in-pre-state) (reconst-basetype)))) show ((?sel-any) (typeid)
(τ)) = (Some ((Some (r)))) ⟹ ?t
  proof − fix aa show ((?sel-any) (aa) (τ)) = (Some ((Some (r)))) ⟹ ?t when τ ⊨ (δ (((?sel-any) (aa))))
  apply(insert that, drule select-object-any-exec_Set[simplified foundation22], erule exE)
  proof − fix e show ?t when ((?sel-any) (aa) (τ)) = (Some ((Some (r)))) ((?sel-any) (aa) (τ)) = (deref-oid_Reservation
(in-pre-state) (reconst-basetype) (e) (τ))
  apply(insert that, simp add: deref-oid_Reservation-def)
  apply(case-tac (heap ((in-pre-state (τ))) (e)), simp add: invalid-def bot-option-def, simp)
  proof − fix aaa show (case aaa of (in_Reservation (obj)) ⇒ (reconst-basetype (obj) (τ))
    | - ⇒ (invalid (τ))) = (Some ((Some (r)))) ⟹ (heap ((in-pre-state (τ))) (e)) = (Some (aaa)) ⟹ ?t
  apply(case-tac aaa, auto simp: invalid-def bot-option-def image-def ran-def)
   apply(rule exI[where x = (in_Reservation (r))], simp add: OclAsType_Reservation-𝔄-def Let-def reconst-basetype-def split:
split-if-asm)
by(rule) qed
  apply-end((blast)+)
 qed
  apply-end(simp add: foundation16 bot-option-def null-option-def)
 qed qed qed qed
  apply-end(simp-all)
 qed
lemma is-repr-dot_Reservation-1---clientat-pre :
assumes def-dot: τ ⊨ (δ ((X::·Reservation) .client@pre))
shows (is-represented-in-state (in-pre-state) (X .client@pre) (Client) (τ))
  apply(insert defined-mono-dot_Reservation-1---clientat-pre[OF def-dot, simplified foundation16])
  apply(case-tac (X (τ)), simp add: bot-option-def)
  proof − fix a0 show (X (τ)) = (Some (a0)) ⟹ ?thesis when (X (τ)) ≠ null
  apply(insert that, case-tac a0, simp add: null-option-def bot-option-def, clarify)
  proof − fix a show (X (τ)) = (Some ((Some (a)))) ⟹ ?thesis
  apply(case-tac (heap ((in-pre-state (τ))) ((oid-of (a)))), simp add: invalid-def bot-option-def)
    apply(insert def-dot, simp add: dot_Reservation-1---clientat-pre is-represented-in-state-def select_Reservation--client-def
deref-oid_Reservation-def in-pre-state-def defined-def OclValid-def false-def true-def invalid-def bot-fun-def split: split-if-asm)
  proof − fix b show (X (τ)) = (Some ((Some (a)))) ⟹ (heap ((in-pre-state (τ))) ((oid-of (a)))) = (Some (b)) ⟹ ?thesis
    apply(insert def-dot[simplified foundation16], auto simp: dot_Reservation-1---clientat-pre is-represented-in-state-def
deref-oid_Reservation-def bot-option-def null-option-def)
  apply(case-tac b, simp-all add: invalid-def bot-option-def)
  apply(simp add: deref-assocs_Reservation-1---client-def deref-assocs-def)
   apply(case-tac (assocs ((in-pre-state (τ))) (oid_Reservation-1---client)), simp add: invalid-def bot-option-def, simp add: se-
lect_Reservation--client-def)
  proof − fix r typeid  let ?t = (Some ((Some (r)))) ∈ (Some o OclAsType_Client-𝔄) ‘ (ran ((heap ((in-pre-state (τ))))))
      let ?sel-any = (select-object-any_Set ((deref-oid_Client (in-pre-state) (reconst-basetype)))) show ((?sel-any) (typeid) (τ))
= (Some ((Some (r)))) ⟹ ?t
  proof − fix aa show ((?sel-any) (aa) (τ)) = (Some ((Some (r)))) ⟹ ?t when τ ⊨ (δ (((?sel-any) (aa))))
  apply(insert that, drule select-object-any-exec_Set[simplified foundation22], erule exE)
  proof − fix e show ?t when ((?sel-any) (aa) (τ)) = (Some ((Some (r)))) ((?sel-any) (aa) (τ)) = (deref-oid_Client (in-pre-state)
(reconst-basetype) (e) (τ))
  apply(insert that, simp add: deref-oid_Client-def)
  apply(case-tac (heap ((in-pre-state (τ))) (e)), simp add: invalid-def bot-option-def, simp)
  proof − fix aaa show (case aaa of (in_Client (obj)) ⇒ (reconst-basetype (obj) (τ))
    | - ⇒ (invalid (τ))) = (Some ((Some (r)))) ⟹ (heap ((in-pre-state (τ))) (e)) = (Some (aaa)) ⟹ ?t
  apply(case-tac aaa, auto simp: invalid-def bot-option-def image-def ran-def)
```

apply(*rule exI*[where $x = (in_{Client}\ (r))$], *simp add*: $OclAsType_{Client}$-$\mathfrak{A}$-*def Let-def reconst-basetype-def split*: *split-if-asm*)
by(*rule*) qed
apply-end((*blast*)+)
qed
apply-end(*simp add*: *foundation16 bot-option-def null-option-def*)
qed qed qed qed
apply-end(*simp-all*)
qed

B.9 Class Model: Towards the Object Instances

lemmas [*simp*,*code-unfold*] = *state.defs*
const-ss

lemmas[*simp*,*code-unfold*] = $OclAsType_{Flight}$-*OclAny*
$OclAsType_{Flight}$-*Staff*
$OclAsType_{Flight}$-*Person*
$OclAsType_{Flight}$-*Client*
$OclAsType_{Flight}$-*Reservation*
$OclAsType_{Client}$-*Person*
$OclAsType_{Client}$-*OclAny*
$OclAsType_{Client}$-*Staff*
$OclAsType_{Client}$-*Reservation*
$OclAsType_{Client}$-*Flight*
$OclAsType_{Staff}$-*Person*
$OclAsType_{Staff}$-*OclAny*
$OclAsType_{Staff}$-*Client*
$OclAsType_{Staff}$-*Reservation*
$OclAsType_{Staff}$-*Flight*
$OclAsType_{Person}$-*OclAny*
$OclAsType_{Person}$-*Client*
$OclAsType_{Person}$-*Staff*
$OclAsType_{Person}$-*Reservation*
$OclAsType_{Person}$-*Flight*
$OclAsType_{Reservation}$-*OclAny*
$OclAsType_{Reservation}$-*Staff*
$OclAsType_{Reservation}$-*Person*
$OclAsType_{Reservation}$-*Client*
$OclAsType_{Reservation}$-*Flight*
$OclAsType_{OclAny}$-*Flight*
$OclAsType_{OclAny}$-*Client*
$OclAsType_{OclAny}$-*Staff*
$OclAsType_{OclAny}$-*Person*
$OclAsType_{OclAny}$-*Reservation*

B.10 Instance

definition (*typecheck-instance-bad-head-on-lhs-F2-F1-R21-R11-C2-C1-S1* (*F2*) (*F1*) (*R21*) (*R11*) (*C2*) (*C1*) (*S1*)) = ()
definition *typecheck-instance-extra-variables-on-rhs-F2-F1-R21-R11-C2-C1-S1* = (λ*F2 F1 R21 R11 C2 C1 S1*. (*F1* , *Mon* , *F1* , *R21* , *F1* , *R11* , *F1* , *F1*))

definition *oid3* = *3*
definition *oid4* = *4*
definition *oid5* = *5*
definition *oid6* = *6*
definition *oid7* = *7*
definition *oid8* = *8*
definition *oid9* = *9*

definition $S1_{Staff}$ = (mk_{Staff} (($mk\mathcal{EXT}_{Staff}$ (*oid3*) (⌊''*Merlin*''⌋))))
definition (*S1*::·*Staff*) = ((λ-. ⌊⌊$S1_{Staff}$⌋⌋))
definition $C1_{Client}$ = (mk_{Client} (($mk\mathcal{EXT}_{Client}$ (*oid4*) (⌊''*Bertha*''⌋))) (⌊''*Miami*''⌋))
definition (*C1*::·*Client*) = ((λ-. ⌊⌊$C1_{Client}$⌋⌋))
definition $C2_{Client}$ = (mk_{Client} (($mk\mathcal{EXT}_{Client}$ (*oid5*) (⌊''*Arthur*''⌋))) (⌊''*Valencia*''⌋))
definition (*C2*::·*Client*) = ((λ-. ⌊⌊$C2_{Client}$⌋⌋))
definition $R11_{Reservation}$ = ($mk_{Reservation}$ (($mk\mathcal{EXT}_{Reservation}$ (*oid6*))) (⌊*12345*⌋) (⌊$constr_{Mon}$⌋) (⌊*oid8*⌋))
definition (*R11*::·*Reservation*) = ((λ-. ⌊⌊$R11_{Reservation}$⌋⌋))
definition $R21_{Reservation}$ = ($mk_{Reservation}$ (($mk\mathcal{EXT}_{Reservation}$ (*oid7*))) (⌊*98765*⌋) (*None*) (⌊*oid8*⌋))

definition $(R21::\cdot Reservation) = ((\lambda\text{-}.\ \lfloor\lfloor R21_{Reservation}\rfloor\rfloor))$
definition $F1_{Flight} = (mk_{Flight}\ ((mk\mathcal{EXT}_{Flight}\ (oid8)))\ (\lfloor 120\rfloor)\ (\lfloor ''Valencia''\rfloor)\ (\lfloor ''Miami''\rfloor)\ (None))$
definition $(F1::\cdot Flight) = ((\lambda\text{-}.\ \lfloor\lfloor F1_{Flight}\rfloor\rfloor))$
definition $F2_{Flight} = (mk_{Flight}\ ((mk\mathcal{EXT}_{Flight}\ (oid9)))\ (\lfloor 370\rfloor)\ (\lfloor ''Miami''\rfloor)\ (\lfloor ''Ottawa''\rfloor)\ (None))$
definition $(F2::\cdot Flight) = ((\lambda\text{-}.\ \lfloor\lfloor F2_{Flight}\rfloor\rfloor))$

ML ‹(*Ty'.check* ([(*META.Writeln* , *S1 .flights* ≅ *Set{ F1 }*) , (*META.Writeln* , *C1 .flights* ≅ *Set{ F1 }*) , (*META.Writeln* , *C1 .cl-res* ≅ *Set{ R11 }*) , (*META.Writeln* , *C2 .flights* ≅ *Set{ F1 }*) , (*META.Writeln* , *C2 .cl-res* ≅ *Set{ R21 }*) , (*META.Writeln* , *R11 .flight* ≅ *Set{ F1 }*) , (*META.Writeln* , *R11 .client* ≅ *Set{ C1 }*) , (*META.Writeln* , *R11 .prev* ≅ *Set{}*) , (*META.Writeln* , *R11 .next* ≅ *Set{}*) , (*META.Writeln* , *R21 .flight* ≅ *Set{ F1 }*) , (*META.Writeln* , *R21 .client* ≅ *Set{ C2 }*) , (*META.Writeln* , *R21 .prev* ≅ *Set{}*) , (*META.Writeln* , *R21 .next* ≅ *Set{}*) , (*META.Writeln* , *F1 .passengers* ≅ *Set{ S1 , C1 , C2 }*) , (*META.Writeln* , *F1 .fl-res* ≅ *Set{ R11 , R21 }*) , (*META.Writeln* , *F2 .passengers* ≅ *Set{}*) , (*META.Writeln* , *F2 .fl-res* ≅ *Set{}*)]) (*error*(*s*)))›

B.11 State (Floor 1)

definition (*typecheck-state-bad-head-on-lhs-*σ_1 (σ_1)) = ()
definition *typecheck-state-extra-variables-on-rhs-*σ_1 = (*F2* , *F1* , *R21* , *R11* , *C2* , *C1* , *S1*)

generation-syntax [shallow]
setup ‹(*Generation-mode.update-compiler-config* ((*K* (*let open META in Compiler-env-config-ext* (*true*, *NONE*, *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 10*)), *I* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 0*)), *Gen-default*, *SOME* (*OclClass* ((*META.SS-base* (*META.ST OclAny*)), *nil*, *uncurry cons* (*OclClass* ((*META.SS-base* (*META.ST Reservation*)), *uncurry cons* (*I* ((*META.SS-base* (*META.ST prev*)), *OclTy-object* (*OclTyObj* (*OclTyCore* (*Ocl-ty-class-ext* ((*META.SS-base* (*META.ST oid*)), (*Code-Numeral.Nat 2*), (*Code-Numeral.Nat 2*), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 0*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 0*)), *SOME* (*Mult-nat* ((*Code-Numeral.Nat 1*)))), *nil*), *SOME* ((*META.SS-base* (*META.ST next*))), *nil*, ()), (*META.SS-base* (*META.ST Reservation*)), ()), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 1*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 0*)), *SOME* (*Mult-nat* ((*Code-Numeral.Nat 1*)))), *nil*), *SOME* ((*META.SS-base* (*META.ST prev*))), *nil*, ()), (*META.SS-base* (*META.ST Reservation*)), ()), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 0*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 0*)), *SOME* (*Mult-nat* ((*Code-Numeral.Nat 1*)))), *nil*), *SOME* ((*META.SS-base* (*META.ST next*))), *nil*, ()), (*META.SS-base* (*META.ST Reservation*)), ()), ()))), *nil*))), *uncurry cons* (*I* ((*META.SS-base* (*META.ST next*)), *OclTy-object* (*OclTyObj* (*OclTyCore* (*Ocl-ty-class-ext* ((*META.SS-base* (*META.ST oid*)), (*Code-Numeral.Nat 2*), (*Code-Numeral.Nat 2*), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 1*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 0*)), *SOME* (*Mult-nat* ((*Code-Numeral.Nat 1*)))), *nil*), *SOME* ((*META.SS-base* (*META.ST prev*))), *nil*, ()), (*META.SS-base* (*META.ST Reservation*)), ()), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 0*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 0*)), *SOME* (*Mult-nat* ((*Code-Numeral.Nat 1*)))), *nil*), *SOME* ((*META.SS-base* (*META.ST next*))), *nil*, ()), (*META.SS-base* (*META.ST Reservation*)), ()), ()))), *nil*))), *uncurry cons* (*I* ((*META.SS-base* (*META.ST client*)), *OclTy-object* (*OclTyObj* (*OclTyCore* (*Ocl-ty-class-ext* ((*META.SS-base* (*META.ST oid*)), (*Code-Numeral.Nat 1*), (*Code-Numeral.Nat 2*), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 1*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST cl-res*))), *nil*, ()), (*META.SS-base* (*META.ST Reservation*)), ()), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 0*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 1*)), *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST client*))), *nil*, ()), (*META.SS-base* (*META.ST Client*)), ()), ()))), *nil*))), *uncurry cons* (*I* ((*META.SS-base* (*META.ST id*)), *OclTy-base-integer*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST date*)), *OclTy-enum* ((*META.SS-base* (*META.ST Week*)))), *uncurry cons* (*I* ((*META.SS-base* (*META.ST flight*)), *OclTy-object* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Flight*))), *nil*))), *nil*))))))), *nil*), *uncurry cons* (*OclClass* ((*META.SS-base* (*META.ST Person*)), *uncurry cons* (*I* ((*META.SS-base* (*META.ST flights*)), *OclTy-object* (*OclTyObj* (*OclTyCore* (*Ocl-ty-class-ext* ((*META.SS-base* (*META.ST oid*)), (*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 2*), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 0*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST passengers*))), *nil*, ()), (*META.SS-base* (*META.ST Person*)), ()), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 1*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST flights*))), *nil*, ()), (*META.SS-base* (*META.ST Flight*)), ()), ()))), *nil*))), *uncurry cons* (*I* ((*META.SS-base* (*META.ST name*)), *OclTy-base-string*), *nil*)), *uncurry cons* (*OclClass* ((*META.SS-base* (*META.ST Staff*)), *nil*, *nil*), *uncurry cons* (*OclClass* ((*META.SS-base* (*META.ST Client*)), *uncurry cons* (*I* ((*META.SS-base* (*META.ST cl-res*)), *OclTy-object* (*OclTyObj* (*OclTyCore* (*Ocl-ty-class-ext* ((*META.SS-base* (*META.ST oid*)), (*Code-Numeral.Nat 1*), (*Code-Numeral.Nat 2*), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 0*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 1*)), *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST client*))), *nil*, ()), (*META.SS-base* (*META.ST Client*)), ()), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 1*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST cl-res*))), *nil*, ()), (*META.SS-base* (*META.ST Reservation*)), ()), ()))), *nil*))), *uncurry cons* (*I* ((*META.SS-base* (*META.ST address*)), *OclTy-base-string*), *nil*)), *nil*), *nil*))), *uncurry cons* (*OclClass* ((*META.SS-base* (*META.ST Flight*)), *uncurry cons* (*I* ((*META.SS-base* (*META.ST passengers*)), *OclTy-object* (*OclTyObj* (*OclTyCore* (*Ocl-ty-class-ext* ((*META.SS-base* (*META.ST oid*)), (*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 2*), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 1*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST flights*))), *nil*, ()), (*META.SS-base* (*META.ST Flight*)), ()), *Ocl-ty-class-node-ext* ((*Code-Numeral.Nat 0*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST passengers*))), *nil*, ()), (*META.SS-base* (*META.ST Person*)), ()), ()))), *nil*))), *uncurry cons* (*I* ((*META.SS-base* (*META.ST seats*)), *OclTy-base-integer*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST from*)), *OclTy-base-string*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST to*)), *OclTy-base-string*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST fl-res*)), *OclTy-collection* (*Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST fl-res*))), *uncurry cons* (*Sequence*, *nil*), ()), *OclTy-object* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Reservation*))), *nil*))), *nil*))))), *nil*), *nil*))))), *uncurry cons* (*META-instance* (*OclInstance* (*uncurry cons* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST S1*))), *SOME* ((*META.SS-base* (*META.ST Staff*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST name*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Merlin*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flights*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*))))), *nil*))),

(()), *uncurry cons* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST C1*))), *SOME* ((*META.SS-base* (*META.ST Client*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST name*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Bertha*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST address*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Miami*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flights*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST cl-res*)), *ShallB-str* ((*META.SS-base* (*META.ST R11*))))), *nil*))))), ()), *uncurry cons* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST C2*))), *SOME* ((*META.SS-base* (*META.ST Client*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST name*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Arthur*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST address*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Valencia*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flights*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST cl-res*)), *ShallB-str* ((*META.SS-base* (*META.ST R21*))))), *nil*))))), ()), *uncurry cons* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST R11*))), *SOME* ((*META.SS-base* (*META.ST Reservation*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST id*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 12345*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flight*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST date*)), *ShallB-str* ((*META.SS-base* (*META.ST Mon*))))), *nil*))))), ()), *uncurry cons* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST R21*))), *SOME* ((*META.SS-base* (*META.ST Reservation*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST id*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 98765*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flight*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*))))), *nil*))), ()), *uncurry cons* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST F1*))), *SOME* ((*META.SS-base* (*META.ST Flight*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST seats*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 120*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST from*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Valencia*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST to*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Miami*)))))), *nil*))))), ()), *uncurry cons* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST F2*))), *SOME* ((*META.SS-base* (*META.ST Flight*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST seats*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 370*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST from*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Miami*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST to*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Ottawa*)))))), *nil*))))), ()), *nil*))))))))), *uncurry cons* (*META-enum* (*OclEnum* ((*META.SS-base* (*META.ST Week*)), *uncurry cons* ((*META.SS-base* (*META.ST Mon*)), *uncurry cons* ((*META.SS-base* (*META.ST Tue*)), *uncurry cons* ((*META.SS-base* (*META.ST Wed*)), *uncurry cons* ((*META.SS-base* (*META.ST Thu*)), *uncurry cons* ((*META.SS-base* (*META.ST Fri*)), *uncurry cons* ((*META.SS-base* (*META.ST Sat*)), *uncurry cons* ((*META.SS-base* (*META.ST Sun*)), *nil*))))))))), *uncurry cons* (*META-association* (*Ocl-association-ext* (*OclAssTy-association*, *OclAssRel* (*uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Reservation*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 0*)), *SOME* (*Mult-nat* ((*Code-Numeral.Nat 1*)))), *nil*), *SOME* ((*META.SS-base* (*META.ST next*))), *nil*, ())), *uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Reservation*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 0*)), *SOME* (*Mult-nat* ((*Code-Numeral.Nat 1*)))), *nil*), *SOME* ((*META.SS-base* (*META.ST prev*))), *nil*, ())), *nil*))), ())), *uncurry cons* (*META-association* (*Ocl-association-ext* (*OclAssTy-association*, *OclAssRel* (*uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Client*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 1*)), *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST client*))), *nil*, ())), *uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Reservation*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST cl-res*))), *nil*, ())), *nil*))), ())), *uncurry cons* (*META-association* (*Ocl-association-ext* (*OclAssTy-aggregation*, *OclAssRel* (*uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Flight*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-nat* ((*Code-Numeral.Nat 1*)), *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST flight*))), *nil*, ())), *uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Reservation*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST fl-res*))), *uncurry cons* (*Sequence*, *nil*), ())), *nil*))), ())), *uncurry cons* (*META-association* (*Ocl-association-ext* (*OclAssTy-association*, *OclAssRel* (*uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Person*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST passengers*))), *nil*, ())), *uncurry cons* (*I* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Flight*))), *nil*), *Ocl-multiplicity-ext* (*uncurry cons* (*I* (*Mult-star*, *NONE*), *nil*), *SOME* ((*META.SS-base* (*META.ST flights*))), *nil*, ())), *nil*))), ())), *uncurry cons* (*META-class-raw* (*Floor1*, *Ocl-class-raw-ext* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Staff*))), *uncurry cons* (*uncurry cons* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Person*))), *nil*), *nil*)), *nil*, *nil*, *false*, ())), *uncurry cons* (*META-class-raw* (*Floor1*, *Ocl-class-raw-ext* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Client*))), *uncurry cons* (*uncurry cons* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Person*))), *nil*), *nil*)), *uncurry cons* (*I* ((*META.SS-base* (*META.ST address*)), *OclTy-base-string*), *nil*), *nil*, *false*, ())), *uncurry cons* (*META-class-raw* (*Floor1*, *Ocl-class-raw-ext* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Person*))), *nil*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST name*)), *OclTy-base-string*), *nil*), *nil*, *false*, ())), *uncurry cons* (*META-class-raw* (*Floor1*, *Ocl-class-raw-ext* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Reservation*))), *nil*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST id*)), *OclTy-base-integer*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST date*)), *OclTy-object* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Week*))), *nil*))), *nil*)), *nil*, *false*, ())), *uncurry cons* (*META-class-raw* (*Floor1*, *Ocl-class-raw-ext* (*OclTyObj* (*OclTyCore-pre* ((*META.SS-base* (*META.ST Flight*))), *nil*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST seats*)), *OclTy-base-integer*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST from*)), *OclTy-base-string*), *uncurry cons* (*I* ((*META.SS-base* (*META.ST to*)), *OclTy-base-string*), *nil*))), *nil*, *false*, ())), *nil*))))))))))), *uncurry cons* (*I* ((*META.ST F2*), *I* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST F2*))), *SOME* ((*META.SS-base* (*META.ST Flight*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST seats*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 370*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST from*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Miami*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST to*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Ottawa*)))))), *nil*))))), ()), *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 9*)))), *uncurry cons* (*I* ((*META.ST F1*), *I* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST F1*))), *SOME* ((*META.SS-base* (*META.ST Flight*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST seats*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 120*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST from*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Valencia*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST to*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Miami*)))))), *nil*))))), ()), *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 8*)))), *uncurry cons* (*I* ((*META.ST*

R21), *I* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST R21*))), *SOME* ((*META.SS-base* (*META.ST Reservation*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST id*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 98765*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flight*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*)))))), *nil*))), ()), *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 7*)))), *uncurry cons* (*I* ((*META.ST R11*), *I* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST R11*))), *SOME* ((*META.SS-base* (*META.ST Reservation*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST id*)), *ShallB-term* (*OclDefInteger* ((*META.SS-base* (*META.ST 12345*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flight*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST date*)), *ShallB-str* ((*META.SS-base* (*META.ST Mon*)))))), *nil*)))), ()), *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 6*)))), *uncurry cons* (*I* ((*META.ST C2*), *I* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST C2*))), *SOME* ((*META.SS-base* (*META.ST Client*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST name*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Arthur*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST address*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Valencia*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flights*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST cl-res*)), *ShallB-str* ((*META.SS-base* (*META.ST R21*)))))), *nil*))))), ()), *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 5*)))), *uncurry cons* (*I* ((*META.ST C1*), *I* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST C1*))), *SOME* ((*META.SS-base* (*META.ST Client*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST name*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Bertha*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST address*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Miami*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flights*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST cl-res*)), *ShallB-str* ((*META.SS-base* (*META.ST R11*)))))), *nil*))))), ()), *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 4*)))), *uncurry cons* (*I* ((*META.ST S1*), *I* (*Ocl-instance-single-ext* (*SOME* ((*META.SS-base* (*META.ST S1*))), *SOME* ((*META.SS-base* (*META.ST Staff*))), *NONE*, *OclAttrNoCast* (*uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST name*)), *ShallB-term* (*OclDefString* ((*META.SS-base* (*META.ST Merlin*)))))), *uncurry cons* (*I* (*NONE*, *I* ((*META.SS-base* (*META.ST flights*)), *ShallB-str* ((*META.SS-base* (*META.ST F1*)))))), *nil*))), ()), *Oids* ((*Code-Numeral.Nat 0*), (*Code-Numeral.Nat 3*), (*Code-Numeral.Nat 3*)))), *nil*)))))), *nil*, *true*, *false*, *I* (*uncurry cons* ((*META.ST dot--flightat-pre*), *uncurry cons* ((*META.ST dot--dateat-pre*), *uncurry cons* ((*META.ST dot--idat-pre*), *uncurry cons* ((*META.ST dot-1---clientat-pre*), *uncurry cons* ((*META.ST dot-1---nextat-pre*), *uncurry cons* ((*META.ST dot-0---prevat-pre*), *uncurry cons* ((*META.ST dot--nameat-pre*), *uncurry cons* ((*META.ST dot-0---flightsat-pre*), *uncurry cons* ((*META.ST dot--addressat-pre*), *uncurry cons* ((*META.ST dot-0---cl-resat-pre*), *uncurry cons* ((*META.ST dot--fl-resat-pre*), *uncurry cons* ((*META.ST dot--toat-pre*), *uncurry cons* ((*META.ST dot--fromat-pre*), *uncurry cons* ((*META.ST dot--seatsat-pre*), *uncurry cons* ((*META.ST dot-1---passengersat-pre*), *nil*))))))))))))), *uncurry cons* ((*META.ST dot--flight*), *uncurry cons* ((*META.ST dot--date*), *uncurry cons* ((*META.ST dot--id*), *uncurry cons* ((*META.ST dot-1---client*), *uncurry cons* ((*META.ST dot-1---next*), *uncurry cons* ((*META.ST dot-0---prev*), *uncurry cons* ((*META.ST dot--name*), *uncurry cons* ((*META.ST dot-0---flights*), *uncurry cons* ((*META.ST dot--address*), *uncurry cons* ((*META.ST dot-0---cl-res*), *uncurry cons* ((*META.ST dot--fl-res*), *uncurry cons* ((*META.ST dot--to*), *uncurry cons* ((*META.ST dot--from*), *uncurry cons* ((*META.ST dot--seats*), *uncurry cons* ((*META.ST dot-1---passengers*), *nil*))))))))))))))), *uncurry cons* ((*META.ST Sequence-Person*), *uncurry cons* ((*META.ST Set-Person*), *uncurry cons* ((*META.ST Sequence-Flight*), *uncurry cons* ((*META.ST Set-Flight*), *uncurry cons* ((*META.ST Sequence-Client*), *uncurry cons* ((*META.ST Set-Client*), *uncurry cons* ((*META.ST Sequence-Reservation*), *uncurry cons* ((*META.ST Set-Reservation*), *nil*))))))))), *I* (*NONE*, *false*), ()) *end*))))⟩
State[shallow] σ_1 = [*S1*, *C1*, *C2*, *R11*, *R21*, *F1*, *F2*]

B.12 State (Floor 1)

definition (*typecheck-state-bad-head-on-lhs-σ_2* (σ_2)) = ()
definition *typecheck-state-extra-variables-on-rhs-σ_2* = (*F2* , *F2* , *F1* , *R21* , *F1* , *R11* , *C2* , *F2* , *F1* , *C1* , *R11* , *F1* , *S1*)

Instance σ_2-*object1* :: *Client* = [*C1* with-only *name* = *Bertha*, *address* = *Saint−Malo*, *flights* = *F1*, *cl-res* = *R11*]
and σ_2-*object2* :: *Client* = [*C2* with-only *name* = *Arthur*, *address* = *Valencia*, *flights* = [*F1*, *F2*], *cl-res* = [self *2*, self *3*]]
and σ_2-*object4* :: *Reservation* = [*R21* with-only *id* = *98765*, *flight* = *F1*, *next* = self *3*]
and σ_2-*object7* :: *Reservation* = [*id* = *19283*, *flight* = *F2*]
State[shallow] σ_2 = [*S1*, σ_2-*object1*, σ_2-*object2*, *R11*, σ_2-*object4*, *F1*, *F2*, σ_2-*object7*]

B.13 Transition (Floor 1)

Transition[shallow] σ_1 σ_2

B.14 Context (Floor 1)

Context[shallow] *f* : *Flight* Inv *A* : (λ *self f*. (**0** $<_{int}$ (*f .seats*)))
Inv *B* : (λ *self f*. (*f .fl-res* $->size_{Seq}()$ $\leq_{int}$ (*f .seats*)))
Inv *C* : (λ *self f*. (*f .passengers* $->select_{Set}(p \mid p\ .oclIsTypeOf(Client))$
$\doteq$ ((*f .fl-res*)$->collect_{Seq}(c \mid c\ .client\ .oclAsType(Person))->asSet_{Seq}()$)))

B.15 Context (Floor 1)

Context[shallow] *r* : *Reservation* Inv *A* : (λ *self r*. (**0** $<_{int}$ (*r .id*)))

Inv B : $(\lambda\ self\ r.\ (r\ .next <> null\ implies\ (r\ .flight\ .to \doteq r\ .next\ .flight\ .from)))$
Inv C : $(\lambda\ self\ r.\ (r\ .next <> null\ implies\ (r\ .client \doteq r\ .next\ .client)))$

B.16 Context (Floor 1)

consts $dot\text{--}book :: (\mathfrak{A}, 'o)\ val \Rightarrow (\cdot Flight) \Rightarrow (Void)\ ((\text{-})\ .book'((\text{-})'))$
consts $dot\text{--}bookat\text{-}pre :: (\mathfrak{A}, 'o)\ val \Rightarrow (\cdot Flight) \Rightarrow (Void)\ ((\text{-})\ .book@pre'((\text{-})'))$
Context[shallow] $Client :: book\ (f : Flight)$
Pre : $(\lambda\ f\ self.\ (f\ .passengers\ ->excludes_{Set}(self\ .oclAsType(Person))$
$and\ (f\ .fl\text{-}res\ ->size_{Seq}()\ <_{int}\ (f\ .seats))))$
Post : $(\lambda\ result\ f\ self.\ (f\ .passengers \doteq (f\ .passengers@pre\ ->including_{Set}(self\ .oclAsType(Person)))$
$and\ (let\ r = self\ .cl\text{-}res\ ->select_{Set}(r \mid r\ .flight \doteq f)->any_{Set}()\ in$
$(r\ .oclIsNew())$
$and\ (r\ .prev \doteq null)$
$and\ (r\ .next \doteq null))))$

B.17 Context (Floor 1)

consts $dot\text{--}booknext :: (\mathfrak{A}, 'o)\ val \Rightarrow (\cdot Flight) \Rightarrow (\cdot Reservation) \Rightarrow (Void)\ ((\text{-})\ .booknext'((\text{-}),(\text{-})'))$
consts $dot\text{--}booknextat\text{-}pre :: (\mathfrak{A}, 'o)\ val \Rightarrow (\cdot Flight) \Rightarrow (\cdot Reservation) \Rightarrow (Void)\ ((\text{-})\ .booknext@pre'((\text{-}),(\text{-})'))$
Context[shallow] $Client :: booknext\ (f : Flight,\ r : Reservation)$
Pre : $(\lambda\ r\ f\ self.\ (f\ .passengers\ ->excludes_{Set}(self\ .oclAsType(Person))$
$and\ (f\ .fl\text{-}res\ ->size_{Seq}()\ <_{int}\ (f\ .seats))$
$and\ (r\ .client \doteq self)$
$and\ (f\ .from \doteq (r\ .flight\ .to))))$
Post : $(\lambda\ result\ r\ f\ self.\ (f\ .passengers \doteq (f\ .passengers@pre\ ->including_{Set}(self\ .oclAsType(Person)))$
$and\ (let\ r = self\ .cl\text{-}res\ ->select_{Set}(r \mid r\ .flight \doteq f)->any_{Set}()\ in$
$(r\ .oclIsNew())$
$and\ (r\ .prev \doteq r)$
$and\ (r\ .next \doteq null))))$

B.18 Context (Floor 1)

consts $dot\text{--}cancel :: (\mathfrak{A}, 'o)\ val \Rightarrow (\cdot Reservation) \Rightarrow (Void)\ ((\text{-})\ .cancel'((\text{-})'))$
consts $dot\text{--}cancelat\text{-}pre :: (\mathfrak{A}, 'o)\ val \Rightarrow (\cdot Reservation) \Rightarrow (Void)\ ((\text{-})\ .cancel@pre'((\text{-})'))$
Context[shallow] $Client :: cancel\ (r : Reservation)$
Pre : $(\lambda\ r\ self.\ (r\ .client \doteq self))$
Post : $(\lambda\ result\ r\ self.\ (self\ .cl\text{-}res\ ->select_{Set}(res \mid res\ .flight \doteq r\ .flight@pre)$
$->isEmpty_{Set}()))$

B.19 Context (Floor 1)

type-synonym $Set\text{-}Integer = (\mathfrak{A}, Integer_{base}\ Set_{base})\ val$
consts $dot\text{--}connections :: (\mathfrak{A}, 'o)\ val \Rightarrow (Set\text{-}Integer)\ ((\text{-})\ .connections'('))$
consts $dot\text{--}connectionsat\text{-}pre :: (\mathfrak{A}, 'o)\ val \Rightarrow (Set\text{-}Integer)\ ((\text{-})\ .connections@pre'('))$
Context[shallow] $Reservation :: connections\ () : Set(Integer)$
Post : $(\lambda\ result\ self.\ (result \triangleq if\ (self\ .next \doteq null)$
$then\ (Set\{\}->including_{Set}(self\ .id))$
$else\ (self\ .next\ .connections()->including_{Set}(self\ .id))$
$endif))$
Pre : $(\lambda\ self.\ (true))$

end

APPENDIX C

The Flight Model (Generated Theory, Floor 2)

This chapter has been generated from Appendix B. For space reasons, all the code occurring at the beginning similar as Appendix B has implicitly been skipped, i. e., we have explicitly removed by hand the piece of code which is propagated across floors in Figure 6.6. However this code actually existed and was correctly evaluated for the Isabelle system being able to generate this PDF document without errors. In addition, we also do not display the generated code associated to each command Context situated at the end of Appendix B, because the end of Appendix B is mixing Isar_HOL commands with meta-commands (we would otherwise obtain a not well-typed file as explained in Figure 6.6).

theory *Flight-Model-generated-generated* imports *../src/UML-Main* *../src/compiler/Static* *../src/compiler/Generator-dynamic*
begin

C.1 State (Floor 2)

locale *state-σ_1* =
fixes *oid3* :: *nat*
fixes *oid4* :: *nat*
fixes *oid5* :: *nat*
fixes *oid6* :: *nat*
fixes *oid7* :: *nat*
fixes *oid8* :: *nat*
fixes *oid9* :: *nat*
assumes *distinct-oid*: (*distinct* ([*oid3* , *oid4* , *oid5* , *oid6* , *oid7* , *oid8* , *oid9*]))
fixes $S1_{Staff}$ :: ty_{Staff}
fixes *S1* :: *·Staff*
assumes *S1-def*: *S1* = (λ-. ⌊⌊$S1_{Staff}$⌋⌋)
fixes $C1_{Client}$ :: ty_{Client}
fixes *C1* :: *·Client*
assumes *C1-def*: *C1* = (λ-. ⌊⌊$C1_{Client}$⌋⌋)
fixes $C2_{Client}$ :: ty_{Client}
fixes *C2* :: *·Client*
assumes *C2-def*: *C2* = (λ-. ⌊⌊$C2_{Client}$⌋⌋)
fixes $R11_{Reservation}$ :: $ty_{Reservation}$
fixes *R11* :: *·Reservation*
assumes *R11-def*: *R11* = (λ-. ⌊⌊$R11_{Reservation}$⌋⌋)
fixes $R21_{Reservation}$ :: $ty_{Reservation}$
fixes *R21* :: *·Reservation*
assumes *R21-def*: *R21* = (λ-. ⌊⌊$R21_{Reservation}$⌋⌋)
fixes $F1_{Flight}$ :: ty_{Flight}
fixes *F1* :: *·Flight*
assumes *F1-def*: *F1* = (λ-. ⌊⌊$F1_{Flight}$⌋⌋)
fixes $F2_{Flight}$ :: ty_{Flight}
fixes *F2* :: *·Flight*
assumes *F2-def*: *F2* = (λ-. ⌊⌊$F2_{Flight}$⌋⌋)
begin
definition σ_1 = (*state.make* ((*Map.empty* (*oid3* ↦ (in_{Staff} ($S1_{Staff}$))) (*oid4* ↦ (in_{Client} ($C1_{Client}$))) (*oid5* ↦ (in_{Client} ($C2_{Client}$))) (*oid6* ↦ ($in_{Reservation}$ ($R11_{Reservation}$))) (*oid7* ↦ ($in_{Reservation}$ ($R21_{Reservation}$))) (*oid8* ↦ (in_{Flight} ($F1_{Flight}$))) (*oid9* ↦ (in_{Flight} ($F2_{Flight}$))))) ((*map-of-list* [(oid_{Staff}*-0---flights* , (*List.map* ((λ(*x* , *y*). [*x* , *y*]) *o* $switch_2$*-01*) ([[[*oid3*] , [*oid8*]]]))) , (oid_{Client}*-0---flights* , (*List.map* ((λ(*x* , *y*). [*x* , *y*]) *o* $switch_2$*-01*) ([[[*oid4*] , [*oid8*]] , [[*oid5*] , [*oid8*]]]))) , (oid_{Client}*-0---cl-res* , (*List.map* ((λ(*x* , *y*). [*x* , *y*]) *o* $switch_2$*-01*) ([[[*oid4*] , [*oid6*]] , [[*oid5*] , [*oid7*]]]))))]))))

lemma *dom-σ_1* : (*dom* ((*heap* (σ_1)))) = {*oid3* , *oid4* , *oid5* , *oid6* , *oid7* , *oid8* , *oid9*}
by(*auto simp*: σ_1*-def*)

lemmas[*simp*,*code-unfold*] = *dom*-σ_1

lemma *perm*-σ_1 : σ_1 = (*state.make* ((*Map.empty* (*oid9* ↦ (in_{Flight} ($F2_{Flight}$))) (*oid8* ↦ (in_{Flight} ($F1_{Flight}$))) (*oid7* ↦ ($in_{Reservation}$ ($R21_{Reservation}$))) (*oid6* ↦ ($in_{Reservation}$ ($R11_{Reservation}$))) (*oid5* ↦ (in_{Client} ($C2_{Client}$))) (*oid4* ↦ (in_{Client} ($C1_{Client}$))) (*oid3* ↦ (in_{Staff} ($S1_{Staff}$))))) ((*assocs* (σ_1))))
apply(*simp add*: σ_1-*def*)
apply(*subst* (*1*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*2*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*1*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*3*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*2*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*1*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*4*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*3*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*2*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*1*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*5*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*4*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*3*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*2*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*1*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*6*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*5*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*4*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*3*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*2*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
apply(*subst* (*1*) *fun-upd-twist*, *metis distinct-oid distinct-length-2-or-more*)
by(*simp*)

lemma σ_1-*OclAllInstances-generic-exec-Flight* :
assumes [*simp*]: (*Flight* ((in_{Staff} ($S1_{Staff}$)))) = *None*
assumes [*simp*]: (*Flight* ((in_{Client} ($C1_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* ((in_{Client} ($C2_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* (($in_{Reservation}$ ($R11_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* (($in_{Reservation}$ ($R21_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* ((in_{Flight} ($F1_{Flight}$)))) ≠ *None*
assumes [*simp*]: (*Flight* ((in_{Flight} ($F2_{Flight}$)))) ≠ *None*
assumes [*simp*]: ($\bigwedge$*a*. (*pre-post* ((*mk* (*a*)))) = *a*)
shows (*mk* (σ_1)) ⊨ (*OclAllInstances-generic* (*pre-post*) (*Flight*)) $\doteq$ *Set*{*F1* , *F2*}
apply(*subst perm*-σ_1)
apply(*simp only*: *state.make-def F1-def F2-def*)
apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp*, *simp*, *simp*)
apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp*, *simp*, *simp*)
apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp*, *simp*, *simp*)
apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp*, *simp*, *simp*)
apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp*, *simp*, *simp*)
apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp*, *simp*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)
apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp*, *simp*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)
apply(*rule state-update-vs-allInstances-generic-empty*)
by(*simp-all only*: *assms*, (*simp-all add*: *OclAsType*$_{Flight}$-$\mathfrak{A}$*-def*)?)

lemma σ_1-*OclAllInstances-at-post-exec-Flight* :
assumes [*simp*]: (*Flight* ((in_{Staff} ($S1_{Staff}$)))) = *None*
assumes [*simp*]: (*Flight* ((in_{Client} ($C1_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* ((in_{Client} ($C2_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* (($in_{Reservation}$ ($R11_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* (($in_{Reservation}$ ($R21_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* ((in_{Flight} ($F1_{Flight}$)))) ≠ *None*
assumes [*simp*]: (*Flight* ((in_{Flight} ($F2_{Flight}$)))) ≠ *None*
shows (*st* , σ_1) ⊨ (*OclAllInstances-at-post* (*Flight*)) $\doteq$ *Set*{*F1* , *F2*}
unfolding *OclAllInstances-at-post-def*

by(rule σ_1-OclAllInstances-generic-exec-Flight, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-at-pre-exec-Flight :
assumes [simp]: (Flight ((in_{Staff} ($S1_{Staff}$)))) = None
assumes [simp]: (Flight ((in_{Client} ($C1_{Client}$)))) = None
assumes [simp]: (Flight ((in_{Client} ($C2_{Client}$)))) = None
assumes [simp]: (Flight (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Flight (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Flight ((in_{Flight} ($F1_{Flight}$)))) $\neq$ None
assumes [simp]: (Flight ((in_{Flight} ($F2_{Flight}$)))) $\neq$ None
shows (σ_1 , st) $\models$ (OclAllInstances-at-pre (Flight)) $\doteq$ Set{F1 , F2}
unfolding OclAllInstances-at-pre-def
by(rule σ_1-OclAllInstances-generic-exec-Flight, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-generic-exec-Client :
assumes [simp]: (Client ((in_{Staff} ($S1_{Staff}$)))) = None
assumes [simp]: (Client ((in_{Client} ($C1_{Client}$)))) $\neq$ None
assumes [simp]: (Client ((in_{Client} ($C2_{Client}$)))) $\neq$ None
assumes [simp]: (Client (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Client (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Client ((in_{Flight} ($F1_{Flight}$)))) = None
assumes [simp]: (Client ((in_{Flight} ($F2_{Flight}$)))) = None
assumes [simp]: ($\bigwedge$a. (pre-post ((mk (a)))) = a)
shows (mk (σ_1)) $\models$ (OclAllInstances-generic (pre-post) (Client)) $\doteq$ Set{C1 , C2}
apply(subst perm-σ_1)
apply(simp only: state.make-def C1-def C2-def)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(rule state-update-vs-allInstances-generic-empty)
by(simp-all only: assms, (simp-all add: OclAsType$_{Client}$-$\mathfrak{A}$-def)?)

lemma σ_1-OclAllInstances-at-post-exec-Client :
assumes [simp]: (Client ((in_{Staff} ($S1_{Staff}$)))) = None
assumes [simp]: (Client ((in_{Client} ($C1_{Client}$)))) $\neq$ None
assumes [simp]: (Client ((in_{Client} ($C2_{Client}$)))) $\neq$ None
assumes [simp]: (Client (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Client (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Client ((in_{Flight} ($F1_{Flight}$)))) = None
assumes [simp]: (Client ((in_{Flight} ($F2_{Flight}$)))) = None
shows (st , σ_1) $\models$ (OclAllInstances-at-post (Client)) $\doteq$ Set{C1 , C2}
unfolding OclAllInstances-at-post-def
by(rule σ_1-OclAllInstances-generic-exec-Client, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-at-pre-exec-Client :
assumes [simp]: (Client ((in_{Staff} ($S1_{Staff}$)))) = None
assumes [simp]: (Client ((in_{Client} ($C1_{Client}$)))) $\neq$ None
assumes [simp]: (Client ((in_{Client} ($C2_{Client}$)))) $\neq$ None
assumes [simp]: (Client (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Client (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Client ((in_{Flight} ($F1_{Flight}$)))) = None
assumes [simp]: (Client ((in_{Flight} ($F2_{Flight}$)))) = None
shows (σ_1 , st) $\models$ (OclAllInstances-at-pre (Client)) $\doteq$ Set{C1 , C2}
unfolding OclAllInstances-at-pre-def
by(rule σ_1-OclAllInstances-generic-exec-Client, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-generic-exec-Staff :
assumes [simp]: (Staff ((in_{Staff} ($S1_{Staff}$)))) $\neq$ None

assumes [simp]: (Staff ((in_{Client} ($C1_{Client}$)))) = None
assumes [simp]: (Staff ((in_{Client} ($C2_{Client}$)))) = None
assumes [simp]: (Staff (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Staff (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Staff ((in_{Flight} ($F1_{Flight}$)))) = None
assumes [simp]: (Staff ((in_{Flight} ($F2_{Flight}$)))) = None
assumes [simp]: ($\bigwedge$a. (pre-post ((mk (a)))) = a)
shows (mk (σ_1)) $\models$ (OclAllInstances-generic (pre-post) (Staff)) $\doteq$ Set{S1}
apply(subst perm-σ_1)
apply(simp only: state.make-def S1-def)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(rule state-update-vs-allInstances-generic-empty)
by(simp-all only: assms, (simp-all add: OclAsType$_{Staff}$-$\mathfrak{A}$-def)?)

lemma σ_1-OclAllInstances-at-post-exec-Staff :
assumes [simp]: (Staff ((in_{Staff} ($S1_{Staff}$)))) $\neq$ None
assumes [simp]: (Staff ((in_{Client} ($C1_{Client}$)))) = None
assumes [simp]: (Staff ((in_{Client} ($C2_{Client}$)))) = None
assumes [simp]: (Staff (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Staff (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Staff ((in_{Flight} ($F1_{Flight}$)))) = None
assumes [simp]: (Staff ((in_{Flight} ($F2_{Flight}$)))) = None
shows (st , σ_1) $\models$ (OclAllInstances-at-post (Staff)) $\doteq$ Set{S1}
unfolding OclAllInstances-at-post-def
by(rule σ_1-OclAllInstances-generic-exec-Staff, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-at-pre-exec-Staff :
assumes [simp]: (Staff ((in_{Staff} ($S1_{Staff}$)))) $\neq$ None
assumes [simp]: (Staff ((in_{Client} ($C1_{Client}$)))) = None
assumes [simp]: (Staff ((in_{Client} ($C2_{Client}$)))) = None
assumes [simp]: (Staff (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Staff (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Staff ((in_{Flight} ($F1_{Flight}$)))) = None
assumes [simp]: (Staff ((in_{Flight} ($F2_{Flight}$)))) = None
shows (σ_1 , st) $\models$ (OclAllInstances-at-pre (Staff)) $\doteq$ Set{S1}
unfolding OclAllInstances-at-pre-def
by(rule σ_1-OclAllInstances-generic-exec-Staff, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-generic-exec-Person :
assumes [simp]: (Person ((in_{Staff} ($S1_{Staff}$)))) $\neq$ None
assumes [simp]: (Person ((in_{Client} ($C1_{Client}$)))) $\neq$ None
assumes [simp]: (Person ((in_{Client} ($C2_{Client}$)))) $\neq$ None
assumes [simp]: (Person (($in_{Reservation}$ ($R11_{Reservation}$)))) = None
assumes [simp]: (Person (($in_{Reservation}$ ($R21_{Reservation}$)))) = None
assumes [simp]: (Person ((in_{Flight} ($F1_{Flight}$)))) = None
assumes [simp]: (Person ((in_{Flight} ($F2_{Flight}$)))) = None
assumes [simp]: (λ-. $\lfloor$(Person ((in_{Staff} ($S1_{Staff}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor S1_{Staff}\rfloor\rfloor$)::$\cdot$Staff)) .oclAsType(Person))
assumes [simp]: (λ-. $\lfloor$(Person ((in_{Client} ($C1_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor C1_{Client}\rfloor\rfloor$)::$\cdot$Client)) .oclAsType(Person))
assumes [simp]: (λ-. $\lfloor$(Person ((in_{Client} ($C2_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor C2_{Client}\rfloor\rfloor$)::$\cdot$Client)) .oclAsType(Person))
assumes [simp]: ($\bigwedge$a. (pre-post ((mk (a)))) = a)
shows (mk (σ_1)) $\models$ (OclAllInstances-generic (pre-post) (Person)) $\doteq$ Set{S1 .oclAsType(Person) , C1 .oclAsType(Person) , C2 .oclAsType(Person)}
apply(subst perm-σ_1)
apply(simp only: state.make-def S1-def C1-def C2-def)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{Person}$-Staff OclAsType$_{Person}$-Client OclAsType$_{Person}$-Client, simp del: OclAsType$_{Person}$-Staff OclAsType$_{Person}$-Client OclAsType$_{Person}$-Client, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only:

assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{Person}$-Staff OclAsType$_{Person}$-Client OclAsType$_{Person}$-Client, simp del: OclAsType$_{Person}$-Staff OclAsType$_{Person}$-Client OclAsType$_{Person}$-Client, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{Person}$-Staff OclAsType$_{Person}$-Client OclAsType$_{Person}$-Client, simp del: OclAsType$_{Person}$-Staff OclAsType$_{Person}$-Client OclAsType$_{Person}$-Client, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)

apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)

apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)

apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)

apply(rule state-update-vs-allInstances-generic-empty)

by(simp-all only: assms, (simp-all add: OclAsType$_{Person}$-$\mathfrak{A}$-def)?)

lemma σ_1-OclAllInstances-at-post-exec-Person :
assumes [simp]: (Person ((in$_{Staff}$ (S1$_{Staff}$)))) $\neq$ None
assumes [simp]: (Person ((in$_{Client}$ (C1$_{Client}$)))) $\neq$ None
assumes [simp]: (Person ((in$_{Client}$ (C2$_{Client}$)))) $\neq$ None
assumes [simp]: (Person ((in$_{Reservation}$ (R11$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Reservation}$ (R21$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (λ-. $\lfloor$(Person ((in$_{Staff}$ (S1$_{Staff}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor$S1$_{Staff}\rfloor\rfloor$)::·Staff)) .oclAsType(Person))
assumes [simp]: (λ-. $\lfloor$(Person ((in$_{Client}$ (C1$_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor$C1$_{Client}\rfloor\rfloor$)::·Client)) .oclAsType(Person))
assumes [simp]: (λ-. $\lfloor$(Person ((in$_{Client}$ (C2$_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor$C2$_{Client}\rfloor\rfloor$)::·Client)) .oclAsType(Person))
shows (st , σ_1) $\models$ (OclAllInstances-at-post (Person)) $\doteq$ Set{S1 .oclAsType(Person) , C1 .oclAsType(Person) , C2 .oclAsType(Person)}
unfolding OclAllInstances-at-post-def
by(rule σ_1-OclAllInstances-generic-exec-Person, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-at-pre-exec-Person :
assumes [simp]: (Person ((in$_{Staff}$ (S1$_{Staff}$)))) $\neq$ None
assumes [simp]: (Person ((in$_{Client}$ (C1$_{Client}$)))) $\neq$ None
assumes [simp]: (Person ((in$_{Client}$ (C2$_{Client}$)))) $\neq$ None
assumes [simp]: (Person ((in$_{Reservation}$ (R11$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Reservation}$ (R21$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (λ-. $\lfloor$(Person ((in$_{Staff}$ (S1$_{Staff}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor$S1$_{Staff}\rfloor\rfloor$)::·Staff)) .oclAsType(Person))
assumes [simp]: (λ-. $\lfloor$(Person ((in$_{Client}$ (C1$_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor$C1$_{Client}\rfloor\rfloor$)::·Client)) .oclAsType(Person))
assumes [simp]: (λ-. $\lfloor$(Person ((in$_{Client}$ (C2$_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor$C2$_{Client}\rfloor\rfloor$)::·Client)) .oclAsType(Person))
shows (σ_1 , st) $\models$ (OclAllInstances-at-pre (Person)) $\doteq$ Set{S1 .oclAsType(Person) , C1 .oclAsType(Person) , C2 .oclAsType(Person)}
unfolding OclAllInstances-at-pre-def
by(rule σ_1-OclAllInstances-generic-exec-Person, simp-all only: assms, simp-all)

lemma σ_1-OclAllInstances-generic-exec-Reservation :
assumes [simp]: (Reservation ((in$_{Staff}$ (S1$_{Staff}$)))) = None
assumes [simp]: (Reservation ((in$_{Client}$ (C1$_{Client}$)))) = None
assumes [simp]: (Reservation ((in$_{Client}$ (C2$_{Client}$)))) = None
assumes [simp]: (Reservation ((in$_{Reservation}$ (R11$_{Reservation}$)))) $\neq$ None
assumes [simp]: (Reservation ((in$_{Reservation}$ (R21$_{Reservation}$)))) $\neq$ None
assumes [simp]: (Reservation ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Reservation ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: ($\bigwedge$a. (pre-post ((mk (a)))) = a)
shows (mk (σ_1)) $\models$ (OclAllInstances-generic (pre-post) (Reservation)) $\doteq$ Set{R11 , R21}
apply(subst perm-σ_1)
apply(simp only: state.make-def R11-def R21-def)

apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)

apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)

apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)

apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms,

blast, *simp*, *rule const-StrictRefEq$_{Set}$-including*, *simp*, *simp*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])*?*, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])*?*, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)
apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)*?*, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq$_{Set}$-including*, *simp*, *simp*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])*?*, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])*?*, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)
apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)*?*, *simp only*: *assms*, *simp*, *rule const-StrictRefEq$_{Set}$-empty*, *simp*)
apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)*?*, *simp only*: *assms*, *simp*, *rule const-StrictRefEq$_{Set}$-empty*, *simp*)
apply(*rule state-update-vs-allInstances-generic-empty*)
by(*simp-all only*: *assms*, (*simp-all add*: *OclAsType$_{Reservation}$-𝔄-def*)*?*)

lemma *σ$_1$-OclAllInstances-at-post-exec-Reservation* :
assumes [*simp*]: (*Reservation* ((*in$_{Staff}$* (*S1$_{Staff}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Client}$* (*C1$_{Client}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Client}$* (*C2$_{Client}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Reservation}$* (*R11$_{Reservation}$*)))) ≠ *None*
assumes [*simp*]: (*Reservation* ((*in$_{Reservation}$* (*R21$_{Reservation}$*)))) ≠ *None*
assumes [*simp*]: (*Reservation* ((*in$_{Flight}$* (*F1$_{Flight}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Flight}$* (*F2$_{Flight}$*)))) = *None*
shows (*st* , *σ$_1$*) ⊨ (*OclAllInstances-at-post* (*Reservation*)) ≐ *Set*{*R11* , *R21*}
unfolding *OclAllInstances-at-post-def*
by(*rule σ$_1$-OclAllInstances-generic-exec-Reservation*, *simp-all only*: *assms*, *simp-all*)

lemma *σ$_1$-OclAllInstances-at-pre-exec-Reservation* :
assumes [*simp*]: (*Reservation* ((*in$_{Staff}$* (*S1$_{Staff}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Client}$* (*C1$_{Client}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Client}$* (*C2$_{Client}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Reservation}$* (*R11$_{Reservation}$*)))) ≠ *None*
assumes [*simp*]: (*Reservation* ((*in$_{Reservation}$* (*R21$_{Reservation}$*)))) ≠ *None*
assumes [*simp*]: (*Reservation* ((*in$_{Flight}$* (*F1$_{Flight}$*)))) = *None*
assumes [*simp*]: (*Reservation* ((*in$_{Flight}$* (*F2$_{Flight}$*)))) = *None*
shows (*σ$_1$* , *st*) ⊨ (*OclAllInstances-at-pre* (*Reservation*)) ≐ *Set*{*R11* , *R21*}
unfolding *OclAllInstances-at-pre-def*
by(*rule σ$_1$-OclAllInstances-generic-exec-Reservation*, *simp-all only*: *assms*, *simp-all*)

lemma *σ$_1$-OclAllInstances-generic-exec-OclAny* :
assumes [*simp*]: (*OclAny* ((*in$_{Staff}$* (*S1$_{Staff}$*)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in$_{Client}$* (*C1$_{Client}$*)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in$_{Client}$* (*C2$_{Client}$*)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in$_{Reservation}$* (*R11$_{Reservation}$*)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in$_{Reservation}$* (*R21$_{Reservation}$*)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in$_{Flight}$* (*F1$_{Flight}$*)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in$_{Flight}$* (*F2$_{Flight}$*)))) ≠ *None*
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in$_{Staff}$* (*S1$_{Staff}$*))))⌋) = ((((λ-. ⌊⌊*S1$_{Staff}$*⌋⌋)::·*Staff*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in$_{Client}$* (*C1$_{Client}$*))))⌋) = ((((λ-. ⌊⌊*C1$_{Client}$*⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in$_{Client}$* (*C2$_{Client}$*))))⌋) = ((((λ-. ⌊⌊*C2$_{Client}$*⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in$_{Reservation}$* (*R11$_{Reservation}$*))))⌋) = ((((λ-. ⌊⌊*R11$_{Reservation}$*⌋⌋)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in$_{Reservation}$* (*R21$_{Reservation}$*))))⌋) = ((((λ-. ⌊⌊*R21$_{Reservation}$*⌋⌋)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in$_{Flight}$* (*F1$_{Flight}$*))))⌋) = ((((λ-. ⌊⌊*F1$_{Flight}$*⌋⌋)::·*Flight*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in$_{Flight}$* (*F2$_{Flight}$*))))⌋) = ((((λ-. ⌊⌊*F2$_{Flight}$*⌋⌋)::·*Flight*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (⋀*a*. (*pre-post* ((*mk* (*a*)))) = *a*)
shows (*mk* (*σ$_1$*)) ⊨ (*OclAllInstances-generic* (*pre-post*) (*OclAny*)) ≐ *Set*{*S1* .*oclAsType*(*OclAny*) , *C1* .*oclAsType*(*OclAny*) , *C2* .*oclAsType*(*OclAny*) , *R11* .*oclAsType*(*OclAny*) , *R21* .*oclAsType*(*OclAny*) , *F1* .*oclAsType*(*OclAny*) , *F2* .*oclAsType*(*OclAny*)}
apply(*subst perm-σ$_1$*)
apply(*simp only*: *state.make-def S1-def C1-def C2-def R11-def R21-def F1-def F2-def*)
apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)*?*, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq$_{Set}$-including*, *simp del*: *OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight*, *simp del*: *OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])*?*, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])*?*, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)
apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)*?*, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq$_{Set}$-including*, *simp del*: *OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight*, *simp del*: *OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])*?*, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*:

assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp del: OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*rule state-update-vs-allInstances-generic-empty*)
by(*simp-all only: assms, (simp-all add: OclAsType*$_{OclAny}$*-*$\mathfrak{A}$*-def)?*)

lemma σ_1*-OclAllInstances-at-post-exec-OclAny* :
assumes [*simp*]: (*OclAny* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Client}$ (*C1*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Client}$ (*C2*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Reservation}$ (*R21*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Staff}$ (*S1*$_{Staff}$))))⌋) = ((((λ-. ⌊⌊*S1*$_{Staff}$⌋⌋)::·*Staff*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Client}$ (*C1*$_{Client}$))))⌋) = ((((λ-. ⌊⌊*C1*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Client}$ (*C2*$_{Client}$))))⌋) = ((((λ-. ⌊⌊*C2*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$))))⌋) = ((((λ-. ⌊⌊*R11*$_{Reservation}$⌋⌋)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Reservation}$ (*R21*$_{Reservation}$))))⌋) = ((((λ-. ⌊⌊*R21*$_{Reservation}$⌋⌋)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Flight}$ (*F1*$_{Flight}$))))⌋) = ((((λ-. ⌊⌊*F1*$_{Flight}$⌋⌋)::·*Flight*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Flight}$ (*F2*$_{Flight}$))))⌋) = ((((λ-. ⌊⌊*F2*$_{Flight}$⌋⌋)::·*Flight*)) .*oclAsType*(*OclAny*))
shows (*st* , σ_1) ⊨ (*OclAllInstances-at-post* (*OclAny*)) $\doteq$ *Set*{*S1* .*oclAsType*(*OclAny*) , *C1* .*oclAsType*(*OclAny*) , *C2* .*oclAsType*(*OclAny*) , *R11* .*oclAsType*(*OclAny*) , *R21* .*oclAsType*(*OclAny*) , *F1* .*oclAsType*(*OclAny*) , *F2* .*oclAsType*(*OclAny*)}
unfolding *OclAllInstances-at-post-def*
by(*rule* σ_1*-OclAllInstances-generic-exec-OclAny, simp-all only: assms, simp-all*)

lemma σ_1*-OclAllInstances-at-pre-exec-OclAny* :
assumes [*simp*]: (*OclAny* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Client}$ (*C1*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Client}$ (*C2*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Reservation}$ (*R21*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Staff}$ (*S1*$_{Staff}$))))⌋) = ((((λ-. ⌊⌊*S1*$_{Staff}$⌋⌋)::·*Staff*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Client}$ (*C1*$_{Client}$))))⌋) = ((((λ-. ⌊⌊*C1*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Client}$ (*C2*$_{Client}$))))⌋) = ((((λ-. ⌊⌊*C2*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$))))⌋) = ((((λ-. ⌊⌊*R11*$_{Reservation}$⌋⌋)::·*Reservation*))

.oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Reservation}$ (R21$_{Reservation}$)))))⌋) = ((((λ-. ⌊⌊R21$_{Reservation}$⌋⌋)::·Reservation)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Flight}$ (F1$_{Flight}$)))))⌋) = ((((λ-. ⌊⌊F1$_{Flight}$⌋⌋)::·Flight)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Flight}$ (F2$_{Flight}$)))))⌋) = ((((λ-. ⌊⌊F2$_{Flight}$⌋⌋)::·Flight)) .oclAsType(OclAny))
shows (σ$_1$, st) ⊨ (OclAllInstances-at-pre (OclAny)) ≐ Set{S1 .oclAsType(OclAny) , C1 .oclAsType(OclAny) , C2 .oclAsType(OclAny) , R11 .oclAsType(OclAny) , R21 .oclAsType(OclAny) , F1 .oclAsType(OclAny) , F2 .oclAsType(OclAny)}
unfolding OclAllInstances-at-pre-def
by(rule σ$_1$-OclAllInstances-generic-exec-OclAny, simp-all only: assms, simp-all)

ML ‹(Ty'.check ([]) (error(s)))›
end

definition (state-interpretation-σ$_1$ (τ)) = (state-σ$_1$ (oid3) (oid4) (oid5) (oid6) (oid7) (oid8) (oid9) (⌈⌈(S1 (τ))⌉⌉) (S1) (⌈⌈(C1 (τ))⌉⌉) (C1) (⌈⌈(C2 (τ))⌉⌉) (C2) (⌈⌈(R11 (τ))⌉⌉) (R11) (⌈⌈(R21 (τ))⌉⌉) (R21) (⌈⌈(F1 (τ))⌉⌉) (F1) (⌈⌈(F2 (τ))⌉⌉) (F2))

C.2 Instance

definition (typecheck-instance-bad-head-on-lhs-σ$_2$-object7-σ$_2$-object4-σ$_2$-object2-σ$_2$-object1 (σ$_2$-object7) (σ$_2$-object4) (σ$_2$-object2) (σ$_2$-object1)) = ()
definition typecheck-instance-extra-variables-on-rhs-σ$_2$-object7-σ$_2$-object4-σ$_2$-object2-σ$_2$-object1 = (λσ$_2$-object7 σ$_2$-object4 σ$_2$-object2 σ$_2$-object1. (F2 , R21 , F1 , C2 , F2 , F1 , C1 , R11 , F1))

definition oid10 = 10

definition σ$_2$-object1$_{Client}$ = (mk$_{Client}$ ((mk$\mathcal{EXT}_{Client}$ (oid4) (⌊''Bertha''⌋))) (⌊(let c = char-of-nat in CHR ''S'' # CHR ''a'' # CHR ''i'' # CHR ''n'' # CHR ''t'' # c 045 # CHR ''M'' # CHR ''a'' # CHR ''l'' # CHR ''o'' # [])⌋)))
definition (σ$_2$-object1::·Client) = ((λ-. ⌊⌊σ$_2$-object1$_{Client}$⌋⌋))
definition σ$_2$-object2$_{Client}$ = (mk$_{Client}$ ((mk$\mathcal{EXT}_{Client}$ (oid5) (⌊''Arthur''⌋))) (⌊''Valencia''⌋))
definition (σ$_2$-object2::·Client) = ((λ-. ⌊⌊σ$_2$-object2$_{Client}$⌋⌋))
definition σ$_2$-object4$_{Reservation}$ = (mk$_{Reservation}$ ((mk$\mathcal{EXT}_{Reservation}$ (oid7))) (⌊98765⌋) (None) (⌊oid8⌋))
definition (σ$_2$-object4::·Reservation) = ((λ-. ⌊⌊σ$_2$-object4$_{Reservation}$⌋⌋))
definition σ$_2$-object7$_{Reservation}$ = (mk$_{Reservation}$ ((mk$\mathcal{EXT}_{Reservation}$ (oid10))) (⌊19283⌋) (None) (⌊oid9⌋))
definition (σ$_2$-object7::·Reservation) = ((λ-. ⌊⌊σ$_2$-object7$_{Reservation}$⌋⌋))

ML ‹(Ty'.check ([(META.Writeln , σ$_2$-object1 .flights ≅ Set{ /*8*/ }) , (META.Writeln , σ$_2$-object1 .cl-res ≅ Set{ /*6*/ }) , (META.Writeln , σ$_2$-object2 .flights ≅ Set{ /*8*/ , /*9*/ }) , (META.Writeln , σ$_2$-object2 .cl-res ≅ Set{ σ$_2$-object4 , σ$_2$-object7 }) , (META.Writeln , σ$_2$-object4 .flight ≅ Set{ /*8*/ }) , (META.Writeln , σ$_2$-object4 .client ≅ Set{ σ$_2$-object2 }) , (META.Writeln , σ$_2$-object4 .prev ≅ Set{}) , (META.Writeln , σ$_2$-object4 .next ≅ Set{ σ$_2$-object7 }) , (META.Writeln , σ$_2$-object7 .flight ≅ Set{ /*9*/ }) , (META.Writeln , σ$_2$-object7 .client ≅ Set{ σ$_2$-object2 }) , (META.Writeln , σ$_2$-object7 .prev ≅ Set{ σ$_2$-object4 }) , (META.Writeln , σ$_2$-object7 .next ≅ Set{})]) (error(s)))›

C.3 State (Floor 2)

locale state-σ$_2$ =
fixes oid3 :: nat
fixes oid4 :: nat
fixes oid5 :: nat
fixes oid6 :: nat
fixes oid7 :: nat
fixes oid8 :: nat
fixes oid9 :: nat
fixes oid10 :: nat
assumes distinct-oid: (distinct ([oid3 , oid4 , oid5 , oid6 , oid7 , oid8 , oid9 , oid10]))
fixes S1$_{Staff}$ :: ty$_{Staff}$
fixes S1 :: ·Staff
assumes S1-def: S1 = (λ-. ⌊⌊S1$_{Staff}$⌋⌋)
fixes σ$_2$-object1$_{Client}$ :: ty$_{Client}$
fixes σ$_2$-object1 :: ·Client
assumes σ$_2$-object1-def: σ$_2$-object1 = (λ-. ⌊⌊σ$_2$-object1$_{Client}$⌋⌋)
fixes σ$_2$-object2$_{Client}$ :: ty$_{Client}$
fixes σ$_2$-object2 :: ·Client
assumes σ$_2$-object2-def: σ$_2$-object2 = (λ-. ⌊⌊σ$_2$-object2$_{Client}$⌋⌋)
fixes R11$_{Reservation}$ :: ty$_{Reservation}$
fixes R11 :: ·Reservation
assumes R11-def: R11 = (λ-. ⌊⌊R11$_{Reservation}$⌋⌋)
fixes σ$_2$-object4$_{Reservation}$ :: ty$_{Reservation}$

fixes $\sigma_2\text{-}object4$ :: $\cdot Reservation$
assumes $\sigma_2\text{-}object4\text{-}def$: $\sigma_2\text{-}object4 = (\lambda\text{-}.\ \lfloor\lfloor \sigma_2\text{-}object4_{Reservation} \rfloor\rfloor)$
fixes $F1_{Flight}$ :: ty_{Flight}
fixes $F1$ :: $\cdot Flight$
assumes $F1\text{-}def$: $F1 = (\lambda\text{-}.\ \lfloor\lfloor F1_{Flight} \rfloor\rfloor)$
fixes $F2_{Flight}$ :: ty_{Flight}
fixes $F2$ :: $\cdot Flight$
assumes $F2\text{-}def$: $F2 = (\lambda\text{-}.\ \lfloor\lfloor F2_{Flight} \rfloor\rfloor)$
fixes $\sigma_2\text{-}object7_{Reservation}$ :: $ty_{Reservation}$
fixes $\sigma_2\text{-}object7$ :: $\cdot Reservation$
assumes $\sigma_2\text{-}object7\text{-}def$: $\sigma_2\text{-}object7 = (\lambda\text{-}.\ \lfloor\lfloor \sigma_2\text{-}object7_{Reservation} \rfloor\rfloor)$
begin
definition $\sigma_2 = (state.make\ ((Map.empty\ (oid3 \mapsto (in_{Staff}\ (S1_{Staff})))\ (oid4 \mapsto (in_{Client}\ (\sigma_2\text{-}object1_{Client})))\ (oid5 \mapsto (in_{Client}\ (\sigma_2\text{-}object2_{Client})))\ (oid6 \mapsto (in_{Reservation}\ (R11_{Reservation})))\ (oid7 \mapsto (in_{Reservation}\ (\sigma_2\text{-}object4_{Reservation})))\ (oid8 \mapsto (in_{Flight}\ (F1_{Flight})))\ (oid9 \mapsto (in_{Flight}\ (F2_{Flight})))\ (oid10 \mapsto (in_{Reservation}\ (\sigma_2\text{-}object7_{Reservation})))))\ ((map\text{-}of\text{-}list\ ([(oid_{Staff}\text{-}0\text{---}flights\ ,\ (List.map\ ((\lambda(x\ ,\ y).\ [x\ ,\ y])\ o\ switch_2\text{-}01)\ ([[[oid3]\ ,\ [oid8]]]))),\ (oid_{Reservation}\text{-}1\text{---}next\ ,\ (List.map\ ((\lambda(x\ ,\ y).\ [x\ ,\ y])\ o\ switch_2\text{-}10)\ ([[[oid7]\ ,\ [oid10]]]))),\ (oid_{Client}\text{-}0\text{---}flights\ ,\ (List.map\ ((\lambda(x\ ,\ y).\ [x\ ,\ y])\ o\ switch_2\text{-}01)\ ([[[oid4]\ ,\ [oid8]]\ ,\ [[oid5]\ ,\ [oid8\ ,\ oid9]]]))),\ (oid_{Client}\text{-}0\text{---}cl\text{-}res\ ,\ (List.map\ ((\lambda(x\ ,\ y).\ [x\ ,\ y])\ o\ switch_2\text{-}01)\ ([[[oid4]\ ,\ [oid6]]\ ,\ [[oid5]\ ,\ [oid7\ ,\ oid10]]]))))]))))$

lemma $dom\text{-}\sigma_2$: $(dom\ ((heap\ (\sigma_2)))) = \{oid3\ ,\ oid4\ ,\ oid5\ ,\ oid6\ ,\ oid7\ ,\ oid8\ ,\ oid9\ ,\ oid10\}$
by$(auto\ simp\text{:}\ \sigma_2\text{-}def)$

lemmas$[simp,code\text{-}unfold] = dom\text{-}\sigma_2$

lemma $perm\text{-}\sigma_2$: $\sigma_2 = (state.make\ ((Map.empty\ (oid10 \mapsto (in_{Reservation}\ (\sigma_2\text{-}object7_{Reservation})))\ (oid9 \mapsto (in_{Flight}\ (F2_{Flight})))\ (oid8 \mapsto (in_{Flight}\ (F1_{Flight})))\ (oid7 \mapsto (in_{Reservation}\ (\sigma_2\text{-}object4_{Reservation})))\ (oid6 \mapsto (in_{Reservation}\ (R11_{Reservation})))\ (oid5 \mapsto (in_{Client}\ (\sigma_2\text{-}object2_{Client})))\ (oid4 \mapsto (in_{Client}\ (\sigma_2\text{-}object1_{Client})))\ (oid3 \mapsto (in_{Staff}\ (S1_{Staff})))))\ ((assocs\ (\sigma_2))))$
apply$(simp\ add\text{:}\ \sigma_2\text{-}def)$
apply$(subst\ (1)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (2)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (1)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (3)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (2)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (1)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (4)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (3)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (2)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (1)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (5)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (4)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (3)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (2)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (1)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (6)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (5)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (4)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (3)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (2)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (1)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (7)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (6)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (5)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (4)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (3)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (2)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
apply$(subst\ (1)\ fun\text{-}upd\text{-}twist,\ metis\ distinct\text{-}oid\ distinct\text{-}length\text{-}2\text{-}or\text{-}more)$
by$(simp)$

lemma $\sigma_2\text{-}OclAllInstances\text{-}generic\text{-}exec\text{-}Flight$:
assumes $[simp]$: $(Flight\ ((in_{Staff}\ (S1_{Staff})))) = None$
assumes $[simp]$: $(Flight\ ((in_{Client}\ (\sigma_2\text{-}object1_{Client})))) = None$
assumes $[simp]$: $(Flight\ ((in_{Client}\ (\sigma_2\text{-}object2_{Client})))) = None$
assumes $[simp]$: $(Flight\ ((in_{Reservation}\ (R11_{Reservation})))) = None$
assumes $[simp]$: $(Flight\ ((in_{Reservation}\ (\sigma_2\text{-}object4_{Reservation})))) = None$
assumes $[simp]$: $(Flight\ ((in_{Flight}\ (F1_{Flight})))) \neq None$
assumes $[simp]$: $(Flight\ ((in_{Flight}\ (F2_{Flight})))) \neq None$
assumes $[simp]$: $(Flight\ ((in_{Reservation}\ (\sigma_2\text{-}object7_{Reservation})))) = None$
assumes $[simp]$: $(\bigwedge a.\ (pre\text{-}post\ ((mk\ (a)))) = a)$
shows $(mk\ (\sigma_2)) \models (OclAllInstances\text{-}generic\ (pre\text{-}post)\ (Flight)) \doteq Set\{F1\ ,\ F2\}$
apply$(subst\ perm\text{-}\sigma_2)$
apply$(simp\ only\text{:}\ state.make\text{-}def\ F1\text{-}def\ F2\text{-}def)$

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp*)

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp*)

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp*)

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp*)

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq*$_{Set}$*-empty, simp*)

apply(*rule state-update-vs-allInstances-generic-empty*)

by(*simp-all only: assms, (simp-all add: OclAsType*$_{Flight}$*-*$\mathfrak{A}$*-def)?*)

lemma σ_2*-OclAllInstances-at-post-exec-Flight* :
assumes [*simp*]: (*Flight* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Client}$ (σ_2*-object1*$_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Client}$ (σ_2*-object2*$_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Reservation}$ (σ_2*-object4*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*Flight* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*Flight* ((*in*$_{Reservation}$ (σ_2*-object7*$_{Reservation}$)))) = *None*
shows (*st* , σ_2) $\models$ (*OclAllInstances-at-post* (*Flight*)) $\doteq$ *Set*{*F1* , *F2*}
unfolding *OclAllInstances-at-post-def*
by(*rule* σ_2*-OclAllInstances-generic-exec-Flight, simp-all only: assms, simp-all*)

lemma σ_2*-OclAllInstances-at-pre-exec-Flight* :
assumes [*simp*]: (*Flight* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Client}$ (σ_2*-object1*$_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Client}$ (σ_2*-object2*$_{Client}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Reservation}$ (σ_2*-object4*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Flight* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*Flight* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*Flight* ((*in*$_{Reservation}$ (σ_2*-object7*$_{Reservation}$)))) = *None*
shows (σ_2 , *st*) $\models$ (*OclAllInstances-at-pre* (*Flight*)) $\doteq$ *Set*{*F1* , *F2*}
unfolding *OclAllInstances-at-pre-def*
by(*rule* σ_2*-OclAllInstances-generic-exec-Flight, simp-all only: assms, simp-all*)

lemma σ_2*-OclAllInstances-generic-exec-Client* :
assumes [*simp*]: (*Client* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) = *None*
assumes [*simp*]: (*Client* ((*in*$_{Client}$ (σ_2*-object1*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*Client* ((*in*$_{Client}$ (σ_2*-object2*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*Client* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Client* ((*in*$_{Reservation}$ (σ_2*-object4*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Client* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Client* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Client* ((*in*$_{Reservation}$ (σ_2*-object7*$_{Reservation}$)))) = *None*
assumes [*simp*]: ($\bigwedge$*a*. (*pre-post* ((*mk* (*a*)))) = *a*)
shows (*mk* (σ_2)) $\models$ (*OclAllInstances-generic* (*pre-post*) (*Client*)) $\doteq$ *Set*{σ_2*-object1* , σ_2*-object2*}
apply(*subst perm-*σ_2)
apply(*simp only: state.make-def* σ_2*-object1-def* σ_2*-object2-def*)

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq*$_{Set}$*-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only:*

assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(rule state-update-vs-allInstances-generic-empty)
by(simp-all only: assms, (simp-all add: OclAsType$_{Client}$-$\mathfrak{A}$-def)?)

lemma σ_2-OclAllInstances-at-post-exec-Client :
assumes [simp]: (Client ((in$_{Staff}$ (S1$_{Staff}$)))) = None
assumes [simp]: (Client ((in$_{Client}$ (σ_2-object1$_{Client}$)))) $\neq$ None
assumes [simp]: (Client ((in$_{Client}$ (σ_2-object2$_{Client}$)))) $\neq$ None
assumes [simp]: (Client ((in$_{Reservation}$ (R11$_{Reservation}$)))) = None
assumes [simp]: (Client ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$)))) = None
assumes [simp]: (Client ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Client ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (Client ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$)))) = None
shows (st , σ_2) $\models$ (OclAllInstances-at-post (Client)) $\doteq$ Set{σ_2-object1 , σ_2-object2}
unfolding OclAllInstances-at-post-def
by(rule σ_2-OclAllInstances-generic-exec-Client, simp-all only: assms, simp-all)

lemma σ_2-OclAllInstances-at-pre-exec-Client :
assumes [simp]: (Client ((in$_{Staff}$ (S1$_{Staff}$)))) = None
assumes [simp]: (Client ((in$_{Client}$ (σ_2-object1$_{Client}$)))) $\neq$ None
assumes [simp]: (Client ((in$_{Client}$ (σ_2-object2$_{Client}$)))) $\neq$ None
assumes [simp]: (Client ((in$_{Reservation}$ (R11$_{Reservation}$)))) = None
assumes [simp]: (Client ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$)))) = None
assumes [simp]: (Client ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Client ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (Client ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$)))) = None
shows (σ_2 , st) $\models$ (OclAllInstances-at-pre (Client)) $\doteq$ Set{σ_2-object1 , σ_2-object2}
unfolding OclAllInstances-at-pre-def
by(rule σ_2-OclAllInstances-generic-exec-Client, simp-all only: assms, simp-all)

lemma σ_2-OclAllInstances-generic-exec-Staff :
assumes [simp]: (Staff ((in$_{Staff}$ (S1$_{Staff}$)))) $\neq$ None
assumes [simp]: (Staff ((in$_{Client}$ (σ_2-object1$_{Client}$)))) = None
assumes [simp]: (Staff ((in$_{Client}$ (σ_2-object2$_{Client}$)))) = None
assumes [simp]: (Staff ((in$_{Reservation}$ (R11$_{Reservation}$)))) = None
assumes [simp]: (Staff ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$)))) = None
assumes [simp]: (Staff ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Staff ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (Staff ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$)))) = None
assumes [simp]: ($\bigwedge$a. (pre-post ((mk (a)))) = a)
shows (mk (σ_2)) $\models$ (OclAllInstances-generic (pre-post) (Staff)) $\doteq$ Set{S1}
apply(subst perm-σ_2)
apply(simp only: state.make-def S1-def)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-empty, simp)
apply(rule state-update-vs-allInstances-generic-empty)
by(simp-all only: assms, (simp-all add: OclAsType$_{Staff}$-$\mathfrak{A}$-def)?)

lemma σ_2-OclAllInstances-at-post-exec-Staff :

assumes [*simp*]: (*Staff* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) ≠ *None*
assumes [*simp*]: (*Staff* ((*in*$_{Client}$ (σ_2-*object1*$_{Client}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Client}$ (σ_2-*object2*$_{Client}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Reservation}$ (σ_2-*object4*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Reservation}$ (σ_2-*object7*$_{Reservation}$)))) = *None*
shows (*st* , σ_2) ⊨ (*OclAllInstances-at-post* (*Staff*)) $\doteq$ *Set*{*S1*}
  unfolding *OclAllInstances-at-post-def*
by(*rule* σ_2-*OclAllInstances-generic-exec-Staff*, *simp-all only*: *assms*, *simp-all*)

lemma σ_2-*OclAllInstances-at-pre-exec-Staff* :
assumes [*simp*]: (*Staff* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) ≠ *None*
assumes [*simp*]: (*Staff* ((*in*$_{Client}$ (σ_2-*object1*$_{Client}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Client}$ (σ_2-*object2*$_{Client}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Reservation}$ (σ_2-*object4*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Staff* ((*in*$_{Reservation}$ (σ_2-*object7*$_{Reservation}$)))) = *None*
shows (σ_2 , *st*) ⊨ (*OclAllInstances-at-pre* (*Staff*)) $\doteq$ *Set*{*S1*}
  unfolding *OclAllInstances-at-pre-def*
by(*rule* σ_2-*OclAllInstances-generic-exec-Staff*, *simp-all only*: *assms*, *simp-all*)

lemma σ_2-*OclAllInstances-generic-exec-Person* :
assumes [*simp*]: (*Person* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) ≠ *None*
assumes [*simp*]: (*Person* ((*in*$_{Client}$ (σ_2-*object1*$_{Client}$)))) ≠ *None*
assumes [*simp*]: (*Person* ((*in*$_{Client}$ (σ_2-*object2*$_{Client}$)))) ≠ *None*
assumes [*simp*]: (*Person* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Person* ((*in*$_{Reservation}$ (σ_2-*object4*$_{Reservation}$)))) = *None*
assumes [*simp*]: (*Person* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Person* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) = *None*
assumes [*simp*]: (*Person* ((*in*$_{Reservation}$ (σ_2-*object7*$_{Reservation}$)))) = *None*
assumes [*simp*]: (λ-. ⌊(*Person* ((*in*$_{Staff}$ (*S1*$_{Staff}$))))⌋) = ((((λ-. ⌊⌊*S1*$_{Staff}$⌋⌋)::·*Staff*)) .*oclAsType*(*Person*))
assumes [*simp*]: (λ-. ⌊(*Person* ((*in*$_{Client}$ (σ_2-*object1*$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-*object1*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*Person*))
assumes [*simp*]: (λ-. ⌊(*Person* ((*in*$_{Client}$ (σ_2-*object2*$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-*object2*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*Person*))
assumes [*simp*]: (⋀*a*. (*pre-post* ((*mk* (*a*)))) = *a*)
shows (*mk* (σ_2)) ⊨ (*OclAllInstances-generic* (*pre-post*) (*Person*)) $\doteq$ *Set*{*S1* .*oclAsType*(*Person*) , σ_2-*object1* .*oclAsType*(*Person*) , σ_2-*object2* .*oclAsType*(*Person*)}
  apply(*subst perm*-σ_2)
  apply(*simp only*: *state.make-def S1-def* σ_2-*object1-def* σ_2-*object2-def*)

apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$-*including*, *simp del*: *OclAsType*$_{Person}$-*Staff OclAsType*$_{Person}$-*Client OclAsType*$_{Person}$-*Client*, *simp del*: *OclAsType*$_{Person}$-*Staff OclAsType*$_{Person}$-*Client OclAsType*$_{Person}$-*Client*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$-*including*, *simp del*: *OclAsType*$_{Person}$-*Staff OclAsType*$_{Person}$-*Client OclAsType*$_{Person}$-*Client*, *simp del*: *OclAsType*$_{Person}$-*Staff OclAsType*$_{Person}$-*Client OclAsType*$_{Person}$-*Client*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$-*including*, *simp del*: *OclAsType*$_{Person}$-*Staff OclAsType*$_{Person}$-*Client OclAsType*$_{Person}$-*Client*, *simp del*: *OclAsType*$_{Person}$-*Staff OclAsType*$_{Person}$-*Client OclAsType*$_{Person}$-*Client*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$-*empty*, *simp*)

apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$-*empty*, *simp*)

apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$-*empty*, *simp*)

apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$-*empty*, *simp*)

apply(*subst state-update-vs-allInstances-generic-ntc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *simp*, *rule const-StrictRefEq*$_{Set}$-*empty*, *simp*)

  apply(*rule state-update-vs-allInstances-generic-empty*)
by(*simp-all only*: *assms*, (*simp-all add*: *OclAsType*$_{Person}$-𝔄-*def*)?)

lemma σ_2-*OclAllInstances-at-post-exec-Person* :
assumes [*simp*]: (*Person* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) ≠ *None*
assumes [*simp*]: (*Person* ((*in*$_{Client}$ (σ_2-*object1*$_{Client}$)))) ≠ *None*

assumes [simp]: (Person ((in$_{Client}$ (σ_2-object2$_{Client}$)))) ≠ None
assumes [simp]: (Person ((in$_{Reservation}$ (R11$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (Person ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$)))) = None
assumes [simp]: (λ-. ⌊(Person ((in$_{Staff}$ (S1$_{Staff}$))))⌋) = ((((λ-. ⌊⌊S1$_{Staff}$⌋⌋)::→Staff)) .oclAsType(Person))
assumes [simp]: (λ-. ⌊(Person ((in$_{Client}$ (σ_2-object1$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-object1$_{Client}$⌋⌋)::→Client)) .oclAsType(Person))
assumes [simp]: (λ-. ⌊(Person ((in$_{Client}$ (σ_2-object2$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-object2$_{Client}$⌋⌋)::→Client)) .oclAsType(Person))
shows (st , σ_2) ⊨ (OclAllInstances-at-post (Person)) ≐ Set{S1 .oclAsType(Person) , σ_2-object1 .oclAsType(Person) , σ_2-object2 .oclAsType(Person)}
unfolding OclAllInstances-at-post-def
by(rule σ_2-OclAllInstances-generic-exec-Person, simp-all only: assms, simp-all)

lemma σ_2-OclAllInstances-at-pre-exec-Person :
assumes [simp]: (Person ((in$_{Staff}$ (S1$_{Staff}$)))) ≠ None
assumes [simp]: (Person ((in$_{Client}$ (σ_2-object1$_{Client}$)))) ≠ None
assumes [simp]: (Person ((in$_{Client}$ (σ_2-object2$_{Client}$)))) ≠ None
assumes [simp]: (Person ((in$_{Reservation}$ (R11$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Person ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (Person ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$)))) = None
assumes [simp]: (λ-. ⌊(Person ((in$_{Staff}$ (S1$_{Staff}$))))⌋) = ((((λ-. ⌊⌊S1$_{Staff}$⌋⌋)::→Staff)) .oclAsType(Person))
assumes [simp]: (λ-. ⌊(Person ((in$_{Client}$ (σ_2-object1$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-object1$_{Client}$⌋⌋)::→Client)) .oclAsType(Person))
assumes [simp]: (λ-. ⌊(Person ((in$_{Client}$ (σ_2-object2$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-object2$_{Client}$⌋⌋)::→Client)) .oclAsType(Person))
shows (σ_2 , st) ⊨ (OclAllInstances-at-pre (Person)) ≐ Set{S1 .oclAsType(Person) , σ_2-object1 .oclAsType(Person) , σ_2-object2 .oclAsType(Person)}
unfolding OclAllInstances-at-pre-def
by(rule σ_2-OclAllInstances-generic-exec-Person, simp-all only: assms, simp-all)

lemma σ_2-OclAllInstances-generic-exec-Reservation :
assumes [simp]: (Reservation ((in$_{Staff}$ (S1$_{Staff}$)))) = None
assumes [simp]: (Reservation ((in$_{Client}$ (σ_2-object1$_{Client}$)))) = None
assumes [simp]: (Reservation ((in$_{Client}$ (σ_2-object2$_{Client}$)))) = None
assumes [simp]: (Reservation ((in$_{Reservation}$ (R11$_{Reservation}$)))) ≠ None
assumes [simp]: (Reservation ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$)))) ≠ None
assumes [simp]: (Reservation ((in$_{Flight}$ (F1$_{Flight}$)))) = None
assumes [simp]: (Reservation ((in$_{Flight}$ (F2$_{Flight}$)))) = None
assumes [simp]: (Reservation ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$)))) ≠ None
assumes [simp]: (⋀a. (pre-post ((mk (a)))) = a)
shows (mk (σ_2)) ⊨ (OclAllInstances-generic (pre-post) (Reservation)) ≐ Set{R11 , σ_2-object4 , σ_2-object7}
apply(subst perm-σ_2)
apply(simp only: state.make-def R11-def σ_2-object4-def σ_2-object7-def)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)
apply(subst state-update-vs-allInstances-generic-ntc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp)
apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp, simp, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)
apply(rule state-update-vs-allInstances-generic-empty)
by(simp-all only: assms, (simp-all add: OclAsType$_{Reservation}$-𝔄-def)?)

lemma σ_2-OclAllInstances-at-post-exec-Reservation :
assumes [simp]: (Reservation ((in$_{Staff}$ (S1$_{Staff}$)))) = None
assumes [simp]: (Reservation ((in$_{Client}$ (σ_2-object1$_{Client}$)))) = None
assumes [simp]: (Reservation ((in$_{Client}$ (σ_2-object2$_{Client}$)))) = None

assumes [*simp*]: (*Reservation* (($in_{Reservation}$ ($R11_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*Reservation* (($in_{Reservation}$ (σ_2-*object4*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*Reservation* ((in_{Flight} ($F1_{Flight}$)))) = *None*
assumes [*simp*]: (*Reservation* ((in_{Flight} ($F2_{Flight}$)))) = *None*
assumes [*simp*]: (*Reservation* (($in_{Reservation}$ (σ_2-*object7*$_{Reservation}$)))) $\neq$ *None*
shows (*st* , σ_2) $\models$ (*OclAllInstances-at-post* (*Reservation*)) $\doteq$ *Set*{*R11* , σ_2-*object4* , σ_2-*object7*}
unfolding *OclAllInstances-at-post-def*
by(*rule* σ_2-*OclAllInstances-generic-exec-Reservation*, *simp-all only*: *assms*, *simp-all*)

lemma σ_2-*OclAllInstances-at-pre-exec-Reservation* :
assumes [*simp*]: (*Reservation* ((in_{Staff} ($S1_{Staff}$)))) = *None*
assumes [*simp*]: (*Reservation* ((in_{Client} (σ_2-*object1*$_{Client}$)))) = *None*
assumes [*simp*]: (*Reservation* ((in_{Client} (σ_2-*object2*$_{Client}$)))) = *None*
assumes [*simp*]: (*Reservation* (($in_{Reservation}$ ($R11_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*Reservation* (($in_{Reservation}$ (σ_2-*object4*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*Reservation* ((in_{Flight} ($F1_{Flight}$)))) = *None*
assumes [*simp*]: (*Reservation* ((in_{Flight} ($F2_{Flight}$)))) = *None*
assumes [*simp*]: (*Reservation* (($in_{Reservation}$ (σ_2-*object7*$_{Reservation}$)))) $\neq$ *None*
shows (σ_2 , *st*) $\models$ (*OclAllInstances-at-pre* (*Reservation*)) $\doteq$ *Set*{*R11* , σ_2-*object4* , σ_2-*object7*}
unfolding *OclAllInstances-at-pre-def*
by(*rule* σ_2-*OclAllInstances-generic-exec-Reservation*, *simp-all only*: *assms*, *simp-all*)

lemma σ_2-*OclAllInstances-generic-exec-OclAny* :
assumes [*simp*]: (*OclAny* ((in_{Staff} ($S1_{Staff}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((in_{Client} (σ_2-*object1*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((in_{Client} (σ_2-*object2*$_{Client}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* (($in_{Reservation}$ ($R11_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* (($in_{Reservation}$ (σ_2-*object4*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((in_{Flight} ($F1_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* ((in_{Flight} ($F2_{Flight}$)))) $\neq$ *None*
assumes [*simp*]: (*OclAny* (($in_{Reservation}$ (σ_2-*object7*$_{Reservation}$)))) $\neq$ *None*
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* ((in_{Staff} ($S1_{Staff}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor S1_{Staff}\rfloor\rfloor$)::·*Staff*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* ((in_{Client} (σ_2-*object1*$_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor\sigma_2$-*object1*$_{Client}\rfloor\rfloor$)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* ((in_{Client} (σ_2-*object2*$_{Client}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor\sigma_2$-*object2*$_{Client}\rfloor\rfloor$)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* (($in_{Reservation}$ ($R11_{Reservation}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor R11_{Reservation}\rfloor\rfloor$)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* (($in_{Reservation}$ (σ_2-*object4*$_{Reservation}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor\sigma_2$-*object4*$_{Reservation}\rfloor\rfloor$)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* ((in_{Flight} ($F1_{Flight}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor F1_{Flight}\rfloor\rfloor$)::·*Flight*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* ((in_{Flight} ($F2_{Flight}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor F2_{Flight}\rfloor\rfloor$)::·*Flight*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. $\lfloor$(*OclAny* (($in_{Reservation}$ (σ_2-*object7*$_{Reservation}$))))$\rfloor$) = ((((λ-. $\lfloor\lfloor\sigma_2$-*object7*$_{Reservation}\rfloor\rfloor$)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: ($\bigwedge$*a*. (*pre-post* ((*mk* (*a*)))) = *a*)
shows (*mk* (σ_2)) $\models$ (*OclAllInstances-generic* (*pre-post*) (*OclAny*)) $\doteq$ *Set*{*S1* .*oclAsType*(*OclAny*) , σ_2-*object1* .*oclAsType*(*OclAny*) , σ_2-*object2* .*oclAsType*(*OclAny*) , *R11* .*oclAsType*(*OclAny*) , σ_2-*object4* .*oclAsType*(*OclAny*) , *F1* .*oclAsType*(*OclAny*) , *F2* .*oclAsType*(*OclAny*) , σ_2-*object7* .*oclAsType*(*OclAny*)}
apply(*subst perm*-σ_2)
apply(*simp only*: *state.make-def S1-def* σ_2-*object1-def* σ_2-*object2-def R11-def* σ_2-*object4-def F1-def F2-def* σ_2-*object7-def*)

apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp del*: *OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Reservation*, *simp del*: *OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Reservation*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp del*: *OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Reservation*, *simp del*: *OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Reservation*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp only*: *assms*, *blast*, *simp*, *rule const-StrictRefEq*$_{Set}$*-including*, *simp del*: *OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Reservation*, *simp del*: *OclAsType*$_{OclAny}$*-Staff OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Client OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Reservation OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Flight OclAsType*$_{OclAny}$*-Reservation*, *simp*, *rule OclIncluding-cong*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*, (*simp only*: *assms*[*symmetric*])?, *simp add*: *valid-def OclValid-def bot-fun-def bot-option-def*)

apply(*subst state-update-vs-allInstances-generic-tc*, *simp*, *simp*, (*metis distinct-oid distinct-length-2-or-more*)?, *simp*

only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(subst state-update-vs-allInstances-generic-tc, simp, simp, (metis distinct-oid distinct-length-2-or-more)?, simp only: assms, blast, simp, rule const-StrictRefEq$_{Set}$-including, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp del: OclAsType$_{OclAny}$-Staff OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Client OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Reservation OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Flight OclAsType$_{OclAny}$-Reservation, simp, rule OclIncluding-cong, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def, (simp only: assms[symmetric])?, simp add: valid-def OclValid-def bot-fun-def bot-option-def)

apply(rule state-update-vs-allInstances-generic-empty)
by(simp-all only: assms, (simp-all add: OclAsType$_{OclAny}$-𝔄-def)?)

lemma σ_2-OclAllInstances-at-post-exec-OclAny :
assumes [simp]: (OclAny ((in$_{Staff}$ (S1$_{Staff}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Client}$ (σ_2-object1$_{Client}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Client}$ (σ_2-object2$_{Client}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Reservation}$ (R11$_{Reservation}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Flight}$ (F1$_{Flight}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Flight}$ (F2$_{Flight}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$)))) ≠ None
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Staff}$ (S1$_{Staff}$))))⌋) = ((((λ-. ⌊⌊S1$_{Staff}$⌋⌋)::·Staff)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Client}$ (σ_2-object1$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-object1$_{Client}$⌋⌋)::·Client)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Client}$ (σ_2-object2$_{Client}$))))⌋) = ((((λ-. ⌊⌊σ_2-object2$_{Client}$⌋⌋)::·Client)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Reservation}$ (R11$_{Reservation}$))))⌋) = ((((λ-. ⌊⌊R11$_{Reservation}$⌋⌋)::·Reservation)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Reservation}$ (σ_2-object4$_{Reservation}$))))⌋) = ((((λ-. ⌊⌊σ_2-object4$_{Reservation}$⌋⌋)::·Reservation)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Flight}$ (F1$_{Flight}$))))⌋) = ((((λ-. ⌊⌊F1$_{Flight}$⌋⌋)::·Flight)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Flight}$ (F2$_{Flight}$))))⌋) = ((((λ-. ⌊⌊F2$_{Flight}$⌋⌋)::·Flight)) .oclAsType(OclAny))
assumes [simp]: (λ-. ⌊(OclAny ((in$_{Reservation}$ (σ_2-object7$_{Reservation}$))))⌋) = ((((λ-. ⌊⌊σ_2-object7$_{Reservation}$⌋⌋)::·Reservation)) .oclAsType(OclAny))
shows (st , σ_2) ⊨ (OclAllInstances-at-post (OclAny)) ≐ Set{S1 .oclAsType(OclAny) , σ_2-object1 .oclAsType(OclAny) , σ_2-object2 .oclAsType(OclAny) , R11 .oclAsType(OclAny) , σ_2-object4 .oclAsType(OclAny) , F1 .oclAsType(OclAny) , F2 .oclAsType(OclAny) , σ_2-object7 .oclAsType(OclAny)}
unfolding OclAllInstances-at-post-def
by(rule σ_2-OclAllInstances-generic-exec-OclAny, simp-all only: assms, simp-all)

lemma σ_2-OclAllInstances-at-pre-exec-OclAny :
assumes [simp]: (OclAny ((in$_{Staff}$ (S1$_{Staff}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Client}$ (σ_2-object1$_{Client}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Client}$ (σ_2-object2$_{Client}$)))) ≠ None
assumes [simp]: (OclAny ((in$_{Reservation}$ (R11$_{Reservation}$)))) ≠ None

assumes [*simp*]: (*OclAny* ((*in*$_{Reservation}$ (*σ2-object4*$_{Reservation}$)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) ≠ *None*
assumes [*simp*]: (*OclAny* ((*in*$_{Reservation}$ (*σ2-object7*$_{Reservation}$)))) ≠ *None*
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))⌋) = ((((λ-. ⌊⌊*S1*$_{Staff}$⌋⌋)::·*Staff*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Client}$ (*σ2-object1*$_{Client}$)))⌋) = ((((λ-. ⌊⌊*σ2-object1*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Client}$ (*σ2-object2*$_{Client}$)))⌋) = ((((λ-. ⌊⌊*σ2-object2*$_{Client}$⌋⌋)::·*Client*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))⌋) = ((((λ-. ⌊⌊*R11*$_{Reservation}$⌋⌋)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Reservation}$ (*σ2-object4*$_{Reservation}$)))⌋) = ((((λ-. ⌊⌊*σ2-object4*$_{Reservation}$⌋⌋)::·*Reservation*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))⌋) = ((((λ-. ⌊⌊*F1*$_{Flight}$⌋⌋)::·*Flight*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))⌋) = ((((λ-. ⌊⌊*F2*$_{Flight}$⌋⌋)::·*Flight*)) .*oclAsType*(*OclAny*))
assumes [*simp*]: (λ-. ⌊(*OclAny* ((*in*$_{Reservation}$ (*σ2-object7*$_{Reservation}$)))⌋) = ((((λ-. ⌊⌊*σ2-object7*$_{Reservation}$⌋⌋)::·*Reservation*)) .*oclAsType*(*OclAny*))
shows (*σ2* , *st*) ⊨ (*OclAllInstances-at-pre* (*OclAny*)) ≐ *Set*{*S1* .*oclAsType*(*OclAny*) , *σ2-object1* .*oclAsType*(*OclAny*) , *σ2-object2* .*oclAsType*(*OclAny*) , *R11* .*oclAsType*(*OclAny*) , *σ2-object4* .*oclAsType*(*OclAny*) , *F1* .*oclAsType*(*OclAny*) , *F2* .*oclAsType*(*OclAny*) , *σ2-object7* .*oclAsType*(*OclAny*)}
unfolding *OclAllInstances-at-pre-def*
by(*rule σ2-OclAllInstances-generic-exec-OclAny*, *simp-all only*: *assms*, *simp-all*)

ML ‹(*Ty'.check* ([]) (*error*(*s*)))›
end

definition (*state-interpretation-σ2* (*τ*)) = (*state-σ2* (*oid3*) (*oid4*) (*oid5*) (*oid6*) (*oid7*) (*oid8*) (*oid9*) (*oid10*) (⌈⌈(*S1* (*τ*))⌉⌉) (*S1*) (⌈⌈(*σ2-object1* (*τ*))⌉⌉) (*σ2-object1*) (⌈⌈(*σ2-object2* (*τ*))⌉⌉) (*σ2-object2*) (⌈⌈(*R11* (*τ*))⌉⌉) (*R11*) (⌈⌈(*σ2-object4* (*τ*))⌉⌉) (*σ2-object4*) (⌈⌈(*F1* (*τ*))⌉⌉) (*F1*) (⌈⌈(*F2* (*τ*))⌉⌉) (*F2*) (⌈⌈(*σ2-object7* (*τ*))⌉⌉) (*σ2-object7*))

C.4 Transition (Floor 2)

locale *transition-σ1-σ2* =
fixes *oid3* :: *nat*
fixes *oid4* :: *nat*
fixes *oid5* :: *nat*
fixes *oid6* :: *nat*
fixes *oid7* :: *nat*
fixes *oid8* :: *nat*
fixes *oid9* :: *nat*
fixes *oid10* :: *nat*
assumes *distinct-oid*: (*distinct* ([*oid3* , *oid4* , *oid5* , *oid6* , *oid7* , *oid8* , *oid9* , *oid10*]))
fixes *S1*$_{Staff}$ :: *ty*$_{Staff}$
fixes *S1* :: ·*Staff*
assumes *S1-def*: *S1* = (λ-. ⌊⌊*S1*$_{Staff}$⌋⌋)
fixes *σ2-object1*$_{Client}$ :: *ty*$_{Client}$
fixes *σ2-object1* :: ·*Client*
assumes *σ2-object1-def*: *σ2-object1* = (λ-. ⌊⌊*σ2-object1*$_{Client}$⌋⌋)
fixes *C1*$_{Client}$ :: *ty*$_{Client}$
fixes *C1* :: ·*Client*
assumes *C1-def*: *C1* = (λ-. ⌊⌊*C1*$_{Client}$⌋⌋)
fixes *σ2-object2*$_{Client}$ :: *ty*$_{Client}$
fixes *σ2-object2* :: ·*Client*
assumes *σ2-object2-def*: *σ2-object2* = (λ-. ⌊⌊*σ2-object2*$_{Client}$⌋⌋)
fixes *C2*$_{Client}$ :: *ty*$_{Client}$
fixes *C2* :: ·*Client*
assumes *C2-def*: *C2* = (λ-. ⌊⌊*C2*$_{Client}$⌋⌋)
fixes *R11*$_{Reservation}$ :: *ty*$_{Reservation}$
fixes *R11* :: ·*Reservation*
assumes *R11-def*: *R11* = (λ-. ⌊⌊*R11*$_{Reservation}$⌋⌋)
fixes *σ2-object4*$_{Reservation}$ :: *ty*$_{Reservation}$
fixes *σ2-object4* :: ·*Reservation*
assumes *σ2-object4-def*: *σ2-object4* = (λ-. ⌊⌊*σ2-object4*$_{Reservation}$⌋⌋)
fixes *R21*$_{Reservation}$ :: *ty*$_{Reservation}$
fixes *R21* :: ·*Reservation*
assumes *R21-def*: *R21* = (λ-. ⌊⌊*R21*$_{Reservation}$⌋⌋)
fixes *F1*$_{Flight}$ :: *ty*$_{Flight}$
fixes *F1* :: ·*Flight*
assumes *F1-def*: *F1* = (λ-. ⌊⌊*F1*$_{Flight}$⌋⌋)
fixes *F2*$_{Flight}$ :: *ty*$_{Flight}$
fixes *F2* :: ·*Flight*
assumes *F2-def*: *F2* = (λ-. ⌊⌊*F2*$_{Flight}$⌋⌋)
fixes *σ2-object7*$_{Reservation}$ :: *ty*$_{Reservation}$

fixes σ_2-*object7* :: ·*Reservation*
assumes σ_2-*object7-def*: σ_2-*object7* = (λ-. ⌊⌊σ_2-*object7*$_{Reservation}$⌋⌋)

assumes σ_1: (*state-*σ_1 (*oid3*) (*oid4*) (*oid5*) (*oid6*) (*oid7*) (*oid8*) (*oid9*) (*S1*$_{Staff}$) (*S1*) (*C1*$_{Client}$) (*C1*) (*C2*$_{Client}$) (*C2*) (*R11*$_{Reservation}$) (*R11*) (*R21*$_{Reservation}$) (*R21*) (*F1*$_{Flight}$) (*F1*) (*F2*$_{Flight}$) (*F2*))

assumes σ_2: (*state-*σ_2 (*oid3*) (*oid4*) (*oid5*) (*oid6*) (*oid7*) (*oid8*) (*oid9*) (*oid10*) (*S1*$_{Staff}$) (*S1*) (σ_2-*object1*$_{Client}$) (σ_2-*object1*) (σ_2-*object2*$_{Client}$) (σ_2-*object2*) (*R11*$_{Reservation}$) (*R11*) (σ_2-*object4*$_{Reservation}$) (σ_2-*object4*) (*F1*$_{Flight}$) (*F1*) (*F2*$_{Flight}$) (*F2*) (σ_2-*object7*$_{Reservation}$) (σ_2-*object7*))
begin
interpretation *state-*σ_1: *state-*σ_1 *oid3 oid4 oid5 oid6 oid7 oid8 oid9 S1*$_{Staff}$ *S1 C1*$_{Client}$ *C1 C2*$_{Client}$ *C2 R11*$_{Reservation}$ *R11 R21*$_{Reservation}$ *R21 F1*$_{Flight}$ *F1 F2*$_{Flight}$ *F2*
by(*rule* σ_1)

interpretation *state-*σ_2: *state-*σ_2 *oid3 oid4 oid5 oid6 oid7 oid8 oid9 oid10 S1*$_{Staff}$ *S1* σ_2-*object1*$_{Client}$ σ_2-*object1* σ_2-*object2*$_{Client}$ σ_2-*object2 R11*$_{Reservation}$ *R11* σ_2-*object4*$_{Reservation}$ σ_2-*object4 F1*$_{Flight}$ *F1 F2*$_{Flight}$ *F2* σ_2-*object7*$_{Reservation}$ σ_2-*object7*
by(*rule* σ_2)

definition σ_1 = *state-*σ_1.σ_1

definition σ_2 = *state-*σ_2.σ_2

lemma *basic-*σ_1*-*σ_2*-wff* :
assumes [*simp*]: (*oid-of* ((*in*$_{Staff}$ (*S1*$_{Staff}$)))) = *oid3*
assumes [*simp*]: (*oid-of* ((*in*$_{Client}$ (σ_2-*object1*$_{Client}$)))) = *oid4*
assumes [*simp*]: (*oid-of* ((*in*$_{Client}$ (*C1*$_{Client}$)))) = *oid4*
assumes [*simp*]: (*oid-of* ((*in*$_{Client}$ (σ_2-*object2*$_{Client}$)))) = *oid5*
assumes [*simp*]: (*oid-of* ((*in*$_{Client}$ (*C2*$_{Client}$)))) = *oid5*
assumes [*simp*]: (*oid-of* ((*in*$_{Reservation}$ (*R11*$_{Reservation}$)))) = *oid6*
assumes [*simp*]: (*oid-of* ((*in*$_{Reservation}$ (σ_2-*object4*$_{Reservation}$)))) = *oid7*
assumes [*simp*]: (*oid-of* ((*in*$_{Reservation}$ (*R21*$_{Reservation}$)))) = *oid7*
assumes [*simp*]: (*oid-of* ((*in*$_{Flight}$ (*F1*$_{Flight}$)))) = *oid8*
assumes [*simp*]: (*oid-of* ((*in*$_{Flight}$ (*F2*$_{Flight}$)))) = *oid9*
assumes [*simp*]: (*oid-of* ((*in*$_{Reservation}$ (σ_2-*object7*$_{Reservation}$)))) = *oid10*
shows (*WFF* ((*state-*σ_1.σ_1 , *state-*σ_2.σ_2)))
proof – have [*simp*]: *oid3* ≠ *oid4* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid3* ≠ *oid5* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid3* ≠ *oid6* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid3* ≠ *oid7* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid3* ≠ *oid8* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid3* ≠ *oid9* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid3* ≠ *oid10* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid4* ≠ *oid3* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid4* ≠ *oid5* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid4* ≠ *oid6* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid4* ≠ *oid7* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid4* ≠ *oid8* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid4* ≠ *oid9* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid4* ≠ *oid10* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid5* ≠ *oid3* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid5* ≠ *oid4* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid5* ≠ *oid6* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid5* ≠ *oid7* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid5* ≠ *oid8* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid5* ≠ *oid9* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid5* ≠ *oid10* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid6* ≠ *oid3* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid6* ≠ *oid4* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid6* ≠ *oid5* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid6* ≠ *oid7* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid6* ≠ *oid8* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid6* ≠ *oid9* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid6* ≠ *oid10* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid7* ≠ *oid3* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid7* ≠ *oid4* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid7* ≠ *oid5* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid7* ≠ *oid6* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid7* ≠ *oid8* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid7* ≠ *oid9* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid7* ≠ *oid10* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid8* ≠ *oid3* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*
proof – have [*simp*]: *oid8* ≠ *oid4* by(*metis distinct-oid distinct-length-2-or-more*) show *?thesis*

proof – have [simp]: $oid8 \neq oid5$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid8 \neq oid6$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid8 \neq oid7$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid8 \neq oid9$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid8 \neq oid10$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid9 \neq oid3$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid9 \neq oid4$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid9 \neq oid5$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid9 \neq oid6$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid9 \neq oid7$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid9 \neq oid8$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid9 \neq oid10$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid10 \neq oid3$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid10 \neq oid4$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid10 \neq oid5$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid10 \neq oid6$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid10 \neq oid7$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid10 \neq oid8$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
proof – have [simp]: $oid10 \neq oid9$ by(metis distinct-oid distinct-length-2-or-more) show ?thesis
by(auto simp: WFF-def state-σ_1.σ_1-def state-σ_2.σ_2-def) qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed
qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed
qed qed qed qed qed qed qed qed qed qed qed qed qed qed qed

lemma oid3$\sigma_1\sigma_2$-σ_1-OclIsMaintained :
assumes [simp]: (oid-of ($S1_{Staff}$)) = oid3
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (S1))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def S1-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid3$\sigma_1\sigma_2$-σ_2-OclIsMaintained :
assumes [simp]: (oid-of ($S1_{Staff}$)) = oid3
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (S1))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def S1-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid4$\sigma_1\sigma_2$-σ_1-OclIsMaintained :
assumes [simp]: (oid-of ($C1_{Client}$)) = oid4
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (C1))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def C1-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid4$\sigma_1\sigma_2$-σ_2-OclIsMaintained :
assumes [simp]: (oid-of (σ_2-object1$_{Client}$)) = oid4
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (σ_2-object1))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def σ_2-object1-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid5$\sigma_1\sigma_2$-σ_1-OclIsMaintained :
assumes [simp]: (oid-of ($C2_{Client}$)) = oid5
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (C2))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def C2-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid5$\sigma_1\sigma_2$-σ_2-OclIsMaintained :
assumes [simp]: (oid-of (σ_2-object2$_{Client}$)) = oid5
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (σ_2-object2))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def σ_2-object2-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid6$\sigma_1\sigma_2$-σ_1-OclIsMaintained :
assumes [simp]: (oid-of ($R11_{Reservation}$)) = oid6
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (R11))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def R11-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid6$\sigma_1\sigma_2$-σ_2-OclIsMaintained :
assumes [simp]: (oid-of ($R11_{Reservation}$)) = oid6
shows (state-σ_1.σ_1 , state-σ_2.σ_2) $\models$ (OclIsMaintained (R11))
apply(simp add: state-σ_1.σ_1-def state-σ_2.σ_2-def R11-def OclIsMaintained-def OclValid-def oid-of-option-def)
by((metis distinct-oid distinct-length-2-or-more)?)

lemma oid7$\sigma_1\sigma_2$-σ_1-OclIsMaintained :
assumes [simp]: (oid-of ($R21_{Reservation}$)) = oid7

shows (*state-σ_1.σ_1* , *state-σ_2.σ_2*) ⊨ (*OclIsMaintained* (*R21*))
apply(*simp add*: *state-σ_1.σ_1-def state-σ_2.σ_2-def R21-def OclIsMaintained-def OclValid-def oid-of-option-def*)
by((*metis distinct-oid distinct-length-2-or-more*)*?*)

lemma *oid7$\sigma_1\sigma_2$-σ_2-OclIsMaintained* :
assumes [*simp*]: (*oid-of* (*σ_2-object4$_{Reservation}$*)) = *oid7*
shows (*state-σ_1.σ_1* , *state-σ_2.σ_2*) ⊨ (*OclIsMaintained* (*σ_2-object4*))
apply(*simp add*: *state-σ_1.σ_1-def state-σ_2.σ_2-def σ_2-object4-def OclIsMaintained-def OclValid-def oid-of-option-def*)
by((*metis distinct-oid distinct-length-2-or-more*)*?*)

lemma *oid8$\sigma_1\sigma_2$-σ_1-OclIsMaintained* :
assumes [*simp*]: (*oid-of* (*F1$_{Flight}$*)) = *oid8*
shows (*state-σ_1.σ_1* , *state-σ_2.σ_2*) ⊨ (*OclIsMaintained* (*F1*))
apply(*simp add*: *state-σ_1.σ_1-def state-σ_2.σ_2-def F1-def OclIsMaintained-def OclValid-def oid-of-option-def*)
by((*metis distinct-oid distinct-length-2-or-more*)*?*)

lemma *oid8$\sigma_1\sigma_2$-σ_2-OclIsMaintained* :
assumes [*simp*]: (*oid-of* (*F1$_{Flight}$*)) = *oid8*
shows (*state-σ_1.σ_1* , *state-σ_2.σ_2*) ⊨ (*OclIsMaintained* (*F1*))
apply(*simp add*: *state-σ_1.σ_1-def state-σ_2.σ_2-def F1-def OclIsMaintained-def OclValid-def oid-of-option-def*)
by((*metis distinct-oid distinct-length-2-or-more*)*?*)

lemma *oid9$\sigma_1\sigma_2$-σ_1-OclIsMaintained* :
assumes [*simp*]: (*oid-of* (*F2$_{Flight}$*)) = *oid9*
shows (*state-σ_1.σ_1* , *state-σ_2.σ_2*) ⊨ (*OclIsMaintained* (*F2*))
apply(*simp add*: *state-σ_1.σ_1-def state-σ_2.σ_2-def F2-def OclIsMaintained-def OclValid-def oid-of-option-def*)
by((*metis distinct-oid distinct-length-2-or-more*)*?*)

lemma *oid9$\sigma_1\sigma_2$-σ_2-OclIsMaintained* :
assumes [*simp*]: (*oid-of* (*F2$_{Flight}$*)) = *oid9*
shows (*state-σ_1.σ_1* , *state-σ_2.σ_2*) ⊨ (*OclIsMaintained* (*F2*))
apply(*simp add*: *state-σ_1.σ_1-def state-σ_2.σ_2-def F2-def OclIsMaintained-def OclValid-def oid-of-option-def*)
by((*metis distinct-oid distinct-length-2-or-more*)*?*)

lemma *oid10$\sigma_1\sigma_2$-σ_2-OclIsNew* :
assumes [*simp*]: (*oid-of* (*σ_2-object7$_{Reservation}$*)) = *oid10*
shows (*state-σ_1.σ_1* , *state-σ_2.σ_2*) ⊨ (*OclIsNew* (*σ_2-object7*))
apply(*simp add*: *state-σ_1.σ_1-def state-σ_2.σ_2-def σ_2-object7-def OclIsNew-def OclValid-def oid-of-option-def*)
by((*metis distinct-oid distinct-length-2-or-more*)*?*)
end

definition (*pp-σ_1-σ_2* (*τ*)) = (*transition-σ_1-σ_2* (*oid3*) (*oid4*) (*oid5*) (*oid6*) (*oid7*) (*oid8*) (*oid9*) (*oid10*) (⌈⌈(*S1* (*τ*))⌉⌉) (*S1*) (⌈⌈(*σ_2-object1* (*τ*))⌉⌉) (*σ_2-object1*) (⌈⌈(*C1* (*τ*))⌉⌉) (*C1*) (⌈⌈(*σ_2-object2* (*τ*))⌉⌉) (*σ_2-object2*) (⌈⌈(*C2* (*τ*))⌉⌉) (*C2*) (⌈⌈(*R11* (*τ*))⌉⌉) (*R11*) (⌈⌈(*σ_2-object4* (*τ*))⌉⌉) (*σ_2-object4*) (⌈⌈(*R21* (*τ*))⌉⌉) (*R21*) (⌈⌈(*F1* (*τ*))⌉⌉) (*F1*) (⌈⌈(*F2* (*τ*))⌉⌉) (*F2*) (⌈⌈(*σ_2-object7* (*τ*))⌉⌉) (*σ_2-object7*))

lemmas *pp-oid-σ_1-σ_2* = *oid3-def*
oid4-def
oid5-def
oid6-def
oid7-def
oid8-def
oid9-def
oid10-def
lemmas *pp-object-σ_1-σ_2* = *S1-def*
σ_2-object1-def
C1-def
σ_2-object2-def
C2-def
R11-def
σ_2-object4-def
R21-def
F1-def
F2-def
σ_2-object7-def
lemmas *pp-object-ty-σ_1-σ_2* = *S1$_{Staff}$-def*
σ_2-object1$_{Client}$-def
C1$_{Client}$-def
σ_2-object2$_{Client}$-def
C2$_{Client}$-def
R11$_{Reservation}$-def

σ_2-$object4_{Reservation}$-def
$R21_{Reservation}$-def
$F1_{Flight}$-def
$F2_{Flight}$-def
σ_2-$object7_{Reservation}$-def

end

HOL-OCL 2.0: The Overall Architecture

All figures of this chapter have been generated from the respective graphs internally generated by the command thy_deps [Wen16b].

Figure D.1 is producing at the end one generated file, but two green boxes are depicted because the overall theories imported by this generated file depends on if it is expected for this file to generate another file or not. So we basically have two situations:

- "model generated (1)" represents the case where the file we are generating does not contain meta-commands (so no dependencies are set to the main entry-point of the meta-tool),
- whereas "model generated (2)" depends on all components of the meta-tool for itself to be able to generate another model, or just call particular type-checking functions defined in the library of the meta-tool.

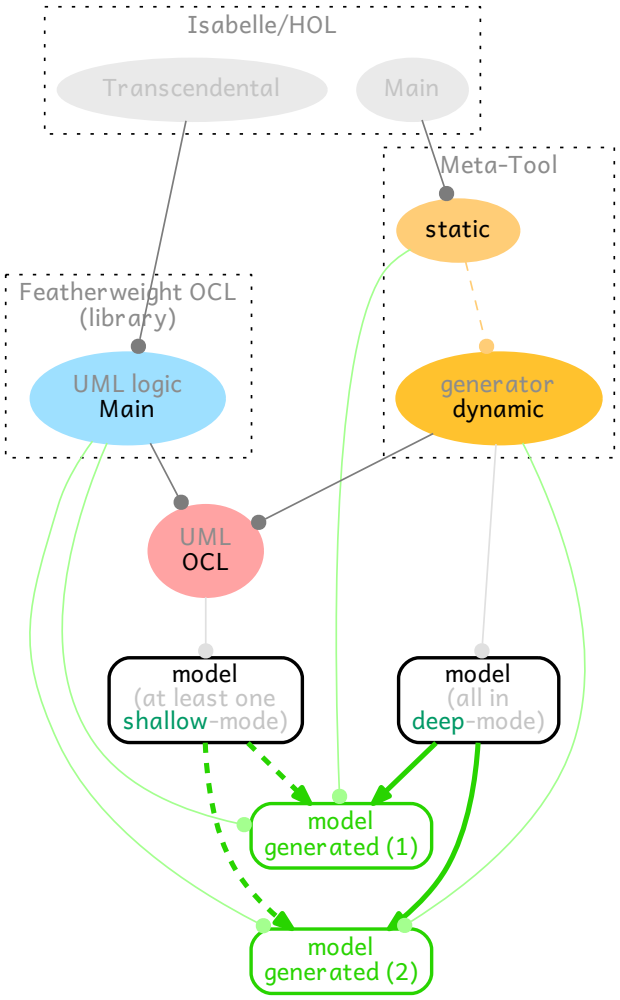


Figure D.1: An overall view of HOL-OCL 2.0

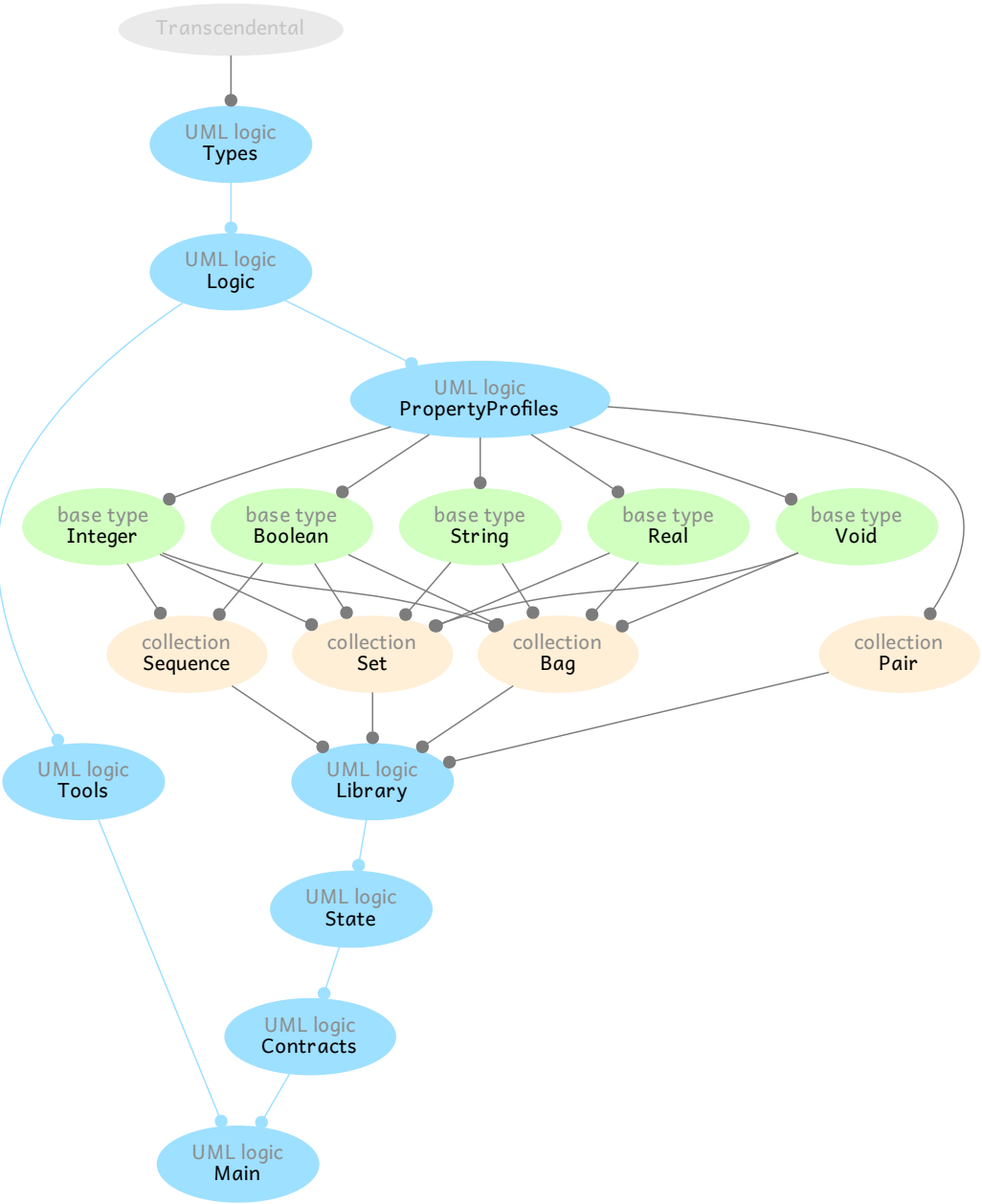


Figure D.2: The library of Featherweight OCL

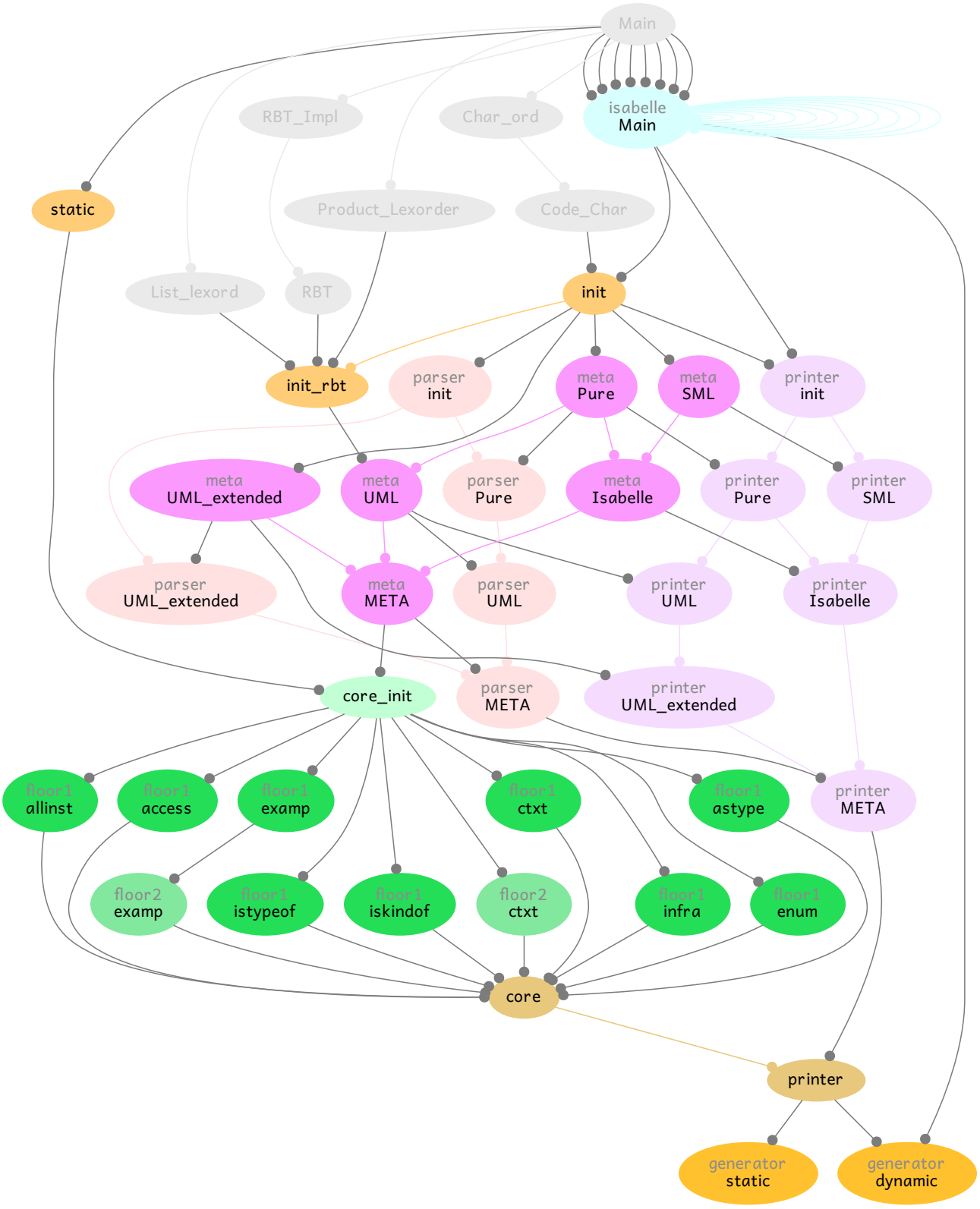


Figure D.3: The meta-tool

HOL-OCL 2.0: Defining Meta-Models

This chapter complements the chapter "Defining Meta-Models" of the document "A Meta-Model for the Isabelle API" [TW15].

E.1 OCL Meta-Model aka. AST definition of OCL (I)

theory *Meta-UML*
imports *../../compiler-generic/meta-isabelle/Meta-Pure*
../Init-rbt
begin

Type Definition

datatype *ocl-collection* = *Set*
| *Sequence*
| *Ordered0*
| *Subsets0*
| *Union0*
| *Redefines0*
| *Derived0*
| *Qualifier0*
| *Nonunique0*

datatype *ocl-multiplicity-single* = *Mult-nat nat*
| *Mult-star*
| *Mult-infinity*

record *ocl-multiplicity* = *TyMult* :: (*ocl-multiplicity-single* × *ocl-multiplicity-single option*) *list*
TyRole :: *string option*
TyCollect :: *ocl-collection list*

record *ocl-ty-class-node* = *TyObjN-ass-switch* :: *nat*
TyObjN-role-multip :: *ocl-multiplicity*
TyObjN-role-ty :: *string*
record *ocl-ty-class* = *TyObj-name* :: *string*
TyObj-ass-id :: *nat*
TyObj-ass-arity :: *nat*
TyObj-from :: *ocl-ty-class-node*
TyObj-to :: *ocl-ty-class-node*
datatype *ocl-ty-obj-core* = *OclTyCore-pre string*
| *OclTyCore ocl-ty-class*
datatype *ocl-ty-obj* = *OclTyObj ocl-ty-obj-core*
ocl-ty-obj-core list (∗ *the 'and' semantics* ∗)
list (∗ *'x # ...' means 'x < ...'* ∗)
datatype *ocl-ty* = *OclTy-base-void*
| *OclTy-base-boolean*
| *OclTy-base-integer*
| *OclTy-base-unlimitednatural*
| *OclTy-base-real*
| *OclTy-base-string*
| *OclTy-object ocl-ty-obj*
| *OclTy-collection ocl-multiplicity ocl-ty*
| *OclTy-pair ocl-ty ocl-ty*

```
                    | OclTy-binding string option (* name *) × ocl-ty
                    | OclTy-arrow ocl-ty ocl-ty
                    | OclTy-class-syn string
                    | OclTy-enum string
                    | OclTy-raw string


datatype ocl-association-type = OclAssTy-native-attribute
                              | OclAssTy-association
                              | OclAssTy-composition
                              | OclAssTy-aggregation
datatype ocl-association-relation = OclAssRel (ocl-ty-obj × ocl-multiplicity) list
record ocl-association =        OclAss-type     :: ocl-association-type
                          OclAss-relation :: ocl-association-relation

datatype ocl-ctxt-prefix = OclCtxtPre | OclCtxtPost

datatype ocl-ctxt-term = T-pure term
                           string option
                       | T-to-be-parsed string
                                  string
                       | T-lambda string ocl-ctxt-term
datatype ocl-prop = OclProp-ctxt string option  ocl-ctxt-term

datatype ocl-ctxt-term-inv = T-inv bool  ocl-prop
datatype ocl-ctxt-term-pp = T-pp ocl-ctxt-prefix ocl-prop
                          | T-invariant ocl-ctxt-term-inv

record ocl-ctxt-pre-post = Ctxt-fun-name :: string
                           Ctxt-fun-ty :: ocl-ty
                           Ctxt-expr :: ocl-ctxt-term-pp list

datatype ocl-ctxt-clause = Ctxt-pp ocl-ctxt-pre-post
                         | Ctxt-inv ocl-ctxt-term-inv
record ocl-ctxt = Ctxt-param :: string list
                  Ctxt-ty :: ocl-ty-obj
                  Ctxt-clause :: ocl-ctxt-clause list

datatype ocl-class =   OclClass
                         string
                         (string (* name *) × ocl-ty) list
                         ocl-class list

record ocl-class-raw = ClassRaw-name :: ocl-ty-obj
                       ClassRaw-own :: (string (* name *) × ocl-ty) list
                       ClassRaw-clause :: ocl-ctxt-clause list
                       ClassRaw-abstract :: bool

datatype ocl-ass-class = OclAssClass ocl-association
                               ocl-class-raw

datatype ocl-class-synonym = OclClassSynonym string  ocl-ty

datatype ocl-enum = OclEnum string  string (* constructor name *) list
```

Extending the Meta-Model

```
definition T-lambdas = List.fold T-lambda
definition TyObjN-role-name = TyRole o TyObjN-role-multip
definition OclTy-class c = OclTy-object (OclTyObj (OclTyCore c) [])
definition OclTy-class-pre c = OclTy-object (OclTyObj (OclTyCore-pre c) [])
definition OclAss-relation' l = (case OclAss-relation l of OclAssRel l ⇒ l)

fun fold-pair-var where
  fold-pair-var f t accu = (case t of
   OclTy-pair t1 t2 ⇒ Option.bind (fold-pair-var f t1 accu) (fold-pair-var f t2)
 | OclTy-binding (Some v, t) ⇒ fold-pair-var f t (f (v, t) accu)
 | OclTy-binding (None, t) ⇒ fold-pair-var f t accu
 | OclTy-collection - t ⇒ fold-pair-var f t accu
 | OclTy-arrow - - ⇒ None
 | - ⇒ Some accu)

definition Ctxt-fun-ty-arg ctxt =
```

```
(case
   fold-pair-var
     Cons
     (case Ctxt-fun-ty ctxt of OclTy-arrow t - ⇒ t
                             | t ⇒ t)
     []
 of Some l ⇒ rev l)

definition Ctxt-fun-ty-out ctxt =
(case Ctxt-fun-ty ctxt of OclTy-arrow - t ⇒ Some t
                        | - ⇒ None)

definition map-pre-post f =
           Ctxt-clause-update
            (L.map
               (λ Ctxt-pp ctxt ⇒
                  Ctxt-pp (Ctxt-expr-update
                            (L.map
                               (λ T-pp pref (OclProp-ctxt n e) ⇒
                                  T-pp pref (OclProp-ctxt n (f pref ctxt e))
                                | x ⇒ x))
                            ctxt)
                | x ⇒ x))

definition fold-pre-post f ctxt =
           List.fold
            (λ Ctxt-pp ctxt ⇒
                f (rev (List.fold
                  (λ T-pp pref (OclProp-ctxt n e) ⇒ Cons (pref, n, e)
                   | - ⇒ id)
                  (Ctxt-expr ctxt) [])) ctxt
             | - ⇒ id)
            (Ctxt-clause ctxt)

definition map-invariant f-inv =
           Ctxt-clause-update
            (L.map
               (λ Ctxt-pp ctxt ⇒
                  Ctxt-pp (Ctxt-expr-update
                            (L.map
                              (λ T-invariant ctxt ⇒ T-invariant (f-inv ctxt)
                               | x ⇒ x))
                            ctxt)
                | Ctxt-inv ctxt ⇒ Ctxt-inv (f-inv ctxt)))

definition fold-invariant f-inv ctxt =
           List.fold
            (λ Ctxt-pp ctxt ⇒
                     List.fold
                       (λ T-invariant ctxt ⇒ f-inv ctxt
                        | - ⇒ id)
                       (Ctxt-expr ctxt)
              | Ctxt-inv ctxt ⇒ f-inv ctxt)
            (Ctxt-clause ctxt)

definition fold-invariant' inva =
  rev (fst (fold-invariant (λ(T-inv - (OclProp-ctxt tit inva)) ⇒ λ (accu, n).
                            ( (let tit = case tit of None ⇒ String.of-nat n
                                                   | Some tit ⇒ tit in
                                 (tit, inva))
                              # accu
                            , Suc n))
                       inva
                       ([], 0)))

fun remove-binding where
   remove-binding e = (λ OclTy-collection m ty ⇒ OclTy-collection m (remove-binding ty)
                     | OclTy-pair ty1 ty2 ⇒ OclTy-pair (remove-binding ty1) (remove-binding ty2)
                     | OclTy-binding (-, ty) ⇒ remove-binding ty
                     | OclTy-arrow ty1 ty2 ⇒ OclTy-arrow (remove-binding ty1) (remove-binding ty2)
                     | x ⇒ x) e
```

Class Translation Preliminaries

```
definition const-oid = ⟨oid⟩
definition var-ty-list = ⟨list⟩
definition var-ty-prod = ⟨prod⟩
definition const-oclany = ⟨OclAny⟩

definition single-multip =
  List.list-all (λ (-, Some (Mult-nat n)) ⇒ n ≤ 1
                 | (Mult-nat n, None) ⇒ n ≤ 1
                 | - ⇒ False) o TyMult

fun fold-max-aux where
   fold-max-aux f l l-acc accu = (case l of
     [] ⇒ accu
   | x # xs ⇒ fold-max-aux f xs (x # l-acc) (f x (L.flatten [rev l-acc, xs]) accu))

definition fold-max f l = fold-max-aux f (L.mapi Pair l) []

locale RBTS
begin
definition lookup m k = RBT.lookup m (String.to-list k)
definition insert where insert k = RBT.insert (String.to-list k)
definition map-entry k = RBT.map-entry (String.to-list k)
definition modify-def v k = RBT.modify-def v (String.to-list k)
definition keys m = L.map (λs. ≪s≫) (RBT.keys m)
definition lookup2 m = (λ(k1, k2). RBT.lookup2 m (String.to-list k1, String.to-list k2))
definition insert2 = (λ(k1, k2). RBT.insert2 (String.to-list k1, String.to-list k2))
definition fold where fold f = RBT.fold (λc. f ≪c≫)
definition entries m = L.map (map-prod (λc. ≪c≫) id) (RBT.entries m)
end
lemmas [code] =

  RBTS.lookup-def
  RBTS.insert-def
  RBTS.map-entry-def
  RBTS.modify-def-def
  RBTS.keys-def
  RBTS.lookup2-def
  RBTS.insert2-def
  RBTS.fold-def
  RBTS.entries-def

syntax -rbt-lookup :: - ⇒ - (lookup) translations lookup ⇌ CONST RBTS.lookup
syntax -rbt-insert :: - ⇒ - (insert) translations insert ⇌ CONST RBTS.insert
syntax -rbt-map-entry :: - ⇒ - (map'-entry) translations map-entry ⇌ CONST RBTS.map-entry
syntax -rbt-modify-def :: - ⇒ - (modify'-def) translations modify-def ⇌ CONST RBTS.modify-def
syntax -rbt-keys :: - ⇒ - (keys) translations keys ⇌ CONST RBTS.keys
syntax -rbt-lookup2 :: - ⇒ - (lookup2) translations lookup2 ⇌ CONST RBTS.lookup2
syntax -rbt-insert2 :: - ⇒ - (insert2) translations insert2 ⇌ CONST RBTS.insert2
syntax -rbt-fold :: - ⇒ - (fold) translations fold ⇌ CONST RBTS.fold
syntax -rbt-entries :: - ⇒ - (entries) translations entries ⇌ CONST RBTS.entries

function (sequential) class-unflat-aux where

  class-unflat-aux rbt rbt-inv rbt-cycle r =
  (case lookup rbt-inv r of None ⇒
    (case lookup rbt-cycle r of None (* cycle detection *) ⇒
        map-option
          (OclClass
            r
            (case lookup rbt r of Some l ⇒ l))
          ((λf0 f l.
       let l = List.map f0 l in
         if list-ex (λ None ⇒ True | - ⇒ False) l then
           None
         else
           Some (f (List.map-filter id l))) (class-unflat-aux rbt rbt-inv (insert r () rbt-cycle))
                id
                ([]))
      | - ⇒ None)
  | Some l ⇒
    (case lookup rbt-cycle r of None (* cycle detection *) ⇒
        map-option
```

```
            (OclClass
              r
              (case lookup rbt r of Some l ⇒ l))
            ((λf0 f l.
          let l = List.map f0 l in
            if list-ex (λ None ⇒ True | - ⇒ False) l then
              None
            else
              Some (f (List.map-filter id l))) (class-unflat-aux rbt rbt-inv (insert r () rbt-cycle))
                    id
                    (l))
          | - ⇒ None))
by pat-completeness auto

termination
proof −
 have arith-diff: ⋀a1 a2 (b :: Nat.nat). a1 = a2 ⟹ a1 > b ⟹ a1 − (b + 1) < a2 − b
 by arith

 have arith-less: ⋀(a:: Nat.nat) b c. b ≥ max (a + 1) c ⟹ a < b
 by arith

 have rbt-length: ⋀rbt-cycle r v. RBT.lookup rbt-cycle r = None ⟹
    length (RBT.keys (RBT.insert r v rbt-cycle)) = length (RBT.keys rbt-cycle) + 1
 apply(subst (1 2) distinct-card[symmetric], (rule distinct-keys)+)
 apply(simp only: lookup-keys[symmetric], simp)
 by (metis card-insert-if domIff finite-dom-lookup)

 have rbt-fold-union'': ⋀ab a x k. dom (λb. if b = ab then Some a else k b) = {ab} ∪ dom k
 by(auto)

 have rbt-fold-union': ⋀l rbt-inv a.
       dom (RBT.lookup (List.fold (λ(k, -). RBT.insert k a) l rbt-inv)) =
       dom (map-of l) ∪ dom (RBT.lookup rbt-inv)
 apply(rule-tac P = λrbt-inv . dom (RBT.lookup (List.fold (λ(k, -). RBT.insert k a) l rbt-inv)) =
       dom (map-of l) ∪ dom (RBT.lookup rbt-inv) in allE, simp-all)
 apply(induct-tac l, simp, rule allI)
 apply(case-tac aa, simp)
 apply(simp add: rbt-fold-union'')
done

 have rbt-fold-union: ⋀rbt-cycle rbt-inv a.
   dom (RBT.lookup (RBT.fold (λk -. RBT.insert k a) rbt-cycle rbt-inv)) =
   dom (RBT.lookup rbt-cycle) ∪ dom (RBT.lookup rbt-inv)
 apply(simp add: fold-fold)
 apply(subst (2) map-of-entries[symmetric])
 apply(rule rbt-fold-union')
done

 have rbt-fold-eq: ⋀rbt-cycle rbt-inv a b.
   dom (RBT.lookup (RBT.fold (λk -. RBT.insert k a) rbt-cycle rbt-inv)) =
   dom (RBT.lookup (RBT.fold (λk -. RBT.insert k b) rbt-inv rbt-cycle))
 by(simp add: rbt-fold-union Un-commute)

 let ?len = λx. length (RBT.keys x)
 let ?len-merge = λrbt-cycle rbt-inv. ?len (RBT.fold (λk -. RBT.insert k []) rbt-cycle rbt-inv)

 have rbt-fold-large: ⋀rbt-cycle rbt-inv. ?len-merge rbt-cycle rbt-inv ≥ max (?len rbt-cycle) (?len rbt-inv)
 apply(subst (1 2 3) distinct-card[symmetric], (rule distinct-keys)+)
 apply(simp only: lookup-keys[symmetric], simp)
 apply(subst (1 2) card-mono, simp-all)
 apply(simp add: rbt-fold-union)+
done

 have rbt-fold-eq: ⋀rbt-cycle rbt-inv r a.
    RBT.lookup rbt-inv r = Some a ⟹
    ?len-merge (RBT.insert r () rbt-cycle) rbt-inv = ?len-merge rbt-cycle rbt-inv
 apply(subst (1 2) distinct-card[symmetric], (rule distinct-keys)+)
 apply(simp only: lookup-keys[symmetric])
 apply(simp add: rbt-fold-union)
 by (metis Un-insert-right insert-dom)

 show ?thesis
```

```
  apply(relation measure (λ(-, rbt-inv, rbt-cycle, -).
                    ?len-merge rbt-cycle rbt-inv − ?len rbt-cycle)
      , simp+)
  unfolding RBTS.lookup-def RBTS.insert-def
  apply(subst rbt-length, simp)
  apply(rule arith-diff)
  apply(rule rbt-fold-eq, simp)
  apply(rule arith-less)
  apply(subst rbt-length[symmetric], simp)
  apply(rule rbt-fold-large)
 done
qed
definition ty-obj-to-string = (λOclTyObj (OclTyCore-pre s) - ⇒ s)
definition cl-name-to-string = ty-obj-to-string o ClassRaw-name

definition normalize0 f l =
  rev (snd (List.fold (λx (rbt, l).
                      let x0 = f x in
                      case RBT.lookup rbt x0 of
                        None ⇒ (RBT.insert x0 () rbt, x # l)
                      | Some - ⇒ (rbt, l))
                    l
                    (RBT.empty, []))))

definition class-unflat = (λ (l-class, l-ass).
  let l =
    let const-oclany' = OclTyCore-pre const-oclany
      ; rbt = (∗ fold classes:
                set ⟨OclAny⟩ as default inherited class (for all classes linking to zero inherited classes) ∗)
            insert
              const-oclany
              (ocl-class-raw.make (OclTyObj const-oclany' []) [] [] False)
              (List.fold
                (λ cflat ⇒
                insert (cl-name-to-string cflat) (cflat (| ClassRaw-name := case ClassRaw-name cflat of OclTyObj n [] ⇒ OclTyObj
n [[const-oclany']] | x ⇒ x |)))
                l-class
                RBT.empty) in
    (∗ fold associations:
       add remaining 'object' attributes ∗)
    L.map snd (entries (List.fold (λ (ass-oid, ass) ⇒
      case let (l-none, l-some) = List.partition (λ(-, m). TyRole m = None) (OclAss-relation' ass ) in
          L.flatten [l-none, normalize0 (λ(-, m). case TyRole m of Some s ⇒ String.to-list s) l-some] of
        [] ⇒ id
      | [-] ⇒ id
      | l-rel ⇒
        fold-max
          (let n-rel = natural-of-nat (List.length l-rel) in
           (λ (cpt-to, (name-to, category-to)).
             case TyRole category-to of
               Some role-to ⇒
                 List.fold (λ (cpt-from, (name-from, mult-from)).
                   let name-from = ty-obj-to-string name-from in
                   map-entry name-from (λcflat. cflat (| ClassRaw-own := (role-to,
                     OclTy-class (ocl-ty-class-ext const-oid ass-oid n-rel
                       (ocl-ty-class-node-ext cpt-from mult-from name-from ())
                       (ocl-ty-class-node-ext cpt-to category-to (ty-obj-to-string name-to) ())
                       ())) # ClassRaw-own cflat |)))
                 | - ⇒ λ-. id))
          l-rel) (L.mapi Pair l-ass) rbt)) in
  class-unflat-aux
    (List.fold (λ cflat. insert (cl-name-to-string cflat)
                          (normalize0 (String.to-list o fst) (L.map (map-prod id remove-binding) (ClassRaw-own cflat))))
               l
               RBT.empty)
    (List.fold
      (λ cflat.
        case ClassRaw-name cflat of
          OclTyObj n [] ⇒ id
        | OclTyObj n l ⇒ case rev ([n] # l) of x0 # xs ⇒ λrbt.
            snd (List.fold
                  (λ x (x0, rbt).
                    (x, List.fold (λ OclTyCore-pre k ⇒ modify-def [] k (λl. L.flatten [L.map (λOclTyCore-pre s ⇒ s) x, l]))
```

```
                                x0
                                rbt))
                    xs
                    (x0, rbt)))
      l
      RBT.empty)
    RBT.empty
    const-oclany)

definition class-unflat' x =
 (case class-unflat x of None ⇒ OclClass const-oclany [] []
                       | Some tree ⇒ tree)

fun nb-class where
   nb-class e = (λ OclClass - - l ⇒ Suc (List.fold (op + o nb-class) l 0)) e

definition apply-optim-ass-arity ty-obj v =
  (if TyObj-ass-arity ty-obj ≤ 2 then None
   else Some v)

definition is-higher-order = (λ OclTy-collection - - ⇒ True | OclTy-pair - - ⇒ True | - ⇒ False)

definition parse-ty-raw = (λ OclTy-raw s ⇒ if s = ‹int› then OclTy-base-integer else OclTy-raw s
                           | x ⇒ x)

definition is-sequence = list-ex (λ Sequence ⇒ True | - ⇒ False) o TyCollect

fun str-of-ty where str-of-ty e =
 (λ OclTy-base-void ⇒ ‹Void›
  | OclTy-base-boolean ⇒ ‹Boolean›
  | OclTy-base-integer ⇒ ‹Integer›
  | OclTy-base-unlimitednatural ⇒ ‹UnlimitedNatural›
  | OclTy-base-real ⇒ ‹Real›
  | OclTy-base-string ⇒ ‹String›
  | OclTy-object (OclTyObj (OclTyCore-pre s) -) ⇒ s
 (*| OclTy-object (OclTyObj (OclTyCore ty-obj) -)*)
  | OclTy-collection t ocl-ty ⇒ (if is-sequence t then
                                  S.flatten [‹Sequence(›, str-of-ty ocl-ty,‹)›]
                                else
                                  S.flatten [‹Set(›, str-of-ty ocl-ty,‹)›])
  | OclTy-pair ocl-ty1 ocl-ty2 ⇒ S.flatten [‹Pair(›, str-of-ty ocl-ty1, ‹,›, str-of-ty ocl-ty2,‹)›]
  | OclTy-binding (-, ocl-ty) ⇒ str-of-ty ocl-ty
  | OclTy-class-syn s ⇒ s
  | OclTy-enum s ⇒ s
  | OclTy-raw s ⇒ S.flatten [‹´›, s, ‹´›]) e

definition ty-void = str-of-ty OclTy-base-void
definition ty-boolean = str-of-ty OclTy-base-boolean
definition ty-integer = str-of-ty OclTy-base-integer
definition ty-unlimitednatural = str-of-ty OclTy-base-unlimitednatural
definition ty-real = str-of-ty OclTy-base-real
definition ty-string = str-of-ty OclTy-base-string

definition pref-ty-enum s = ‹ty-enum› @@ String.isub s
definition pref-ty-syn s = ‹ty-syn› @@ String.isub s
definition pref-constr-enum s = ‹constr› @@ String.isub s

fun str-hol-of-ty-all where str-hol-of-ty-all f b e =
 (λ OclTy-base-void ⇒ b ‹unit›
  | OclTy-base-boolean ⇒ b ‹bool›
  | OclTy-base-integer ⇒ b ‹int›
  | OclTy-base-unlimitednatural ⇒ b ‹nat›
  | OclTy-base-real ⇒ b ‹real›
  | OclTy-base-string ⇒ b ‹string›
  | OclTy-object (OclTyObj (OclTyCore-pre s) -) ⇒ b const-oid
  | OclTy-object (OclTyObj (OclTyCore ty-obj) -) ⇒ f (b var-ty-list) [b (TyObj-name ty-obj)]
  | OclTy-collection - ty ⇒ f (b var-ty-list) [str-hol-of-ty-all f b ty]
  | OclTy-pair ty1 ty2 ⇒ f (b var-ty-prod) [str-hol-of-ty-all f b ty1, str-hol-of-ty-all f b ty2]
  | OclTy-binding (-, t) ⇒ str-hol-of-ty-all f b t
  | OclTy-class-syn s ⇒ b (pref-ty-syn s)
  | OclTy-enum s ⇒ b (pref-ty-enum s)
  | OclTy-raw s ⇒ b s) e
```

```
definition print-infra-type-synonym-class-set-name name = ‹Set-› @@ name
definition print-infra-type-synonym-class-sequence-name name = ‹Sequence-› @@ name

fun get-class-hierarchy-strict-aux where
  get-class-hierarchy-strict-aux dataty l-res =
  (List.fold
    (λ OclClass name l-attr dataty ⇒ λ l-res.
      get-class-hierarchy-strict-aux dataty (OclClass name l-attr dataty # l-res))
    dataty
    l-res)
definition get-class-hierarchy-strict d = get-class-hierarchy-strict-aux d []

fun get-class-hierarchy'-aux where
  get-class-hierarchy'-aux l-res (OclClass name l-attr dataty) =
  (let l-res = OclClass name l-attr dataty # l-res in
   case dataty of [] ⇒ rev l-res
                | dataty ⇒ List.fold (λx acc. get-class-hierarchy'-aux acc x) dataty l-res)
definition get-class-hierarchy' = get-class-hierarchy'-aux []

definition get-class-hierarchy e = L.map (λ OclClass n l - ⇒ (n, l)) (get-class-hierarchy' e)
definition get-class-hierarchy-sub = (λ None ⇒ []
                                    | Some next-dataty ⇒ get-class-hierarchy next-dataty)
definition get-class-hierarchy-sub' = (λ None ⇒ []
                                     | Some next-dataty ⇒ get-class-hierarchy' next-dataty)

datatype position = EQ | LT | GT | UN'

fun fold-less-gen where fold-less-gen f-gen f-jump f l = (case l of
   x # xs ⇒ λacc. fold-less-gen f-gen f-jump f xs (f-gen (f x) xs (f-jump acc))
 | [] ⇒ id)

definition fold-less2 = fold-less-gen List.fold
```

E.2 Translation of AST

```
definition var-in-pre-state = ‹in-pre-state›
definition var-in-post-state = ‹in-post-state›
definition var-at-when-hol-post = ‹›
definition var-at-when-hol-pre = ‹at-pre›
definition var-at-when-ocl-post = ‹›
definition var-at-when-ocl-pre = ‹@pre›

datatype 'a tmp-sub = Tsub 'a
record 'a inheritance =
  Inh :: 'a
  Inh-sib :: ('a × 'a list (* flat version of the 1st component *)) list
  Inh-sib-unflat :: 'a list
datatype 'a tmp-inh = Tinh 'a
datatype 'a tmp-univ = Tuniv 'a
definition of-inh = (λTinh l ⇒ l)
definition of-linh = L.map Inh
definition of-linh-sib l = L.flatten (L.map snd (L.flatten (L.map Inh-sib l)))
definition of-sub = (λTsub l ⇒ l)
definition of-univ = (λTuniv l ⇒ l)
definition map-inh f = (λTinh l ⇒ Tinh (f l))
definition map-linh f cl = (| Inh = f (Inh cl)
                            , Inh-sib = L.map (map-prod f (L.map f)) (Inh-sib cl)
                            , Inh-sib-unflat = L.map f (Inh-sib-unflat cl) |)

fun fold-class-gen-aux where
   fold-class-gen-aux l-inh f accu (OclClass name l-attr dataty) =
 (let accu = f (λs. s @@ String.isub name)
               name
               l-attr
               (Tinh l-inh)
               (Tsub (get-class-hierarchy-strict dataty)) (* order: bfs or dfs (modulo reversing) *)
               dataty
               accu in
  case dataty of [] ⇒ accu
               | - ⇒
    fst (List.fold
      (λ node (accu, l-inh-l, l-inh-r).
```

```
      ( fold-class-gen-aux
          ( (| Inh = OclClass name l-attr dataty
            , Inh-sib = L.flatten (L.map (L.map (λl. (l, get-class-hierarchy' l))) [l-inh-l, tl l-inh-r])
            , Inh-sib-unflat = L.flatten [l-inh-l, tl l-inh-r] |)
          # l-inh)
          f accu node
      , hd l-inh-r # l-inh-l
      , tl l-inh-r))
    dataty
    (accu, [], dataty)))

definition fold-class-gen f accu expr =
 (let (l-res, accu) =
    fold-class-gen-aux
      []
      (λ isub-name name l-attr l-inh l-subtree next-dataty (l-res, accu).
        let (r, accu) = f isub-name name l-attr l-inh l-subtree next-dataty accu in
        (r # l-res, accu))
      ([], accu)
      expr in
  (L.flatten l-res, accu))

definition map-class-gen f = fst o fold-class-gen
  (λ isub-name name l-attr l-inh l-subtree last-d. λ () ⇒
    (f isub-name name l-attr l-inh l-subtree last-d, ())) ()

definition add-hierarchy f x = (λisub-name name - - - -. f isub-name name (Tuniv (L.map fst (get-class-hierarchy x))))
definition add-hierarchy' f x = (λisub-name name - - - -. f isub-name name (Tuniv (get-class-hierarchy x)))
definition add-hierarchy'' f x = (λisub-name name l-attr - - -. f isub-name name (Tuniv (get-class-hierarchy x)) l-attr)
definition add-hierarchy''' f x = (λisub-name name l-attr l-inh - next-dataty. f isub-name name (Tuniv (get-class-hierarchy x))
l-attr (map-inh (L.map (λ OclClass - l - ⇒ l) o of-linh) l-inh) next-dataty)
definition add-hierarchy'''' f x = (λisub-name name l-attr l-inh l-subtree -. f isub-name name (Tuniv (get-class-hierarchy x))
l-attr (map-inh (L.map (λ OclClass - l - ⇒ l) o of-linh) l-inh) l-subtree)
definition add-hierarchy''''' f = (λisub-name name l-attr l-inh l-subtree. f isub-name name l-attr (of-inh l-inh) (of-sub l-subtree))
definition map-class f = map-class-gen (λisub-name name l-attr l-inh l-subtree next-dataty. [f isub-name name l-attr l-inh (Tsub
(L.map (λ OclClass n - - ⇒ n) (of-sub l-subtree))) next-dataty])
definition map-class' f = map-class-gen (λisub-name name l-attr l-inh l-subtree next-dataty. [f isub-name name l-attr l-inh
l-subtree next-dataty])
definition fold-class f = fold-class-gen (λisub-name name l-attr l-inh l-subtree next-dataty accu. let (x, accu) = f isub-name
name l-attr (map-inh of-linh l-inh) (Tsub (L.map (λ OclClass n - - ⇒ n) (of-sub l-subtree))) next-dataty accu in ([x], accu))
definition map-class-gen-h f x = map-class-gen (add-hierarchy f x) x
definition map-class-gen-h' f x = map-class-gen (add-hierarchy' f x) x
definition map-class-gen-h'' f x = map-class-gen (add-hierarchy'' f x) x
definition map-class-gen-h''' f x = map-class-gen (add-hierarchy''' f x) x
definition map-class-gen-h'''' f x = map-class-gen (add-hierarchy'''' (λisub-name name l-inherited l-attr l-inh l-subtree. f
isub-name name l-inherited l-attr l-inh (Tsub (L.map (λ OclClass n - - ⇒ n) (of-sub l-subtree)))) x) x
definition map-class-gen-h''''' f x = map-class-gen (add-hierarchy''''' f) x
definition map-class-h f x = map-class (add-hierarchy f x) x
definition map-class-h' f x = map-class (add-hierarchy' f x) x
definition map-class-h'' f x = map-class (add-hierarchy'' f x) x
definition map-class-h''' f x = map-class (add-hierarchy''' f x) x
definition map-class-h'''' f x = map-class (add-hierarchy'''' f x) x
definition map-class-h''''' f x = map-class' (add-hierarchy''''' f) x
definition map-class-arg-only f = map-class-gen (λ isub-name name l-attr - - -. case l-attr of [] ⇒ [] | l ⇒ f isub-name name l)
definition map-class-arg-only' f = map-class-gen (λ isub-name name l-attr l-inh l-subtree -.
  case filter (λ OclClass - [] - ⇒ False | - ⇒ True) (of-linh (of-inh l-inh)) of
    [] ⇒ []
  | l ⇒ f isub-name name (l-attr, Tinh l, l-subtree))
definition map-class-arg-only0 f1 f2 u = map-class-arg-only f1 u @@@@ map-class-arg-only' f2 u
definition map-class-arg-only-var0 = (λf-expr f-app f-lattr isub-name name l-attr.
  L.flatten (L.flatten (
    L.map (λ(var-in-when-state, dot-at-when, attr-when).
      L.flatten (L.map (λ l-attr. L.map (λ(attr-name, attr-ty).
        f-app
          isub-name
          name
          (var-in-when-state, dot-at-when)
          attr-ty
          (λs. s @@ String.isup attr-name)
          (λs. f-expr s
            [ case case attr-ty of
                    OclTy-object (OclTyObj (OclTyCore ty-obj) -) ⇒
                      apply-optim-ass-arity ty-obj
```

```
                    (let ty-obj = TyObj-from ty-obj in
                    case TyObjN-role-name ty-obj of
                      None => String.of-natural (TyObjN-ass-switch ty-obj)
                    | Some s => s)
                | - ⇒ None of
              None ⇒ mk-dot attr-name attr-when
            | Some s2 ⇒ mk-dot-comment attr-name attr-when s2 ])) l-attr)
    (f-lattr l-attr)))
  [ (var-in-post-state, var-at-when-hol-post, var-at-when-ocl-post)
  , (var-in-pre-state, var-at-when-hol-pre, var-at-when-ocl-pre)]))))
definition map-class-arg-only-var-gen f-expr f1 f2 = map-class-arg-only0 (map-class-arg-only-var0 f-expr f1 (λl. [l]))
(map-class-arg-only-var0 f-expr f2 (λ (-, Tinh l, -) ⇒ L.map (λ OclClass - l - ⇒ l) l))
definition map-class-arg-only-var′-gen f-expr f = map-class-arg-only0 (map-class-arg-only-var0 f-expr f (λl. [l]))
(map-class-arg-only-var0 f-expr f (λ (-, Tinh l, -) ⇒ L.map (λ OclClass - l - ⇒ l) l))
definition map-class-arg-only-var′′-gen f-expr f = map-class-arg-only (map-class-arg-only-var0 f-expr f (λl. [l]))
definition map-class-one f-l f expr =
  (case f-l (fst (fold-class (λisub-name name l-attr l-inh l-inh-sib next-dataty -. ((isub-name, name, l-attr, l-inh, l-inh-sib,
next-dataty), ())) () expr)) of
    (isub-name, name, l-attr, l-inh, l-inh-sib, next-dataty) # - ⇒
    f isub-name name l-attr l-inh l-inh-sib next-dataty)
definition map-class-top = map-class-one rev
definition get-hierarchy-map f f-l x = L.flatten (L.flatten (
  let (l1, l2, l3) = f-l (L.map fst (get-class-hierarchy x)) in
  L.map (λname1. L.map (λname2. L.map (f name1 name2) l3) l2) l1))

definition class-arity = RBT.keys o (λl. List.fold (λx. RBT.insert x ()) l RBT.empty) o
  L.flatten o L.flatten o map-class (λ - - l-attr - - -.
    L.map (λ (-, OclTy-object (OclTyObj (OclTyCore ty-obj) -)) ⇒ [TyObj-ass-arity ty-obj]
            | - ⇒ []) l-attr)

definition map-class-gen-h′-inh f =
  map-class-gen-h′′′′′ (λisub-name name - l-inh l-subtree -.
    let l-mem = λl. List.member (L.map (λ OclClass n - - ⇒ String.to-list n) l) in
    f isub-name
      name
      (λn. let n = String.to-list n in
           if (∗ TODO use ≜ ∗) n = String.to-list name then EQ else
           if l-mem (of-linh l-inh) n then GT else
           if l-mem l-subtree n then LT else
           UN′))

definition m-class-gen2 base-attr f print =
 (let m-base-attr = λ OclClass n l b ⇒ OclClass n (base-attr l) b
    ; f-base-attr = L.map m-base-attr in
  map-class-gen-h′′′′′ (λisub-name name nl-attr l-inh l-subtree next-dataty.
    f name
      l-inh
      l-subtree
      (L.flatten (L.flatten (L.map (
        let print-astype =
              print
                (L.map (map-linh m-base-attr) l-inh)
                (f-base-attr l-subtree)
                next-dataty
          ; nl-attr = base-attr nl-attr in
        (λ(l-hierarchy, l).
          L.map
            (print-astype l-hierarchy (isub-name, name, nl-attr) o m-base-attr)
            l))
      [ (EQ, [OclClass name nl-attr next-dataty])
      , (GT, of-linh l-inh)
      , (LT, l-subtree)
      , (UN′, of-linh-sib l-inh) ])))))

definition f-less2 =
  (λf l. rev (fst (fold-less2 (λ(l, -). (l, None)) (λx y (l, acc). (f x y acc # l, Some y)) l ([], None))))
    (λa b -. (a,b))

definition m-class-gen3-GE base-attr f print =
 (let m-base-attr = λ OclClass n l b ⇒ OclClass n (base-attr l) b
    ; f-base-attr = L.map m-base-attr in
  map-class-gen-h′′′′′ (λisub-name name nl-attr l-inh l-subtree next-dataty.
    let print-astype =
```

```
        print
          (L.map (map-linh m-base-attr) l-inh)
          (f-base-attr l-subtree)
          next-dataty in
    L.flatten
     [ f (L.flatten (L.map (λ (l-hierarchy, l).
         L.map (λ OclClass h-name - - ⇒ print-astype name h-name h-name) l)
         [ (GT, of-linh l-inh) ]))
     , f (L.flatten (L.map (λ (l-hierarchy, l).
         L.map (λ (h-name, hh-name). print-astype name h-name hh-name) (f-less2 (L.map (λ OclClass n - - ⇒ n) l)))
         [ (GT, of-linh l-inh) ]))
     , f (L.flatten (L.map (λ (l-hierarchy, l).
         L.flatten (L.map (λ OclClass h-name - - ⇒
           L.map (λ OclClass sib-name - - ⇒ print-astype name sib-name h-name) (of-linh-sib l-inh)) l))
         [ (GT, of-linh l-inh) ])) ]))

definition m-class-gen3 base-attr f print =
 (let m-base-attr = λ OclClass n l b ⇒ OclClass n (base-attr l) b
    ; f-base-attr = L.map m-base-attr in
  map-class-gen-h''''' (λisub-name name nl-attr l-inh l-subtree next-dataty.
    let print-astype =
        print
          (L.map (map-linh m-base-attr) l-inh)
          (f-base-attr l-subtree)
          next-dataty in
    f (L.flatten (
        let l-tree = L.map (λ(cmp,l). (cmp, f-base-attr l))
          [ (EQ, [OclClass name nl-attr next-dataty])
          , (GT, of-linh l-inh)
          , (LT, l-subtree)
          , (UN', of-linh-sib l-inh) ] in
        (λf. L.flatten (L.map (λ (l-hierarchy, l). L.map (f l-hierarchy) l) l-tree))
        (λ l-hierarchy1. λ OclClass h-name hl-attr hb ⇒
        (λf. L.flatten (L.map (λ (l-hierarchy, l). L.map (f l-hierarchy) l) l-tree))
        (λ l-hierarchy2. λ OclClass hh-name hhl-attr hhb ⇒
          print-astype
            name
            h-name
            hh-name))))))

definition m-class-default = (λ- - -. id)
definition m-class base-attr f print = m-class-gen2 base-attr f (λ- - -. print)
definition m-class3-GE base-attr f print = m-class-gen3-GE base-attr f (λ- - -. print)
definition m-class' base-attr print =
  m-class base-attr m-class-default (λ l-hierarchy x0 x1. [ print l-hierarchy x0 x1 ])

definition map-class-nupl2'-inh f = List.map-filter id o
 (m-class' id (λcompare (-, name, -). λ OclClass h-name - - ⇒
    if compare = GT then Some (f name h-name) else None))

definition map-class-nupl2'-inh-large f = List.map-filter id o
 (m-class' id (λcompare (-, name, -). λ OclClass h-name - - ⇒
    if compare = GT
     | compare = UN' then Some (f name h-name) else None))

definition map-class-nupl2''-inh f = List.map-filter id o
 (m-class-gen2 id m-class-default (λ l-inh - - compare (-, name, -). λ OclClass h-name - h-subtree ⇒
    [ if compare = GT then
        Some (f name h-name (L.map (λx. (x, List.member (of-linh l-inh) x)) h-subtree))
      else
        None]))

definition map-class-nupl2l'-inh-gen f = List.map-filter id o
 (m-class-gen2 id m-class-default (λ l-inh l-subtree - compare (-, name, -). λ OclClass h-name - - ⇒
    [ if compare = GT then
        Some (f l-subtree name (fst (List.fold (λx. λ (l, True, prev-x) ⇒ (l, True, prev-x)
                                    | (l, False, prev-x) ⇒
                                      case Inh x of OclClass n - next-d ⇒
                                      ( (x, L.map (λ OclClass n l next-d ⇒
                                                     (OclClass n l next-d, n = prev-x))
                                                  next-d)
                                        # l
                                      , n = h-name
```

```
                                    , n))
                                l-inh
                                ([], False, name))))
    else
      None]))

definition map-class-nupl2l'-inh f = map-class-nupl2l'-inh-gen (λ- x l. f x l)

definition map-class-nupl3'-LE'-inh f = L.flatten o map-class-nupl2l'-inh-gen (λl-subtree x l.
  L.map
    (λname-bot. f name-bot x l)
    (x # L.map (λ OclClass n - - ⇒ n) l-subtree))

definition map-class-nupl3'-GE-inh = m-class3-GE id id

definition map-class-inh l-inherited = L.map (λ OclClass - l - ⇒ l) (of-inh (map-inh of-linh l-inherited))

definition find-inh name class =
 (case fold-class
    (λ- name0 - l-inh - - accu.
      Pair () (if accu = None & name ≙ name0 then
                 Some (L.map (λOclClass n - - ⇒ n) (of-inh l-inh))
               else
                 accu))
    None
    class
  of (-, Some l) ⇒ l)

end
```

E.3 OCL Meta-Model aka. AST definition of OCL (II)

```
theory  Meta-UML-extended
imports ../../compiler-generic/Init
begin
```

Type Definition

```
datatype internal-oid = Oid nat
datatype internal-oids = Oids nat
                         nat
                         nat

datatype ocl-def-base = OclDefInteger string
                      | OclDefReal string (* integer digit (left) *) × string (* integer digit (right) *)
                      | OclDefString string

datatype ocl-data-shallow = ShallB-term ocl-def-base
                          | ShallB-str string
                          | ShallB-self internal-oid
                          | ShallB-list ocl-data-shallow list

datatype 'a ocl-list-attr = OclAttrNoCast 'a
                          | OclAttrCast
                            string
                            'a ocl-list-attr
                            'a

record ocl-instance-single = Inst-name :: string option
                             Inst-ty :: string option
                             Inst-attr-with :: string (* name *) option
                             Inst-attr :: ((  (string (* pre state *) × string (* post state *)) option
                                              (* state used when ocl-data-shallow is an object variable (for retrieving its oid) *)
                                            × string (*name*)
                                            × ocl-data-shallow) list) (* inh and own *)
                                          ocl-list-attr

datatype ocl-instance = OclInstance ocl-instance-single list

datatype ocl-def-base-l = OclDefBase ocl-def-base list

datatype 'a ocl-def-state-core = OclDefCoreAdd ocl-instance-single
```

```
                        | OclDefCoreBinding 'a

datatype ocl-def-state = OclDefSt  string
                           string (* name *) ocl-def-state-core list

datatype ocl-def-pp-core = OclDefPPCoreAdd string (* name *) ocl-def-state-core list
                 | OclDefPPCoreBinding string

datatype ocl-def-transition = OclDefPP
                        string option
                        ocl-def-pp-core
                        ocl-def-pp-core option

datatype ocl-class-tree = OclClassTree nat
                              nat
```

Object ID Management

```
definition oidInit = (λ Oid n ⇒ Oids n n n)

definition oidSucAssoc = (λ Oids n1 n2 n3 ⇒ Oids n1 (Succ n2) (Succ n3))
definition oidSucInh = (λ Oids n1 n2 n3 ⇒ Oids n1 n2 (Succ n3))
definition oidGetAssoc = (λ Oids - n - ⇒ Oid n)
definition oidGetInh = (λ Oids - - n ⇒ Oid n)

definition oidReinitAll = (λOids n1 - - ⇒ Oids n1 n1 n1)
definition oidReinitInh = (λOids n1 n2 - ⇒ Oids n1 n2 n2)
```

Operations of Fold, Map, ..., on the Meta-Model

```
definition ocl-instance-single-empty =
  (| Inst-name = None, Inst-ty = None, Inst-attr-with = None, Inst-attr = OclAttrNoCast [] |)

fun map-data-shallow-self where
   map-data-shallow-self f e = (λ ShallB-self s ⇒ f s
                             | ShallB-list l ⇒ ShallB-list (List.map (map-data-shallow-self f) l)
                             | x ⇒ x) e

fun map-list-attr where
   map-list-attr f e =
     (λ OclAttrNoCast x ⇒ OclAttrNoCast (f x)
      | OclAttrCast c-from l-attr x ⇒ OclAttrCast c-from (map-list-attr f l-attr) (f x)) e

definition map-instance-single f ocli = ocli (| Inst-attr := map-list-attr (L.map f) (Inst-attr ocli) |)

fun fold-list-attr where
   fold-list-attr cast-from f l-attr accu = (case l-attr of
        OclAttrNoCast x ⇒ f cast-from x accu
     | OclAttrCast c-from l-attr x ⇒ fold-list-attr (Some c-from) f l-attr (f cast-from x accu))

definition inst-ty0 ocli = (case Inst-ty ocli of Some ty ⇒ Some ty
                                              | None ⇒ (case Inst-attr ocli of OclAttrCast ty - - ⇒ Some ty
                                                                             | - ⇒ None))
definition inst-ty ocli = (case inst-ty0 ocli of Some ty ⇒ ty)

definition fold-instance-single f ocli = fold-list-attr (inst-ty0 ocli) (λ Some x ⇒ f x) (Inst-attr ocli)
definition fold-instance-single' f ocli = fold-list-attr (Inst-ty ocli) f (Inst-attr ocli)

definition str-of-def-base = (λ OclDefInteger - ⇒ ‹Integer›
                        | OclDefReal - ⇒ ‹Real›
                        | OclDefString - ⇒ ‹String›)

fun' str-of-data-shallow where
   ‹str-of-data-shallow e = (λ ShallB-term b ⇒ str-of-def-base b
                        | ShallB-str s ⇒ ‹› @@ s @@ ‹›
                        | ShallB-self - ⇒ ‹(*object-oid*)›
                        | ShallB-list l ⇒ ‹[ › @@ String-concatWith ‹, › (List.map str-of-data-shallow l) @@ ‹ ]›) e›

definition map-inst-single-self f =
  map-instance-single
    (map-prod id
      (map-prod id
```

(*map-data-shallow-self f*)))

end

E.4 Regrouping Together All Existing Meta-Models

theory *Meta-META*
imports *Meta-UML*
Meta-UML-extended
../../compiler-generic/meta-isabelle/Meta-Isabelle
begin

A Basic Meta-Model

The following basic Meta-Model is an empty Meta-Model.

Most of the Meta-Model we have defined (in particular those defined in UML) can be used in exceptional situations for requiring an eager or lazy interactive evaluation of already encountered Meta-Models. This is also the case for this basic Meta-Model.

datatype *ocl-flush-all* = *OclFlushAll*

The META Meta-Model (I)

datatype *floor* = *Floor1* | *Floor2* | *Floor3*

Meta-Models can be seen as arranged in a semantic tower with several floors. By default, *Floor1* corresponds to the first level we are situating by default, then a subsequent meta-evaluation would jump to a deeper floor, to *Floor2*, then *Floor3*...

It is not mandatory to jump to a floor superior than the one we currently are. The important point is to be sure that all jumps will ultimately terminate.

Most of the following constructors are preceded by an additional *floor* field, which explicitly indicates the intended associated semantic to consider during the meta-embedding to Isabelle. In case no *floor* is precised, we fix it to be *Floor1* by default.

datatype *all-meta-embedding* =

META-enum ocl-enum
| *META-class-raw floor ocl-class-raw*
| *META-association ocl-association*
| *META-ass-class floor ocl-ass-class*
| *META-ctxt floor ocl-ctxt*

| *META-class-synonym ocl-class-synonym*
| *META-instance ocl-instance*
| *META-def-base-l ocl-def-base-l*
| *META-def-state floor ocl-def-state*
| *META-def-transition floor ocl-def-transition*
| *META-class-tree ocl-class-tree*
| *META-flush-all ocl-flush-all*

Main Compiling Environment

The environment constitutes the main data-structure carried by all monadic translations.

datatype *generation-semantics-ocl* = *Gen-only-design* | *Gen-only-analysis* | *Gen-default*
datatype *generation-lemma-mode* = *Gen-sorry* | *Gen-no-dirty*

record *compiler-env-config* = *D-output-disable-thy* :: *bool*
D-output-header-thy :: (*string* (∗ *theory* ∗)
× *string list* (∗ *imports* ∗)
× *string* (∗ *import optional* (*compiler bootstrap*) ∗)) *option*
D-ocl-oid-start :: *internal-oids*
D-output-position :: *nat* × *nat*

```
D-ocl-semantics :: generation-semantics-ocl
D-input-class :: ocl-class option

D-input-meta :: all-meta-embedding list
D-input-instance :: (string_base (* name (as key for rbt) *)
                    × ocl-instance-single
                    × internal-oids) list

D-input-state :: (string_base (* name (as key for rbt) *)
                 × (internal-oids
                 × (string (* name *)
                   × ocl-instance-single (* alias *))
                   ocl-def-state-core) list) list

D-output-header-force :: bool
D-output-auto-bootstrap :: bool
D-ocl-accessor :: string_base (* name of the constant added *) list (* pre *)
                × string_base (* name of the constant added *) list (* post *)
D-ocl-HO-type :: (string_base (* raw HOL name (as key for rbt) *)) list
D-output-sorry-dirty :: generation-lemma-mode option × bool (* dirty *)
```

Operations of Fold, Map, ..., on the Meta-Model

```
definition ignore-meta-header = (λ META-ctxt Floor1 - ⇒ True
                                | META-def-state Floor1 - ⇒ True
                                | META-def-transition Floor1 - ⇒ True
                                | - ⇒ False)
```

As remark in *ignore-meta-header*, *META-class-raw* and *META-ass-class* do not occur, even if the associated meta-commands will be put at the beginning when generating files during the reordering step. This is because some values for which *ignore-meta-header* returns *False* can exist just before meta-commands associated to *META-class-raw* or *META-ass-class*.

```
definition map2-ctxt-term f =
 (let f-prop = λ OclProp-ctxt n prop ⇒ OclProp-ctxt n (f prop)
   ; f-inva = λ T-inv b prop ⇒ T-inv b (f-prop prop) in
  λ META-ctxt Floor2 c ⇒
    META-ctxt Floor2
      (Ctxt-clause-update
        (L.map (λ Ctxt-pp pp ⇒ Ctxt-pp (Ctxt-expr-update (L.map (λ T-pp pref prop ⇒ T-pp pref (f-prop prop)
                                                              | T-invariant inva ⇒ T-invariant (f-inva inva))) pp)
                | Ctxt-inv l-inv ⇒ Ctxt-inv (f-inva l-inv))) c)
 | x ⇒ x)

definition compiler-env-config-more-map f ocl =
          compiler-env-config.extend  (compiler-env-config.truncate ocl) (f (compiler-env-config.more ocl))

definition compiler-env-config-empty output-disable-thy output-header-thy oid-start design-analysis sorry-dirty =
  compiler-env-config.make
    output-disable-thy
    output-header-thy
    oid-start
    (0, 0)
    design-analysis
    None [] [] [] False False ([], []) []
    sorry-dirty

definition compiler-env-config-reset-no-env env =
  compiler-env-config-empty
    (D-output-disable-thy env)
    (D-output-header-thy env)
    (oidReinitAll (D-ocl-oid-start env))
    (D-ocl-semantics env)
    (D-output-sorry-dirty env)
    (| D-input-meta := D-input-meta env |)
```

The META Meta-Model (II)

Type Definition

For bootstrapping the environment through the jumps to another semantic floor, we additionally consider the environment as a Meta-Model.

datatype *boot-generation-syntax = Boot-generation-syntax generation-semantics-ocl*
datatype *boot-setup-env = Boot-setup-env compiler-env-config*

datatype *all-meta =*
META-semi--theories semi--theories

| *META-boot-generation-syntax boot-generation-syntax*
| *META-boot-setup-env boot-setup-env*
| *META-all-meta-embedding all-meta-embedding*

As remark, the Isabelle Meta-Model represented by *semi--theories* can be merged with the previous META Meta-Model *all-meta-embedding*. However a corresponding parser and printer would then be required.

Extending the Meta-Model

locale *O*
begin
definition *i x = META-semi--theories o Theories-one o x*
definition *datatype = i Theory-datatype*
definition *type-synonym = i Theory-type-synonym*
definition *type-notation = i Theory-type-notation*
definition *instantiation = i Theory-instantiation*
definition *overloading = i Theory-overloading*
definition *consts = i Theory-consts*
definition *definition = i Theory-definition*
definition *lemmas = i Theory-lemmas*
definition *lemma = i Theory-lemma*
definition *axiomatization = i Theory-axiomatization*
definition *section = i Theory-section*
definition *text = i Theory-text*
definition *text-raw = i Theory-text-raw*
definition *ML = i Theory-ML*
definition *setup = i Theory-setup*
definition *thm = i Theory-thm*
definition *interpretation = i Theory-interpretation*
end

lemmas [*code*] =

O.i-def
O.datatype-def
O.type-synonym-def
O.type-notation-def
O.instantiation-def
O.overloading-def
O.consts-def
O.definition-def
O.lemmas-def
O.lemma-def
O.axiomatization-def
O.section-def
O.text-def
O.text-raw-def
O.ML-def
O.setup-def
O.thm-def
O.interpretation-def

locale *O′*
begin
definition *datatype = Theory-datatype*
definition *type-synonym = Theory-type-synonym*
definition *type-notation = Theory-type-notation*
definition *instantiation = Theory-instantiation*
definition *overloading = Theory-overloading*
definition *consts = Theory-consts*
definition *definition = Theory-definition*
definition *lemmas = Theory-lemmas*
definition *lemma = Theory-lemma*

definition *axiomatization* = *Theory-axiomatization*
definition *section* = *Theory-section*
definition *text* = *Theory-text*
definition *ML* = *Theory-ML*
definition *setup* = *Theory-setup*
definition *thm* = *Theory-thm*
definition *interpretation* = *Theory-interpretation*
end

lemmas [*code*] =

O′.datatype-def
O′.type-synonym-def
O′.type-notation-def
O′.instantiation-def
O′.overloading-def
O′.consts-def
O′.definition-def
O′.lemmas-def
O′.lemma-def
O′.axiomatization-def
O′.section-def
O′.text-def
O′.ML-def
O′.setup-def
O′.thm-def
O′.interpretation-def

Operations of Fold, Map, ..., on the Meta-Model

definition *map-semi--theory f* = (λ *META-semi--theories* (*Theories-one x*) ⇒ *META-semi--theories* (*Theories-one* (*f x*))
| *META-semi--theories* (*Theories-locale data l*) ⇒ *META-semi--theories* (*Theories-locale data* (*L.map* (*L.map f*) *l*))
| *x* ⇒ *x*)

end

APPENDIX F

HOL-OCL 2.0: Translating Meta-Models

For space reasons, we will skip the presentation of all packaging functions and only present their final assembling. (As detailed in Section 5.3, a packaging function is a mapping between two meta-models.)

F.1 General Environment for the Translation: Conclusion

```
theory Core
imports core/Floor1-enum
        core/Floor1-infra
        core/Floor1-astype
        core/Floor1-istypeof
        core/Floor1-iskindof
        core/Floor1-allinst
        core/Floor1-access
        core/Floor1-examp
        core/Floor2-examp
        core/Floor1-ctxt
        core/Floor2-ctxt
begin
```

Preliminaries

```
datatype 'a embedding-fun = Embedding-fun-info string 'a
                          | Embedding-fun-simple 'a

datatype ('a, 'b) embedding = Embed-theories ('a ⇒ 'b ⇒ all-meta list × 'b) embedding-fun list
                            | Embed-locale ('a ⇒ 'b ⇒ all-meta list × 'b) embedding-fun list
                                           'a ⇒ 'b ⇒ semi--locale × 'b
                                           ('a ⇒ 'b ⇒ semi--theory list × 'b) list
                                           ('a ⇒ 'b ⇒ all-meta list × 'b) embedding-fun list

type-synonym 'a embedding' = ('a, compiler-env-config) embedding

definition L-fold f =
 (let f-locale = λloc-data l.
    f (Embedding-fun-simple (λa b.
          let (loc-data, b) = loc-data a b
            ; (l, b) = List.fold (λf0. λ(l, b) ⇒ let (x, b) = f0 a b in (x # l, b)) l ([], b) in
          ([META-semi--theories (Theories-locale loc-data (rev l))], b))) in
  λ Embed-theories l ⇒ List.fold f l
  | Embed-locale l-th1 loc-data l-loc l-th2 ⇒ List.fold f l-th2 o f-locale loc-data l-loc o List.fold f l-th1)
```

Preliminaries: Setting Up Aliases Names

```
ML‹
local
fun definition s = (#2 oo Specification.definition-cmd (NONE, ((@{binding }, []), s))) true
fun def-info lhs rhs = definition (lhs ^ =  ^
                                  @{const-name Embedding-fun-info} ^
                                    (‹ ^ rhs ^ ›)  ^
                                  rhs)
fun name-print x = String.implode (case String.explode (Long-Name.base-name x) of
      #p :: #r :: #i :: #n :: #t :: #- :: l => l
```

```
    | - => error 'print' expected)
fun name x = PRINT- ^ name-print x
fun name1 x = floor1-PRINT- ^ name-print x
fun name2 x = floor2-PRINT- ^ name-print x
in
fun embedding-fun-info rhs = def-info (name rhs) rhs
fun embedding-fun-simple rhs = definition (name rhs ^ "  ^
                                @{const-name Embedding-fun-simple} ^ ( ^ rhs ^ ))
fun embedding-fun-info-f1 rhs = def-info (name1 rhs) rhs
fun embedding-fun-simple-f1 rhs = definition (name1 rhs ^ "  ^
                                @{const-name Embedding-fun-simple} ^ ( ^ rhs ^ ))
fun embedding-fun-info-f2 rhs = def-info (name2 rhs) rhs
fun embedding-fun-simple-f2 rhs = definition (name2 rhs ^ "  ^
                                @{const-name Embedding-fun-simple} ^ ( ^ rhs ^ ))
fun emb-info rhs = def-info (Long-Name.base-name rhs ^ "info") rhs
fun emb-simple rhs = definition (Long-Name.base-name rhs ^ "simple" ^ "  ^
                                @{const-name Embedding-fun-simple} ^ ( ^ rhs ^ ))
end
⟩


local-setup ⟨embedding-fun-info @{const-name print-infra-enum-synonym}⟩
local-setup ⟨embedding-fun-info @{const-name print-latex-infra-datatype-class}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-datatype-class}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-datatype-universe}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-type-synonym-class}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-type-synonym-class-higher}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-type-synonym-class-rec}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-enum-syn}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-instantiation-class}⟩
local-setup ⟨embedding-fun-info @{const-name print-infra-instantiation-universe}⟩
local-setup ⟨embedding-fun-info @{const-name print-instantia-def-strictrefeq}⟩
local-setup ⟨embedding-fun-info @{const-name print-instantia-lemmas-strictrefeq}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-consts}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-class}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-from-universe}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-lemmas-id}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-lemma-cp}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-lemmas-cp}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-lemma-strict}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-lemmas-strict}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-defined}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-up-d-cast0}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-up-d-cast}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-d-up-cast}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-lemma-const}⟩
local-setup ⟨embedding-fun-info @{const-name print-astype-lemmas-const}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-consts}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-class}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-from-universe}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-lemmas-id}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-lemma-cp}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-lemmas-cp}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-lemma-strict}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-lemmas-strict}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-defined}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-defined'}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-up-larger}⟩
local-setup ⟨embedding-fun-info @{const-name print-istypeof-up-d-cast}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-consts}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-class}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-from-universe}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-lemmas-id}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-lemma-cp}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-lemmas-cp}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-lemma-strict}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-lemmas-strict}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-defined}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-defined'}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-up-eq-asty}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-up-larger}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-up-istypeof-unfold}⟩
local-setup ⟨embedding-fun-info @{const-name print-iskindof-up-istypeof}⟩
```

local-setup ⟨*embedding-fun-info* @{*const-name print-iskindof-up-d-cast*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-def-id*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-lemmas-id*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-astype*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-exec*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-istypeof-pre*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-istypeof*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-iskindof-eq*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-allinst-iskindof-larger*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-oid-uniq-ml*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-oid-uniq*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-eval-extract*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-choose-ml*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-choose*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-deref-oid*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-deref-assocs*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-select*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-select-obj*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-dot-consts*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-dot*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-dot-lemmas-id*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-dot-cp-lemmas*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-dot-lemma-cp*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-dot-lemmas-cp*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-lemma-strict*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-def-mono*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-is-repr*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-access-repr-allinst*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-examp-def-st-defs*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-astype-lemmas-id2*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-enum*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-examp-instance-defassoc-typecheck-var*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-examp-instance-defassoc*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-examp-instance*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-examp-instance-defassoc-typecheck*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-examp-oclbase*}⟩
local-setup ⟨*embedding-fun-info-f1* @{*const-name Floor1-examp.print-examp-def-st-typecheck-var*}⟩
local-setup ⟨*embedding-fun-info-f1* @{*const-name Floor1-examp.print-examp-def-st1*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st-locale*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st2*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st-dom*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st-dom-lemmas*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st-perm*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st-allinst*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st-defassoc-typecheck*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-examp-def-st-def-interp*}⟩
local-setup ⟨*embedding-fun-info-f1* @{*const-name Floor1-examp.print-transition*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-transition-locale*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-transition-interp*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-transition-def-state*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-transition-wff*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-transition-where*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-transition-def-interp*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-examp.print-transition-lemmas-oid*}⟩
local-setup ⟨*embedding-fun-info-f1* @{*const-name Floor1-ctxt.print-ctxt*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-ctxt.print-ctxt-pre-post*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-ctxt.print-ctxt-inv*}⟩
local-setup ⟨*embedding-fun-info-f2* @{*const-name Floor2-ctxt.print-ctxt-thm*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-meta-setup-def-state*}⟩
local-setup ⟨*embedding-fun-info* @{*const-name print-meta-setup-def-transition*}⟩

Assembling Translations

definition *section-aux n s* = *start-map'* (λ-. [*O.section* (*Section n s*)])
definition *section* = *section-aux 0*
definition *subsection* = *section-aux 1*
definition *subsubsection* = *section-aux 2*
definition *txt f* = *Embedding-fun-simple* (*start-map''''''* *O.text o* (λ- *n-thy design-analysis*. [*Text* (*f n-thy design-analysis*)]))
definition *txt-raw f* = *Embedding-fun-simple* (*start-map''''''* *O.text-raw o* (λ- *n-thy design-analysis*. [*Text-raw* (*f n-thy design-analysis*)]))
definition *txt' s* = *txt* (λ- -. *s*)
definition *txt''* = *txt' o S.flatten*
definition *txt''d s* = *txt* (λ -. λ *Gen-only-design* ⇒ *S.flatten* (*s*) | - ⇒ ‹›)

```
definition txt''d' s = txt (λ n-thy. λ Gen-only-design ⇒ S.flatten (s n-thy) | - ⇒ ⟨⟩)
definition txt-raw''d' s = txt-raw (λ n-thy. λ Gen-only-design ⇒ S.flatten (s n-thy) | - ⇒ ⟨⟩)
definition txt''a s = txt (λ -. λ Gen-only-design ⇒ ⟨⟩ | - ⇒ S.flatten s)
definition txt''a' s = txt (λ n-thy. λ Gen-only-design ⇒ ⟨⟩ | - ⇒ S.flatten (s n-thy))
definition txt-raw''a' s = txt-raw (λ n-thy. λ Gen-only-design ⇒ ⟨⟩ | - ⇒ S.flatten (s n-thy))

definition' thy-class ::

 - embedding' where ⟨thy-class =
 (let section = Embedding-fun-simple o section o (λs. ⟨Class Model: ⟩ @@ s)
    ; subsection = Embedding-fun-simple o subsection
    ; subsection-def = subsection ⟨Definition⟩
    ; subsection-cp = subsection ⟨Context Passing⟩
    ; subsection-exec = subsection ⟨Execution with Invalid or Null as Argument⟩
    ; subsection-defined = subsection ⟨Validity and Definedness Properties⟩
    ; subsection-up = subsection ⟨Up Down Casting⟩
    ; subsection-const = subsection ⟨Const⟩ in
 (Embed-theories o L.flatten)
   [ [ PRINT-infra-enum-synonym ]
   , [ section ⟨The Construction of the Object Universe⟩
     (∗, PRINT-latex-infra-datatype-class∗)
     , PRINT-infra-datatype-class
     , PRINT-infra-datatype-universe
     , PRINT-infra-type-synonym-class
     , PRINT-infra-type-synonym-class-higher
     , PRINT-infra-type-synonym-class-rec
     , PRINT-infra-enum-syn
     , PRINT-infra-instantiation-class
     , PRINT-infra-instantiation-universe

     , section ⟨Instantiation of the Generic Strict Equality⟩
     , PRINT-instantia-def-strictrefeq
     , PRINT-instantia-lemmas-strictrefeq ]

   , L.flatten (L.map (λ(title, body-def, body-cp, body-exec, body-defined, body-up, body-const).
       section title # L.flatten [ subsection-def # body-def
                                 , subsection-cp # body-cp
                                 , subsection-exec # body-exec
                                 , subsection-defined # body-defined
                                 , subsection-up # body-up
                                 , subsection-const # body-const ])
         [ (⟨OclAsType⟩,
           [ PRINT-astype-consts
           , PRINT-astype-class
           , PRINT-astype-from-universe
           , PRINT-astype-lemmas-id ]
           , [ PRINT-astype-lemma-cp
           , PRINT-astype-lemmas-cp ]
           , [ PRINT-astype-lemma-strict
           , PRINT-astype-lemmas-strict ]
           , [ PRINT-astype-defined ]
           , [ PRINT-astype-up-d-cast0
           , PRINT-astype-up-d-cast
           , PRINT-astype-d-up-cast ]
           , [ PRINT-astype-lemma-const
             , PRINT-astype-lemmas-const ])

         , (⟨OclIsTypeOf⟩,
           [ PRINT-istypeof-consts
           , PRINT-istypeof-class
           , PRINT-istypeof-from-universe
           , PRINT-istypeof-lemmas-id ]
           , [ PRINT-istypeof-lemma-cp
           , PRINT-istypeof-lemmas-cp ]
           , [ PRINT-istypeof-lemma-strict
           , PRINT-istypeof-lemmas-strict ]
           , [ PRINT-istypeof-defined
           , PRINT-istypeof-defined' ]
           , [ PRINT-istypeof-up-larger
           , PRINT-istypeof-up-d-cast ]
           , [])

         , (⟨OclIsKindOf⟩,
```

```
    [ PRINT-iskindof-consts
    , PRINT-iskindof-class
    , PRINT-iskindof-from-universe
    , PRINT-iskindof-lemmas-id ]
    , [ PRINT-iskindof-lemma-cp
    , PRINT-iskindof-lemmas-cp ]
    , [ PRINT-iskindof-lemma-strict
    , PRINT-iskindof-lemmas-strict ]
    , [ PRINT-iskindof-defined
    , PRINT-iskindof-defined' ]
    , [ PRINT-iskindof-up-eq-asty
    , PRINT-iskindof-up-larger
    , PRINT-iskindof-up-istypeof-unfold
    , PRINT-iskindof-up-istypeof
    , PRINT-iskindof-up-d-cast ]
    , []) ])

  , [ section ‹OclAllInstances›
    , PRINT-allinst-def-id
    , PRINT-allinst-lemmas-id
    , PRINT-allinst-astype
    , PRINT-allinst-exec
    , subsection ‹OclIsTypeOf›
    , PRINT-allinst-istypeof-pre
    , PRINT-allinst-istypeof
    , subsection ‹OclIsKindOf›
    , PRINT-allinst-iskindof-eq
    , PRINT-allinst-iskindof-larger

    , section ‹The Accessors›
    , subsection-def
    , PRINT-access-oid-uniq-ml
    , PRINT-access-oid-uniq
    , PRINT-access-eval-extract
    , PRINT-access-choose-ml
    , PRINT-access-choose
    , PRINT-access-deref-oid
    , PRINT-access-deref-assocs
    , PRINT-access-select
    , PRINT-access-select-obj
    , PRINT-access-dot-consts
    , PRINT-access-dot
    , PRINT-access-dot-lemmas-id
    , subsection-cp
    , PRINT-access-dot-cp-lemmas
    , PRINT-access-dot-lemma-cp
    , PRINT-access-dot-lemmas-cp
    , subsection-exec
    , PRINT-access-lemma-strict
    , subsection ‹Representation in States›
    , PRINT-access-def-mono
    , PRINT-access-is-repr
    , PRINT-access-repr-allinst

    , section ‹Towards the Object Instances›
    , PRINT-examp-def-st-defs
    , PRINT-astype-lemmas-id2 ] ])›

definition thy-enum-flat = Embed-theories []
definition  thy-enum :: - embedding' where
      thy-enum = Embed-theories [ Embedding-fun-simple (section (‹Enum›))
                                , PRINT-enum ]
definition thy-class-synonym = Embed-theories []
definition thy-class-tree = Embed-theories []
definition thy-class-flat = Embed-theories []
definition thy-association = Embed-theories []
definition  thy-instance :: - embedding' where
      thy-instance = Embed-theories
                     [ Embedding-fun-simple (section (‹Instance›))
                     , PRINT-examp-instance-defassoc-typecheck-var
                     , PRINT-examp-instance-defassoc
                     , PRINT-examp-instance
                     , PRINT-examp-instance-defassoc-typecheck ]
```

```
definition thy-def-base-l :: - embedding' where
        thy-def-base-l = Embed-theories [ Embedding-fun-simple (section (‹BaseType›))
                                        , PRINT-examp-oclbase ]
definition thy-def-state = (λ Floor1 ⇒ Embed-theories
                                        [ Embedding-fun-simple (section (‹State (Floor 1)›))
                                        , floor1-PRINT-examp-def-st-typecheck-var
                                        , floor1-PRINT-examp-def-st1 ]
                           | Floor2 ⇒ Embed-locale
                                        [ Embedding-fun-simple (section (‹State (Floor 2)›)) ]
                                        Floor2-examp.print-examp-def-st-locale
                                        [ Floor2-examp.print-examp-def-st2
                                        , Floor2-examp.print-examp-def-st-dom
                                        , Floor2-examp.print-examp-def-st-dom-lemmas
                                        , Floor2-examp.print-examp-def-st-perm
                                        , Floor2-examp.print-examp-def-st-allinst
                                        , Floor2-examp.print-examp-def-st-defassoc-typecheck ]
                                        [ floor2-PRINT-examp-def-st-def-interp ])
definition thy-def-transition = (λ Floor1 ⇒ Embed-theories
                                          [ Embedding-fun-simple (section (‹Transition (Floor 1)›))
                                          , floor1-PRINT-transition ]
                                 | Floor2 ⇒ Embed-locale
                                          [ Embedding-fun-simple (section (‹Transition (Floor 2)›)) ]
                                          Floor2-examp.print-transition-locale
                                          [ Floor2-examp.print-transition-interp
                                          , Floor2-examp.print-transition-def-state
                                          , Floor2-examp.print-transition-wff
                                          , Floor2-examp.print-transition-where ]
                                          [ floor2-PRINT-transition-def-interp
                                          , floor2-PRINT-transition-lemmas-oid ])
definition thy-ctxt = (λ Floor1 ⇒ Embed-theories
                                  [ Embedding-fun-simple (section (‹Context (Floor 1)›))
                                  , floor1-PRINT-ctxt ]
                       | Floor2 ⇒ Embed-theories
                                  [ Embedding-fun-simple (section (‹Context (Floor 2)›))
                                  , floor2-PRINT-ctxt-pre-post
                                  , floor2-PRINT-ctxt-inv
                                  , floor2-PRINT-ctxt-thm ])
definition thy-flush-all = Embed-theories []
```

Combinators Folding the Compiling Environment

```
definition compiler-env-config-reset-all env =
  (let env = compiler-env-config-reset-no-env env in
   ( env (| D-input-meta := [] |)
   , let (l-class, l-env) = find-class-ass env in
     L.flatten
       [ l-class
       , List.filter (λ META-flush-all - ⇒ False | - ⇒ True) l-env
       , [META-flush-all OclFlushAll] ] ))

definition fold-thy0 meta thy-object0 f =
  L-fold (λx (acc1, acc2).
    let (sorry, dirty) = D-output-sorry-dirty acc1
      ; (msg, x) = case x of Embedding-fun-info msg x ⇒ (Some msg, x)
                           | Embedding-fun-simple x ⇒ (None, x)
      ; (l, acc1) = x meta acc1 in
    (f msg
       (if sorry = Some Gen-sorry | sorry = None & dirty then
          L.map (map-semi--theory (map-lemma (λ Lemma n spec - - ⇒ Lemma n spec [] C.sorry
                                              | Lemma-assumes n spec1 spec2 - - ⇒ Lemma-assumes n spec1 spec2 [] C.sorry))) l
        else
          l) acc1 acc2)) thy-object0

definition comp-env-input-class-rm f-fold f env-accu =
  (let (env, accu) = f-fold f env-accu in
   (env (| D-input-class := None |), accu))

definition comp-env-save ast f-fold f env-accu =
  (let (env, accu) = f-fold f env-accu in
   (env (| D-input-meta := ast # D-input-meta env |), accu))

definition comp-env-save-deep ast f-fold =
  comp-env-save ast (λf. map-prod
```

```
  (case ast of META-def-state Floor1 meta ⇒ Floor1-examp.print-meta-setup-def-state meta
             | META-def-transition Floor1 meta ⇒ Floor1-examp.print-meta-setup-def-transition meta
             | - ⇒ id)
  id o
  f-fold f)

definition comp-env-input-class-mk f-try f-accu-reset f-fold f =
 (λ (env, accu).
   f-fold f
     (case D-input-class env of Some - ⇒ (env, accu) | None ⇒
      let (l-class, l-env) = find-class-ass env
        ; (l-enum, l-env) = partition (λMETA-enum - ⇒ True | - ⇒ False) l-env in
      (f-try (λ () ⇒
        let D-input-meta0 = D-input-meta env
          ; (env, accu) =
            let meta = class-unflat' (arrange-ass True (D-ocl-semantics env ≠ Gen-default) l-class (L.map (λ META-enum e ⇒
e) l-enum))
              ; (env, accu) = List.fold (λ ast. comp-env-save ast (case ast of META-enum meta ⇒ fold-thy0 meta thy-enum) f)
                                        l-enum
                                        (let env = compiler-env-config-reset-no-env env in
                                        (env (| D-input-meta := List.filter (λ META-enum - ⇒ False | - ⇒ True) (D-input-meta env)
|), f-accu-reset env accu))
              ; (env, accu) = fold-thy0 meta thy-class f (env, accu) in
            (env (| D-input-class := Some meta |), accu)
        ; (env, accu) =
            List.fold
              (λast. comp-env-save ast (case ast of
                  META-instance meta ⇒ fold-thy0 meta thy-instance
                | META-def-base-l meta ⇒ fold-thy0 meta thy-def-base-l
                | META-def-state floor meta ⇒ fold-thy0 meta (thy-def-state floor)
                | META-def-transition floor meta ⇒ fold-thy0 meta (thy-def-transition floor)
                | META-ctxt floor meta ⇒ fold-thy0 meta (thy-ctxt floor)
                | META-flush-all meta ⇒ fold-thy0 meta thy-flush-all)
                  f)
              l-env
              (env (| D-input-meta := L.flatten [l-class, l-enum] |), accu) in
        (env (| D-input-meta := D-input-meta0 |), accu)))))

definition comp-env-input-class-bind l f =
  List.fold (λx. x f) l

definition fold-thy' f-env-save f-try f-accu-reset f =
 (let comp-env-input-class-mk = comp-env-input-class-mk f-try f-accu-reset in
  List.fold (λ ast.
    f-env-save ast (case ast of
     META-enum meta ⇒ comp-env-input-class-rm (fold-thy0 meta thy-enum-flat)
   | META-class-raw Floor1 meta ⇒ comp-env-input-class-rm (fold-thy0 meta thy-class-flat)
   | META-association meta ⇒ comp-env-input-class-rm (fold-thy0 meta thy-association)
   | META-ass-class Floor1 (OclAssClass meta-ass meta-class) ⇒
       comp-env-input-class-rm (comp-env-input-class-bind [ fold-thy0 meta-ass thy-association
                                                          , fold-thy0 meta-class thy-class-flat ])
   | META-class-synonym meta ⇒ comp-env-input-class-rm (fold-thy0 meta thy-class-synonym)
   | META-class-tree meta ⇒ comp-env-input-class-rm (fold-thy0 meta thy-class-tree)
   | META-instance meta ⇒ comp-env-input-class-mk (fold-thy0 meta thy-instance)
   | META-def-base-l meta ⇒ fold-thy0 meta thy-def-base-l
   | META-def-state floor meta ⇒ comp-env-input-class-mk (fold-thy0 meta (thy-def-state floor))
   | META-def-transition floor meta ⇒ fold-thy0 meta (thy-def-transition floor)
   | META-ctxt floor meta ⇒ comp-env-input-class-mk (fold-thy0 meta (thy-ctxt floor))
   | META-flush-all meta ⇒ comp-env-input-class-mk (fold-thy0 meta thy-flush-all)) f))

definition compiler-env-config-update f env =
  (* WARNING The semantics of the meta−embedded language is not intended to be reset here (like oid-start), only syntactic
configurations of the compiler (path, etc...) *)
 (let env' = f env in
  if D-input-meta env = [] then
    env'
      (| D-output-disable-thy := D-output-disable-thy env
      , D-output-header-thy := D-output-header-thy env
      (*D-ocl-oid-start*)
      (*D-output-position*)
      , D-ocl-semantics := D-ocl-semantics env
      (*D-input-class*)
      (*D-input-meta*)
```

```
    (*D-input-instance*)
    (*D-input-state*)
    (*D-output-header-force*)
    (*D-output-auto-bootstrap*)
    (*D-ocl-accessor*)
    (*D-ocl-HO-type*)
    , D-output-sorry-dirty := D-output-sorry-dirty env ⦆
  else
    fst (fold-thy'
          comp-env-save-deep
          (λf. f ())
          (λ-. id)
          (λ- -. Pair)
          (D-input-meta env')
          (env, ())))

definition fold-thy-shallow f-try f-accu-reset x =
  fold-thy'
    comp-env-save
    f-try
    f-accu-reset
    (λname l acc1.
      map-prod (λ env. env (| D-input-meta := D-input-meta acc1 |)) id
      o List.fold (x name) l
      o Pair acc1)

definition fold-thy-deep obj env =
  (case fold-thy'
          comp-env-save-deep
          (λf. f ())
          (λenv -. D-output-position env)
          (λ- l acc1 (i, cpt). (acc1, (Succ i, natural-of-nat (List.length l) + cpt)))
          obj
          (env, D-output-position env) of
    (env, output-position) ⇒ env (| D-output-position := output-position |))

end
```

APPENDIX G

HOL-OCL 2.0: Parsing Meta-Models

This chapter complements the chapter "Parsing Meta-Models" of the document "A Meta-Model for the Isabelle API" [TW15].

G.1 Instantiating the Parser of OCL (I)

theory *Parser-UML*
imports *Meta-UML*
../../compiler-generic/meta-isabelle/Parser-Pure
begin

Building Recursors for Records

definition *ocl-multiplicity-rec0 f ocl = f*
(*TyMult ocl*)
(*TyRole ocl*)
(*TyCollect ocl*)

definition *ocl-multiplicity-rec f ocl = ocl-multiplicity-rec0 f ocl*
(*ocl-multiplicity.more ocl*)

definition *ocl-ty-class-node-rec0 f ocl = f*
(*TyObjN-ass-switch ocl*)
(*TyObjN-role-multip ocl*)
(*TyObjN-role-ty ocl*)

definition *ocl-ty-class-node-rec f ocl = ocl-ty-class-node-rec0 f ocl*
(*ocl-ty-class-node.more ocl*)

definition *ocl-ty-class-rec0 f ocl = f*
(*TyObj-name ocl*)
(*TyObj-ass-id ocl*)
(*TyObj-ass-arity ocl*)
(*TyObj-from ocl*)
(*TyObj-to ocl*)

definition *ocl-ty-class-rec f ocl = ocl-ty-class-rec0 f ocl*
(*ocl-ty-class.more ocl*)

definition *ocl-class-raw-rec0 f ocl = f*
(*ClassRaw-name ocl*)
(*ClassRaw-own ocl*)
(*ClassRaw-clause ocl*)
(*ClassRaw-abstract ocl*)

definition *ocl-class-raw-rec f ocl = ocl-class-raw-rec0 f ocl*
(*ocl-class-raw.more ocl*)

definition *ocl-association-rec0 f ocl = f*
(*OclAss-type ocl*)
(*OclAss-relation ocl*)

definition *ocl-association-rec f ocl = ocl-association-rec0 f ocl*

(*ocl-association.more ocl*)

definition *ocl-ctxt-pre-post-rec0 f ocl = f*
(*Ctxt-fun-name ocl*)
(*Ctxt-fun-ty ocl*)
(*Ctxt-expr ocl*)

definition *ocl-ctxt-pre-post-rec f ocl = ocl-ctxt-pre-post-rec0 f ocl*
(*ocl-ctxt-pre-post.more ocl*)

definition *ocl-ctxt-rec0 f ocl = f*
(*Ctxt-param ocl*)
(*Ctxt-ty ocl*)
(*Ctxt-clause ocl*)

definition *ocl-ctxt-rec f ocl = ocl-ctxt-rec0 f ocl*
(*ocl-ctxt.more ocl*)

lemma [*code*]: *ocl-class-raw.extend = (λocl v. ocl-class-raw-rec0 (co4 (λf. f v) ocl-class-raw-ext) ocl)*
by(*intro ext, simp add*: *ocl-class-raw-rec0-def*
ocl-class-raw.extend-def
co4-def K-def)
lemma [*code*]: *ocl-class-raw.make = co4 (λf. f ()) ocl-class-raw-ext*
by(*intro ext, simp add*: *ocl-class-raw.make-def*
co4-def)
lemma [*code*]: *ocl-class-raw.truncate = ocl-class-raw-rec (co4 K ocl-class-raw.make)*
by(*intro ext, simp add*: *ocl-class-raw-rec0-def*
ocl-class-raw-rec-def
ocl-class-raw.truncate-def
ocl-class-raw.make-def
co4-def K-def)

lemma [*code*]: *ocl-association.extend = (λocl v. ocl-association-rec0 (co2 (λf. f v) ocl-association-ext) ocl)*
by(*intro ext, simp add*: *ocl-association-rec0-def*
ocl-association.extend-def
co2-def K-def)
lemma [*code*]: *ocl-association.make = co2 (λf. f ()) ocl-association-ext*
by(*intro ext, simp add*: *ocl-association.make-def*
co2-def)
lemma [*code*]: *ocl-association.truncate = ocl-association-rec (co2 K ocl-association.make)*
by(*intro ext, simp add*: *ocl-association-rec0-def*
ocl-association-rec-def
ocl-association.truncate-def
ocl-association.make-def
co2-def K-def)

Main

context *Parse*
begin

definition *of-ocl-collection b = rec-ocl-collection*
(*b ‹Set›*)
(*b ‹Sequence›*)
(*b ‹Ordered0›*)
(*b ‹Subsets0›*)
(*b ‹Union0›*)
(*b ‹Redefines0›*)
(*b ‹Derived0›*)
(*b ‹Qualifier0›*)
(*b ‹Nonunique0›*)

definition *of-ocl-multiplicity-single a b = rec-ocl-multiplicity-single*
(*ap1 a (b ‹Mult-nat›) (of-nat a b)*)
(*b ‹Mult-star›*)
(*b ‹Mult-infinity›*)

definition *of-ocl-multiplicity a b f = ocl-multiplicity-rec*
(*ap4 a (b (ext ‹ocl-multiplicity-ext›)*)
(*of-list a b (of-pair a b (of-ocl-multiplicity-single a b) (of-option a b (of-ocl-multiplicity-single a b))))*
(*of-option a b (of-string a b)*)

```
    (of-list a b (of-ocl-collection b))
    (f a b))

definition of-ocl-ty-class-node a b f = ocl-ty-class-node-rec
  (ap4 a (b (ext ‹ocl-ty-class-node-ext›))
    (of-nat a b)
    (of-ocl-multiplicity a b (K of-unit))
    (of-string a b)
    (f a b))

definition of-ocl-ty-class a b f = ocl-ty-class-rec
  (ap6 a (b (ext ‹ocl-ty-class-ext›))
    (of-string a b)
    (of-nat a b)
    (of-nat a b)
    (of-ocl-ty-class-node a b (K of-unit))
    (of-ocl-ty-class-node a b (K of-unit))
    (f a b))

definition of-ocl-ty-obj-core a b = rec-ocl-ty-obj-core
  (ap1 a (b ‹OclTyCore-pre›) (of-string a b))
  (ap1 a (b ‹OclTyCore›) (of-ocl-ty-class a b (K of-unit)))

definition of-ocl-ty-obj a b = rec-ocl-ty-obj
  (ap2 a (b ‹OclTyObj›) (of-ocl-ty-obj-core a b) (of-list a b (of-list a b (of-ocl-ty-obj-core a b))))

definition of-ocl-ty a b = (λf1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12 f13 f14 f15.
                              rec-ocl-ty f1 f2 f3 f4 f5 f6
                                  f7 (K o f8) (λ- -. f9) (f10 o map-prod id snd) (λ- -. f11) f12 f13 f14 f15)
  (b ‹OclTy-base-void›)
  (b ‹OclTy-base-boolean›)
  (b ‹OclTy-base-integer›)
  (b ‹OclTy-base-unlimitednatural›)
  (b ‹OclTy-base-real›)
  (b ‹OclTy-base-string›)
  (ap1 a (b ‹OclTy-object›) (of-ocl-ty-obj a b))
  (ar2 a (b ‹OclTy-collection›) (of-ocl-multiplicity a b (K of-unit)))
  (ar2 a (b ‹OclTy-pair›) id)
  (ap1 a (b ‹OclTy-binding›) (of-pair a b (of-option a b (of-string a b)) id))
  (ar2 a (b ‹OclTy-arrow›) id)
  (ap1 a (b ‹OclTy-class-syn›) (of-string a b))
  (ap1 a (b ‹OclTy-enum›) (of-string a b))
  (ap1 a (b ‹OclTy-raw›) (of-string a b))

definition of-ocl-association-type a b = rec-ocl-association-type
  (b ‹OclAssTy-native-attribute›)
  (b ‹OclAssTy-association›)
  (b ‹OclAssTy-composition›)
  (b ‹OclAssTy-aggregation›)

definition of-ocl-association-relation a b = rec-ocl-association-relation
  (ap1 a (b ‹OclAssRel›)
    (of-list a b (of-pair a b (of-ocl-ty-obj a b) (of-ocl-multiplicity a b (K of-unit)))))

definition of-ocl-association a b f = ocl-association-rec
  (ap3 a (b (ext ‹ocl-association-ext›))
    (of-ocl-association-type a b)
    (of-ocl-association-relation a b)
    (f a b))

definition of-ocl-ctxt-prefix a b = rec-ocl-ctxt-prefix
  (b ‹OclCtxtPre›)
  (b ‹OclCtxtPost›)

definition of-ocl-ctxt-term a b = (λf0 f1 f2. rec-ocl-ctxt-term f0 f1 (co1 K f2))
  (ap2 a (b ‹T-pure›) (of-pure-term a b) (of-option a b (of-string a b)))
  (ap2 a (b ‹T-to-be-parsed›) (of-string a b) (of-string a b))
  (ar2 a (b ‹T-lambda›) (of-string a b))

definition of-ocl-prop a b = rec-ocl-prop
  (ap2 a (b ‹OclProp-ctxt›) (of-option a b (of-string a b)) (of-ocl-ctxt-term a b))

definition of-ocl-ctxt-term-inv a b = rec-ocl-ctxt-term-inv
```

```
  (ap2 a (b ‹T-inv›) (of-bool b) (of-ocl-prop a b))

definition of-ocl-ctxt-term-pp a b = rec-ocl-ctxt-term-pp
  (ap2 a (b ‹T-pp›) (of-ocl-ctxt-prefix a b) (of-ocl-prop a b))
  (ap1 a (b ‹T-invariant›) (of-ocl-ctxt-term-inv a b))

definition of-ocl-ctxt-pre-post a b f = ocl-ctxt-pre-post-rec
  (ap4 a (b (ext ‹ocl-ctxt-pre-post-ext›))
    (of-string a b)
    (of-ocl-ty a b)
    (of-list a b (of-ocl-ctxt-term-pp a b))
    (f a b))

definition of-ocl-ctxt-clause a b = rec-ocl-ctxt-clause
  (ap1 a (b ‹Ctxt-pp›) (of-ocl-ctxt-pre-post a b (K of-unit)))
  (ap1 a (b ‹Ctxt-inv›) (of-ocl-ctxt-term-inv a b))

definition of-ocl-ctxt a b f = ocl-ctxt-rec
  (ap4 a (b (ext ‹ocl-ctxt-ext›))
    (of-list a b (of-string a b))
    (of-ocl-ty-obj a b)
    (of-list a b (of-ocl-ctxt-clause a b))
    (f a b))

definition of-ocl-class a b = (λf0 f1 f2 f3. rec-ocl-class (ap3 a f0 f1 f2 f3))
  (b ‹OclClass›)
    (of-string a b)
    (of-list a b (of-pair a b (of-string a b) (of-ocl-ty a b)))
    (of-list a b snd)

definition of-ocl-class-raw a b f = ocl-class-raw-rec
  (ap5 a (b (ext ‹ocl-class-raw-ext›))
    (of-ocl-ty-obj a b)
    (of-list a b (of-pair a b (of-string a b) (of-ocl-ty a b)))
    (of-list a b (of-ocl-ctxt-clause a b))
    (of-bool b)
    (f a b))

definition of-ocl-ass-class a b = rec-ocl-ass-class
  (ap2 a (b ‹OclAssClass›)
    (of-ocl-association a b (K of-unit))
    (of-ocl-class-raw a b (K of-unit)))

definition of-ocl-class-synonym a b = rec-ocl-class-synonym
  (ap2 a (b ‹OclClassSynonym›)
    (of-string a b)
    (of-ocl-ty a b))

definition of-ocl-enum a b = rec-ocl-enum
  (ap2 a (b ‹OclEnum›)
    (of-string a b)
    (of-list a b (of-string a b)))

end

lemmas [code] =
  Parse.of-ocl-collection-def
  Parse.of-ocl-multiplicity-single-def
  Parse.of-ocl-multiplicity-def
  Parse.of-ocl-ty-class-node-def
  Parse.of-ocl-ty-class-def
  Parse.of-ocl-ty-obj-core-def
  Parse.of-ocl-ty-obj-def
  Parse.of-ocl-ty-def
  Parse.of-ocl-association-type-def
  Parse.of-ocl-association-relation-def
  Parse.of-ocl-association-def
  Parse.of-ocl-ctxt-prefix-def
  Parse.of-ocl-ctxt-term-def
  Parse.of-ocl-prop-def
  Parse.of-ocl-ctxt-term-inv-def
  Parse.of-ocl-ctxt-term-pp-def
  Parse.of-ocl-ctxt-pre-post-def
```

Parse.of-ocl-ctxt-clause-def
Parse.of-ocl-ctxt-def
Parse.of-ocl-class-def
Parse.of-ocl-class-raw-def
Parse.of-ocl-ass-class-def
Parse.of-ocl-class-synonym-def
Parse.of-ocl-enum-def

end

G.2 Instantiating the Parser of OCL (II)

theory *Parser-UML-extended*
imports *Meta-UML-extended*
../../compiler-generic/meta-isabelle/Parser-init
begin

Building Recursors for Records

definition *ocl-instance-single-rec0 f ocl = f*
(*Inst-name ocl*)
(*Inst-ty ocl*)
(*Inst-attr-with ocl*)
(*Inst-attr ocl*)

definition *ocl-instance-single-rec f ocl = ocl-instance-single-rec0 f ocl*
(*ocl-instance-single.more ocl*)

lemma [*code*]: *ocl-instance-single.extend = (λocl v. ocl-instance-single-rec0 (co4 (λf. f v) ocl-instance-single-ext) ocl)*
by(*intro ext*, *simp add*: *ocl-instance-single-rec0-def*
ocl-instance-single.extend-def
co4-def K-def)
lemma [*code*]: *ocl-instance-single.make = co4 (λf. f ()) ocl-instance-single-ext*
by(*intro ext*, *simp add*: *ocl-instance-single.make-def*
co4-def)
lemma [*code*]: *ocl-instance-single.truncate = ocl-instance-single-rec (co4 K ocl-instance-single.make)*
by(*intro ext*, *simp add*: *ocl-instance-single-rec0-def*
ocl-instance-single-rec-def
ocl-instance-single.truncate-def
ocl-instance-single.make-def
co4-def K-def)

Main

context *Parse*
begin

definition *of-internal-oid a b = rec-internal-oid*
(*ap1 a (b ⟨Oid⟩) (of-nat a b)*)

definition *of-internal-oids a b = rec-internal-oids*
(*ap3 a (b ⟨Oids⟩)*
(*of-nat a b*)
(*of-nat a b*)
(*of-nat a b*))

definition *of-ocl-def-base a b = rec-ocl-def-base*
(*ap1 a (b ⟨OclDefInteger⟩) (of-string a b)*)
(*ap1 a (b ⟨OclDefReal⟩) (of-pair a b (of-string a b) (of-string a b))*)
(*ap1 a (b ⟨OclDefString⟩) (of-string a b)*)

definition *of-ocl-data-shallow a b = rec-ocl-data-shallow*
(*ap1 a (b ⟨ShallB-term⟩) (of-ocl-def-base a b)*)
(*ap1 a (b ⟨ShallB-str⟩) (of-string a b)*)
(*ap1 a (b ⟨ShallB-self⟩) (of-internal-oid a b)*)
(*ap1 a (b ⟨ShallB-list⟩) (of-list a b snd)*)

definition *of-ocl-list-attr a b f = (λf0. co4 (λf1. rec-ocl-list-attr f0 (λs - a rec. f1 s rec a)) (ap3 a))*
(*ap1 a (b ⟨OclAttrNoCast⟩) f*)

```
  (b ⟨OclAttrCast⟩)
    (of-string a b)
    id
    f

definition of-ocl-instance-single a b f = ocl-instance-single-rec
  (ap5 a (b (ext ⟨ocl-instance-single-ext⟩))
    (of-option a b (of-string a b))
    (of-option a b (of-string a b))
    (of-option a b (of-string a b))
    (of-ocl-list-attr a b (of-list a b (of-pair a b (of-option a b (of-pair a b (of-string a b) (of-string a b))) (of-pair a b (of-string a
b) (of-ocl-data-shallow a b)))))
    (f a b))

definition of-ocl-instance a b = rec-ocl-instance
  (ap1 a (b ⟨OclInstance⟩)
    (of-list a b (of-ocl-instance-single a b (K of-unit))))

definition of-ocl-def-base-l a b = rec-ocl-def-base-l
  (ap1 a (b ⟨OclDefBase⟩) (of-list a b (of-ocl-def-base a b)))

definition of-ocl-def-state-core a b f = rec-ocl-def-state-core
  (ap1 a (b ⟨OclDefCoreAdd⟩) (of-ocl-instance-single a b (K of-unit)))
  (ap1 a (b ⟨OclDefCoreBinding⟩) f)

definition of-ocl-def-state a b = rec-ocl-def-state
  (ap2 a (b ⟨OclDefSt⟩) (of-string a b) (of-list a b (of-ocl-def-state-core a b (of-string a b))))

definition of-ocl-def-pp-core a b = rec-ocl-def-pp-core
  (ap1 a (b ⟨OclDefPPCoreAdd⟩) (of-list a b (of-ocl-def-state-core a b (of-string a b))))
  (ap1 a (b ⟨OclDefPPCoreBinding⟩) (of-string a b))

definition of-ocl-def-transition a b = rec-ocl-def-transition
  (ap3 a (b ⟨OclDefPP⟩)
    (of-option a b (of-string a b))
    (of-ocl-def-pp-core a b)
    (of-option a b (of-ocl-def-pp-core a b)))

definition of-ocl-class-tree a b = rec-ocl-class-tree
  (ap2 a (b ⟨OclClassTree⟩)
    (of-nat a b)
    (of-nat a b))

end

lemmas [code] =
  Parse.of-internal-oid-def
  Parse.of-internal-oids-def
  Parse.of-ocl-def-base-def
  Parse.of-ocl-data-shallow-def
  Parse.of-ocl-list-attr-def
  Parse.of-ocl-instance-single-def
  Parse.of-ocl-instance-def
  Parse.of-ocl-def-base-l-def
  Parse.of-ocl-def-state-core-def
  Parse.of-ocl-def-state-def
  Parse.of-ocl-def-pp-core-def
  Parse.of-ocl-def-transition-def
  Parse.of-ocl-class-tree-def

end
```

G.3 Instantiating the Parser of META

```
theory Parser-META
imports Meta-META
        Parser-UML
        Parser-UML-extended
begin
```

Building Recursors for Records

definition *compiler-env-config-rec0 f env = f*
(*D-output-disable-thy env*)
(*D-output-header-thy env*)
(*D-ocl-oid-start env*)
(*D-output-position env*)
(*D-ocl-semantics env*)
(*D-input-class env*)
(*D-input-meta env*)
(*D-input-instance env*)
(*D-input-state env*)
(*D-output-header-force env*)
(*D-output-auto-bootstrap env*)
(*D-ocl-accessor env*)
(*D-ocl-HO-type env*)
(*D-output-sorry-dirty env*)

definition *compiler-env-config-rec f env = compiler-env-config-rec0 f env*
(*compiler-env-config.more env*)

lemma [*code*]: *compiler-env-config.extend = (λenv v. compiler-env-config-rec0 (co14 (λf. f v) compiler-env-config-ext) env)*
by(*intro ext*, *simp add*: *compiler-env-config-rec0-def*
compiler-env-config.extend-def
co14-def K-def)
lemma [*code*]: *compiler-env-config.make = co14 (λf. f ()) compiler-env-config-ext*
by(*intro ext*, *simp add*: *compiler-env-config.make-def*
co14-def)
lemma [*code*]: *compiler-env-config.truncate = compiler-env-config-rec (co14 K compiler-env-config.make)*
by(*intro ext*, *simp add*: *compiler-env-config-rec0-def*
compiler-env-config-rec-def
compiler-env-config.truncate-def
compiler-env-config.make-def
co14-def K-def)

Main

context *Parse*
begin

definition *of-ocl-flush-all a b = rec-ocl-flush-all*
(*b ‹OclFlushAll›*)

definition *of-floor a b = rec-floor*
(*b ‹Floor1›*)
(*b ‹Floor2›*)
(*b ‹Floor3›*)

definition *of-all-meta-embedding a b = rec-all-meta-embedding*
(*ap1 a (b ‹META-enum›) (of-ocl-enum a b)*)
(*ap2 a (b ‹META-class-raw›) (of-floor a b) (of-ocl-class-raw a b (K of-unit))*)
(*ap1 a (b ‹META-association›) (of-ocl-association a b (K of-unit))*)
(*ap2 a (b ‹META-ass-class›) (of-floor a b) (of-ocl-ass-class a b)*)
(*ap2 a (b ‹META-ctxt›) (of-floor a b) (of-ocl-ctxt a b (K of-unit))*)

(*ap1 a (b ‹META-class-synonym›) (of-ocl-class-synonym a b)*)
(*ap1 a (b ‹META-instance›) (of-ocl-instance a b)*)
(*ap1 a (b ‹META-def-base-l›) (of-ocl-def-base-l a b)*)
(*ap2 a (b ‹META-def-state›) (of-floor a b) (of-ocl-def-state a b)*)
(*ap2 a (b ‹META-def-transition›) (of-floor a b) (of-ocl-def-transition a b)*)
(*ap1 a (b ‹META-class-tree›) (of-ocl-class-tree a b)*)
(*ap1 a (b ‹META-flush-all›) (of-ocl-flush-all a b)*)

definition *of-generation-semantics-ocl a b = rec-generation-semantics-ocl*
(*b ‹Gen-only-design›*)
(*b ‹Gen-only-analysis›*)
(*b ‹Gen-default›*)

definition *of-generation-lemma-mode a b = rec-generation-lemma-mode*
(*b ‹Gen-sorry›*)
(*b ‹Gen-no-dirty›*)

definition *of-compiler-env-config a b f = compiler-env-config-rec*
(*ap15 a* (*b* (*ext* ‹*compiler-env-config-ext*›))
(*of-bool b*)
(*of-option a b* (*of-pair a b* (*of-string a b*) (*of-pair a b* (*of-list a b* (*of-string a b*)) (*of-string a b*))))
(*of-internal-oids a b*)
(*of-pair a b* (*of-nat a b*) (*of-nat a b*))
(*of-generation-semantics-ocl a b*)
(*of-option a b* (*of-ocl-class a b*))
(*of-list a b* (*of-all-meta-embedding a b*))
(*of-list a b* (*of-pair a b* (*of-string$_{base}$ a b*) (*of-pair a b* (*of-ocl-instance-single a b* (*K of-unit*)) (*of-internal-oids a b*))))
(*of-list a b* (*of-pair a b* (*of-string$_{base}$ a b*) (*of-list a b* (*of-pair a b* (*of-internal-oids a b*) (*of-ocl-def-state-core a b* (*of-pair a b* (*of-string a b*) (*of-ocl-instance-single a b* (*K of-unit*))))))))
(*of-bool b*)
(*of-bool b*)
(*of-pair a b* (*of-list a b* (*of-string$_{base}$ a b*)) (*of-list a b* (*of-string$_{base}$ a b*)))
(*of-list a b* (*of-string$_{base}$ a b*))
(*of-pair a b* (*of-option a b* (*of-generation-lemma-mode a b*)) (*of-bool b*))
(*f a b*))

end

lemmas [*code*] =
Parse.of-ocl-flush-all-def
Parse.of-floor-def
Parse.of-all-meta-embedding-def
Parse.of-generation-semantics-ocl-def
Parse.of-generation-lemma-mode-def
Parse.of-compiler-env-config-def

G.4 Finalizing the Parser

It should be feasible to invent a meta-command (e.g., *datatype′*) to automatically generate the previous recursors in *Parse*.

Otherwise as an extra check, one can also overload polymorphic cartouches in *Init* to really check that all the given constructor exists at the time of editing (similarly as writing `@{term ...}`, when it is embedded in a `text` command).

Isabelle Syntax

locale *Parse-Isabelle*
begin

definition *Of-Pair* = ‹*Pair*›
definition *Of-Nil* = ‹*Nil*›
definition *Of-Cons* = ‹*Cons*›
definition *Of-None* = ‹*None*›
definition *Of-Some* = ‹*Some*›

definition *of-pair a b f1 f2* = (λ*f*. λ(*c*, *d*) ⇒ *f c d*)
(*ap2 a* (*b Of-Pair*) *f1 f2*)

definition *of-list a b f* = (λ*f0*. *rec-list f0 o co1 K*)
(*b Of-Nil*)
(*ar2 a* (*b Of-Cons*) *f*)

definition *of-option a b f = rec-option*
(*b Of-None*)
(*ap1 a* (*b Of-Some*) *f*)

definition *of-unit b = case-unit*
(*b* ‹()›)

definition *of-bool* where *of-bool b = case-bool*
(*b* ‹*True*›)
(*b* ‹*False*›)

```
definition of-nibble b = rec-nibble
  (b ⟨Nibble0⟩)
  (b ⟨Nibble1⟩)
  (b ⟨Nibble2⟩)
  (b ⟨Nibble3⟩)
  (b ⟨Nibble4⟩)
  (b ⟨Nibble5⟩)
  (b ⟨Nibble6⟩)
  (b ⟨Nibble7⟩)
  (b ⟨Nibble8⟩)
  (b ⟨Nibble9⟩)
  (b ⟨NibbleA⟩)
  (b ⟨NibbleB⟩)
  (b ⟨NibbleC⟩)
  (b ⟨NibbleD⟩)
  (b ⟨NibbleE⟩)
  (b ⟨NibbleF⟩)

definition of-char a b = rec-char
  (ap2 a (b ⟨Char⟩) (of-nibble b) (of-nibble b))

definition of-string-gen s-flatten s-st0 s-st a b s =
  b (let s = textstr-of-str (λc. ⟨(⟩ @@ s-flatten @@ ⟨ ⟩ @@ c @@ ⟨)⟩)
                            (λChar n1 n2 ⇒
                                s-st0 (S.flatten [⟨ (⟩, ⟨Char ⟩, of-nibble id n1, ⟨ ⟩, of-nibble id n2, ⟨)⟩]))
                            (λc. s-st (S.flatten [⟨ (⟩, c, ⟨)⟩]))
                            s in
     S.flatten [ ⟨(⟩, s, ⟨)⟩ ])

definition of-string = of-string-gen ⟨Init.S.flatten⟩
                                     (λs. S.flatten [⟨(Init.ST0⟩, s, ⟨)⟩])
                                     (λs. S.flatten [⟨(Init.abr-string.SS-base (Init.string_base.ST⟩, s, ⟨))⟩])
definition of-string_base a b s = of-string-gen ⟨Init.String_base.flatten⟩
                                                (λs. S.flatten [⟨(Init.ST0-base⟩, s, ⟨)⟩])
                                                (λs. S.flatten [⟨(Init.string_base.ST⟩, s, ⟨)⟩])
                                                a
                                                b
                                                (String_base.to-String s)

definition of-nat where of-nat a b = b o String.of-natural

end

sublocale Parse-Isabelle < Parse id
                           Parse-Isabelle.of-string
                           Parse-Isabelle.of-string_base
                           Parse-Isabelle.of-nat
                           Parse-Isabelle.of-unit
                           Parse-Isabelle.of-bool
                           Parse-Isabelle.Of-Pair
                           Parse-Isabelle.Of-Nil
                           Parse-Isabelle.Of-Cons
                           Parse-Isabelle.Of-None
                           Parse-Isabelle.Of-Some
done

context Parse-Isabelle begin
  definition compiler-env-config a b =
    of-compiler-env-config a b (λ a b.
      of-pair a b
        (of-list a b (of-all-meta-embedding a b))
        (of-option a b (of-string a b)))
end

definition isabelle-of-compiler-env-config = Parse-Isabelle.compiler-env-config

lemmas [code] =
  Parse-Isabelle.Of-Pair-def
  Parse-Isabelle.Of-Nil-def
  Parse-Isabelle.Of-Cons-def
  Parse-Isabelle.Of-None-def
  Parse-Isabelle.Of-Some-def
```

Parse-Isabelle.of-pair-def
Parse-Isabelle.of-list-def
Parse-Isabelle.of-option-def
Parse-Isabelle.of-unit-def
Parse-Isabelle.of-bool-def
Parse-Isabelle.of-nibble-def
Parse-Isabelle.of-char-def
Parse-Isabelle.of-string-gen-def
Parse-Isabelle.of-string-def
Parse-Isabelle.of-string$_{base}$-def
Parse-Isabelle.of-nat-def

Parse-Isabelle.compiler-env-config-def

definition *isabelle-apply s l* = *S.flatten* [*s*, *S.flatten* (*L.map* (λ *s*. *S.flatten* [‹ (›, *s*, ‹)›]) *l*)]

SML Syntax

locale *Parse-SML*
begin

definition *Of-Pair* = ‹*I*›
definition *Of-Nil* = ‹*nil*›
definition *Of-Cons* = ‹*uncurry cons*›
definition *Of-None* = ‹*NONE*›
definition *Of-Some* = ‹*SOME*›

definition *of-pair a b f1 f2* = (λ*f*. λ(*c*, *d*) ⇒ *f c d*)
(*ap2 a* (*b Of-Pair*) *f1 f2*)

definition *of-list a b f* = (λ*f0*. *rec-list f0 o co1 K*)
(*b Of-Nil*)
(*ar2 a* (*b Of-Cons*) *f*)

definition *of-option a b f* = *rec-option*
(*b Of-None*)
(*ap1 a* (*b Of-Some*) *f*)

definition *of-unit b* = *case-unit*
(*b* ‹()›)

definition *of-bool* where *of-bool b* = *case-bool*
(*b* ‹*true*›)
(*b* ‹*false*›)

definition′ ‹*sml-escape* =
String.replace-chars ((∗ (∗ *ERROR code-reflect* ∗)
λ *Char Nibble0 NibbleA* ⇒ ‹*n*›
| *Char Nibble0 Nibble5* ⇒ ‹*005*›
| *Char Nibble0 Nibble6* ⇒ ‹*006*›
| *Char Nibble7 NibbleF* ⇒ ‹*127*›
| *x* ⇒ °*x*°∗)
λ*x*. *if x* = *Char Nibble0 NibbleA then* ‹*n*›
else if x = *Char Nibble0 Nibble5 then* ‹*005*›
else if x = *Char Nibble0 Nibble6 then* ‹*006*›
else if x = *Char Nibble7 NibbleF then* ‹*127*›
else °*x*°)›

definition′ ‹*of-string a b* =
(λ*x*. *b* (*S.flatten* [‹(*META.SS-base* (*META.ST* ›
, *sml-escape x*
, ‹))›]))›

definition′ ‹*of-string$_{base}$ a b* =
(λ*x*. *b* (*S.flatten* [‹(*META.ST* ›
, *sml-escape* (*String$_{base}$.to-String x*)

, ‹›]))›

definition *of-nat* where *of-nat a b* = (λ*x*. *b* (*S.flatten* [‹(*Code-Numeral.Nat* ›, *String.of-natural x*, ‹)›]))

end

sublocale *Parse-SML* < *Parse* λ*c*. *case String.to-list c of x # xs* ⇒ *S.flatten* [*String.uppercase* ≪[*x*]≫, ≪*xs*≫]
Parse-SML.of-string
Parse-SML.of-string$_{base}$
Parse-SML.of-nat
Parse-SML.of-unit
Parse-SML.of-bool
Parse-SML.Of-Pair
Parse-SML.Of-Nil
Parse-SML.Of-Cons
Parse-SML.Of-None
Parse-SML.Of-Some
done

context *Parse-SML* begin
definition *compiler-env-config a b* = *of-compiler-env-config a b* (λ -. *of-unit*)
end

definition *sml-of-compiler-env-config* = *Parse-SML.compiler-env-config*

lemmas [*code*] =
Parse-SML.Of-Pair-def
Parse-SML.Of-Nil-def
Parse-SML.Of-Cons-def
Parse-SML.Of-None-def
Parse-SML.Of-Some-def

Parse-SML.of-pair-def
Parse-SML.of-list-def
Parse-SML.of-option-def
Parse-SML.of-unit-def
Parse-SML.of-bool-def
Parse-SML.of-string-def
Parse-SML.of-string$_{base}$*-def*
Parse-SML.of-nat-def

Parse-SML.sml-escape-def
Parse-SML.compiler-env-config-def

definition *sml-apply s l* = *S.flatten* [*s*, ‹ (›, *case l of x # xs* ⇒ *S.flatten* [*x*, *S.flatten* (*L.map* (λ*s*. *S.flatten* [‹, ›, *s*]) *xs*)], ‹)›]

end

HOL-OCL 2.0: Printing Meta-Models

This chapter complements the chapter "Printing Meta-Models" of the document "A Meta-Model for the Isabelle API" [TW15].

H.1 Instantiating the Printer for OCL (I)

```
theory Printer-UML
imports Meta-UML
        ../../compiler-generic/meta-isabelle/Printer-Pure
begin

context Print
begin

declare[[cartouche-type' = abr-string]]

definition concatWith l =
 (if l = [] then
    id
  else
    sprint2 ≺''(%s. (%s))''≻' (To-string (String-concatWith ‹ › (‹λ› # rev l))))

declare[[cartouche-type' = fun_printf]]

fun of-ctxt2-term-aux where of-ctxt2-term-aux l e =
 (λ T-pure pure o-s ⇒ (case o-s of None ⇒ concatWith l (of-pure-term True [] pure)
                                | Some s ⇒ To-string s)
  | T-to-be-parsed - s ⇒ concatWith l (To-string s)
  | T-lambda s c ⇒ of-ctxt2-term-aux (s # l) c) e
definition of-ctxt2-term = of-ctxt2-term-aux []

definition' ‹of-ocl-ctxt - (floor :: (* polymorphism weakening needed by code-reflect *)
                                String.literal) ctxt =
 (let f-inv = λ T-inv b (OclProp-ctxt n s) ⇒ ‹  %sInv %s : %s›
              (if b then ‹Existential› else ‹›)
              (case n of None ⇒ ‹› | Some s ⇒ To-string s)
              (of-ctxt2-term s) in
  ‹Context%s %s%s %s›
       floor
       (case Ctxt-param ctxt of
          [] ⇒ ‹›
        | l ⇒ ‹%s : › (String-concat ‹, › (L.map To-string l)))
       (To-string (ty-obj-to-string (Ctxt-ty ctxt)))
       (String-concat ‹
› (L.map (λ Ctxt-pp ctxt ⇒
             ‹:: %s (%s) %s
%s›
       (To-string (Ctxt-fun-name ctxt))
       (String-concat ‹, ›
         (L.map
           (λ (s, ty). ‹%s : %s› (To-string s) (To-string (str-of-ty ty)))
           (Ctxt-fun-ty-arg ctxt)))
       (case Ctxt-fun-ty-out ctxt of None ⇒ ‹›
```

```
                                | Some ty ⇒ ‹: %s› (To-string (str-of-ty ty)))
      (String-concat ‹
›
        (L.map
          (λ T-pp pref (OclProp-ctxt n s) ⇒ ‹  %s %s: %s›
             (case pref of OclCtxtPre ⇒ ‹Pre›
                         | OclCtxtPost ⇒ ‹Post›)
             (case n of None ⇒ ‹› | Some s ⇒ To-string s)
             (of-ctxt2-term s)
           | T-invariant inva ⇒ f-inv inva)
          (Ctxt-expr ctxt)))
        | Ctxt-inv inva ⇒ f-inv inva)
       (Ctxt-clause ctxt))))›
```

end

lemmas [code] =

  Print.concatWith-def
  Print.of-ctxt2-term-def
  Print.of-ocl-ctxt-def

  Print.of-ctxt2-term-aux.simps

end

H.2 Instantiating the Printer for OCL (II)

theory Printer-UML-extended
imports Meta-UML-extended
        Printer-UML
begin

context Print
begin

definition To-oid = (λOid n ⇒ To-nat n)

definition′ ‹of-ocl-def-base = (λ OclDefInteger i ⇒ To-string i
                              | OclDefReal (i1, i2) ⇒ ‹%s.%s› (To-string i1) (To-string i2)
                              | OclDefString s ⇒ ‹%s› (To-string s))›

```
fun of-ocl-data-shallow where
  of-ocl-data-shallow e = (λ ShallB-term b ⇒ of-ocl-def-base b
                           | ShallB-str s ⇒ To-string s
                           | ShallB-self s ⇒ ‹self %d› (To-oid s)
                           | ShallB-list l ⇒ ‹[ %s ]› (String-concat ‹, › (List.map of-ocl-data-shallow l))) e
```

```
fun of-ocl-list-attr where
  of-ocl-list-attr f e = (λ OclAttrNoCast x ⇒ f x
                          | OclAttrCast ty (OclAttrNoCast x) - ⇒ ‹(%s :: %s)› (f x) (To-string ty)
                          | OclAttrCast ty l - ⇒ ‹%s → oclAsType( %s )› (of-ocl-list-attr f l) (To-string ty)) e
```

```
definition′ ‹of-ocl-instance-single ocli =
 (let (s-left, s-right) =
    case Inst-name ocli of
      None ⇒ (case Inst-ty ocli of Some ty ⇒ (‹(›, ‹ :: %s)› (To-string ty)))
    | Some s ⇒
        ( ‹%s%s = ›
            (To-string s)
            (case Inst-ty ocli of None ⇒ ‹› | Some ty ⇒ ‹ :: %s› (To-string ty))
        , ‹›) in
  ‹%s%s%s›
    s-left
    (of-ocl-list-attr
      (λl. ‹[ %s%s ]›
            (case Inst-attr-with ocli of None ⇒ ‹› | Some s ⇒ ‹%s with-only › (To-string s))
            (String-concat ‹, ›
              (L.map (λ(pre-post, attr, v).
                        ‹%s%s = %s› (case pre-post of None ⇒ ‹›
                                                    | Some (s1, s2) ⇒ ‹(%s, %s) |= › (To-string s1) (To-string s2))
                        (To-string attr)
```

```
                                    (of-ocl-data-shallow v))
                          l)))
      (Inst-attr ocli))
    s-right)⟩

definition of-ocl-instance - = (λ OclInstance l ⇒
  ⟨Instance %s⟩ (String-concat ⟨
    and ⟩ (L.map of-ocl-instance-single l)))

definition of-ocl-def-state-core l =
  String-concat ⟨, ⟩ (L.map (λ OclDefCoreBinding s ⇒ To-string s
                        | OclDefCoreAdd ocli ⇒ of-ocl-instance-single ocli) l)

definition of-ocl-def-state - (floor :: (* polymorphism weakening needed by code-reflect *)
                                  String.literal) = (λ OclDefSt n l ⇒
  ⟨State%s %s = [ %s ]⟩
    floor
    (To-string n)
    (of-ocl-def-state-core l))

definition of-ocl-def-pp-core = (λ OclDefPPCoreBinding s ⇒ To-string s
                              | OclDefPPCoreAdd l ⇒ ⟨[ %s ]⟩ (of-ocl-def-state-core l))

definition of-ocl-def-transition - (floor :: (* polymorphism weakening needed by code-reflect *)
                                    String.literal) = (λ OclDefPP n s-pre s-post ⇒
  ⟨Transition%s %s%s%s⟩
    floor
    (case n of None ⇒ ⟨⟩ | Some n ⇒ ⟨%s = ⟩ (To-string n))
    (of-ocl-def-pp-core s-pre)
    (case s-post of None ⇒ ⟨⟩ | Some s-post ⇒ ⟨ %s⟩ (of-ocl-def-pp-core s-post)))

end

lemmas [code] =

  Print.To-oid-def
  Print.of-ocl-def-base-def
  Print.of-ocl-instance-single-def
  Print.of-ocl-instance-def
  Print.of-ocl-def-state-core-def
  Print.of-ocl-def-state-def
  Print.of-ocl-def-pp-core-def
  Print.of-ocl-def-transition-def


  Print.of-ocl-list-attr.simps
  Print.of-ocl-data-shallow.simps

end
```

H.3 Instantiating the Printer for META

```
theory Printer-META
imports Parser-META
        ../../compiler-generic/meta-isabelle/Printer-Isabelle
        Printer-UML-extended
begin

context Print
begin

definition of_env-section env =
 (if D-output-disable-thy env then
    λ-. ⟨⟩
  else
    of-section env)

definition of_env-semi--theory env =
          (λ Theory-section section-title ⇒ of_env-section env section-title
           | x ⇒ of-semi--theory env x)

definition' ⟨of_env-semi--theories env =
```

```
 (λ Theories-one t ⇒ of_env-semi--theory env t
  | Theories-locale data l ⇒
      ‹locale %s =
%s
begin
%s
end›   (To-string (HolThyLocale-name data))
        (String-concat-map
           ‹
›
          (λ (l-fix, o-assum).
              ‹%s%s› (String-concat-map ‹
› (λ(e, ty). ‹fixes %s :: %s› (of-semi--term e) (of-semi--typ ty)) l-fix)
                                (case o-assum of None ⇒ ‹›
                                               | Some (name, e) ⇒ ‹
assumes %s: %s› (To-string name) (of-semi--term e)))
          (HolThyLocale-header data))
        (String-concat-map ‹

› (String-concat-map ‹

› (of_env-semi--theory env)) l))›


definition of-floor = (λ Floor1 ⇒ ‹› | Floor2 ⇒ ‹[shallow]› | Floor3 ⇒ ‹[shallow-shallow]›)

definition of-all-meta-embedding env =
 (λ META-ctxt floor ctxt ⇒ of-ocl-ctxt env (of-floor floor) ctxt
  | META-instance i ⇒ of-ocl-instance env i
  | META-def-state floor s ⇒ of-ocl-def-state env (of-floor floor) s
  | META-def-transition floor p ⇒ of-ocl-def-transition env (of-floor floor) p)

definition of-boot-generation-syntax - = (λ Boot-generation-syntax mode ⇒
  ‹generation-syntax [ shallow%s ]›
    (let f = ‹ (generation-semantics [ %s ])› in
     case mode of Gen-only-design ⇒ f ‹design›
                | Gen-only-analysis ⇒ f ‹analysis›
                | Gen-default ⇒ ‹›))

declare[[cartouche-type' = abr-string]]

definition of-boot-setup-env env = (λ Boot-setup-env e ⇒
  of-setup
    env
    (Setup
      (SML.app
        ‹Generation-mode.update-compiler-config›
        [ SML.app
            ‹K›
            [ SML-let-open
                ‹META›
                ((* Instead of using
                    (*sml-of-compiler-env-config SML-apply (λx. SML-basic [x]) e*)
                    the following allows to 'automatically' return an uncurried expression: *)
                 SML-basic [sml-of-compiler-env-config sml-apply id e])]])))

declare[[cartouche-type' = fun_printf]]

definition of-all-meta env = (λ
    META-semi--theories thy ⇒ of_env-semi--theories env thy
  | META-boot-generation-syntax generation-syntax ⇒ of-boot-generation-syntax env generation-syntax
  | META-boot-setup-env setup-env ⇒ of-boot-setup-env env setup-env
  | META-all-meta-embedding all-meta-embedding ⇒ of-all-meta-embedding env all-meta-embedding)

definition of-all-meta-lists env l-thy =
  (let (th-beg, th-end) = case D-output-header-thy env of None ⇒ ([], [])
   | Some (name, fic-import, fic-import-boot) ⇒
      ( [ ‹theory %s imports %s begin›
           (To-string name)
           (of-semi--term (term-binop ‹'' ''›
                            (L.map Term-string
                                 (fic-import @@@@ (if D-output-header-force env
```

```
                                                    | D-output-auto-bootstrap env then
                                                      [fic-import-boot]
                                                    else
                                                      [])))) ]
      , [ ‹›, ‹end› ]) in
  L.flatten
      [ th-beg
      , L.flatten (fst (L.mapM (λ(msg, l) (i, cpt).
          let (l-thy, lg) = L.mapM (λl n. (of-all-meta env l, Succ n)) l 0 in
          (( ‹›
           # ‹%s(* %d ********************************** %d + %d *)%s›
              (To-string (if compiler-env-config.more env then ‹'''› else °char-escape°))
              (To-nat (Succ i))
              (To-nat cpt)
              (To-nat lg)
              (case msg of None ⇒ ‹› | Some msg ⇒ ‹  (* term %s *)› (To-string msg))
           # l-thy), Succ i, cpt + lg)) l-thy (D-output-position env)))
      , th-end ])
end

lemmas [code] =

  Print.of_env-section-def
  Print.of_env-semi--theory-def
  Print.of_env-semi--theories-def
  Print.of-floor-def
  Print.of-all-meta-embedding-def
  Print.of-boot-generation-syntax-def
  Print.of-boot-setup-env-def
  Print.of-all-meta-def
  Print.of-all-meta-lists-def


end
```

H.4 Finalizing the Printer

```
theory Printer
imports Core
        meta/Printer-META
begin

definition List-iterM f l =
  List.fold (λx m. bind m (λ () ⇒ f x)) l (return ())

context Print
begin

declare[[cartouche-type' = String.literal]]

definition (write-file0 :: - ⇒ (((- ⇒ String.literal ⇒ -) ⇒ -) ⇒ -) × -) env =
 (let (l-thy, Sys-argv) = compiler-env-config.more env
   ; (is-file, f-output) = case (D-output-header-thy env, Sys-argv)
     of (Some (file-out, -), Some dir) ⇒
          let dir = To-string dir in
          (True, λf. bind (Sys-is-directory2 dir) (λ Sys-is-directory2-dir.
                    out-file1 f (if Sys-is-directory2-dir then sprint2 ‹%s/%s.thy›' dir (To-string file-out) else dir)))
      | - ⇒ (False, out-stand1)
   ; (env, l) =
         fold-thy'
           comp-env-save-deep
           (λf. f ())
           (λ- -. [])
           (λmsg x acc1 acc2. (acc1, Cons (msg, x) acc2))
           (fst (compiler-env-config.more env))
           (compiler-env-config.truncate env, []) in
  (f-output, of-all-meta-lists (compiler-env-config-more-map (λ-. is-file) env) (rev l)))

definition write-file env =
 (let (f-output, l) = write-file0 env in
  f-output
```

```
    (λfprintf1.
      List-iterM (fprintf1 ‹%s
› )
                 l))
end

definition write-file0 = Print.write-file0 (String.implode o String.to-list) (ToNat integer-of-natural)
definition write-file = Print.write-file (String.implode o String.to-list) (ToNat integer-of-natural)

lemmas [code] =

  Print.write-file0-def
  Print.write-file-def
```

H.5 Miscellaneous: Garbage Collection of Notations

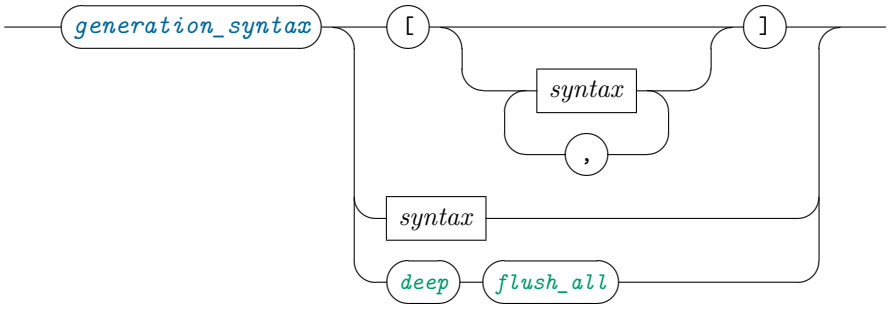
```
no-type-notation natural (nat)
no-type-notation abr-string (string)

end
```

APPENDIX I

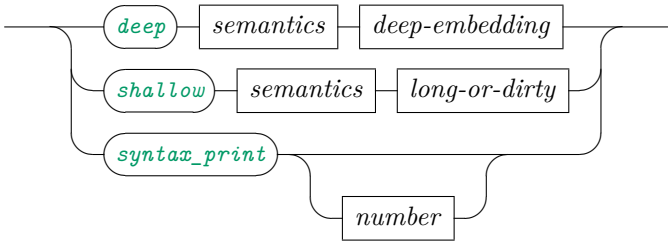
HOL-OCL 2.0: Syntax Diagrams of Commands

I.1 Main Setup of Meta Commands

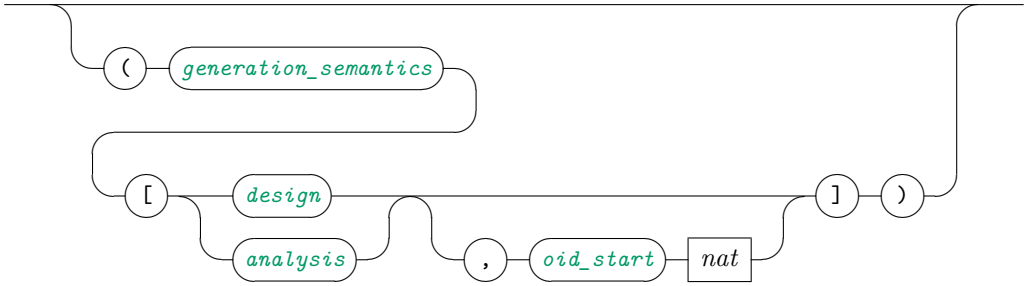
generation-syntax : *theory* → *theory*



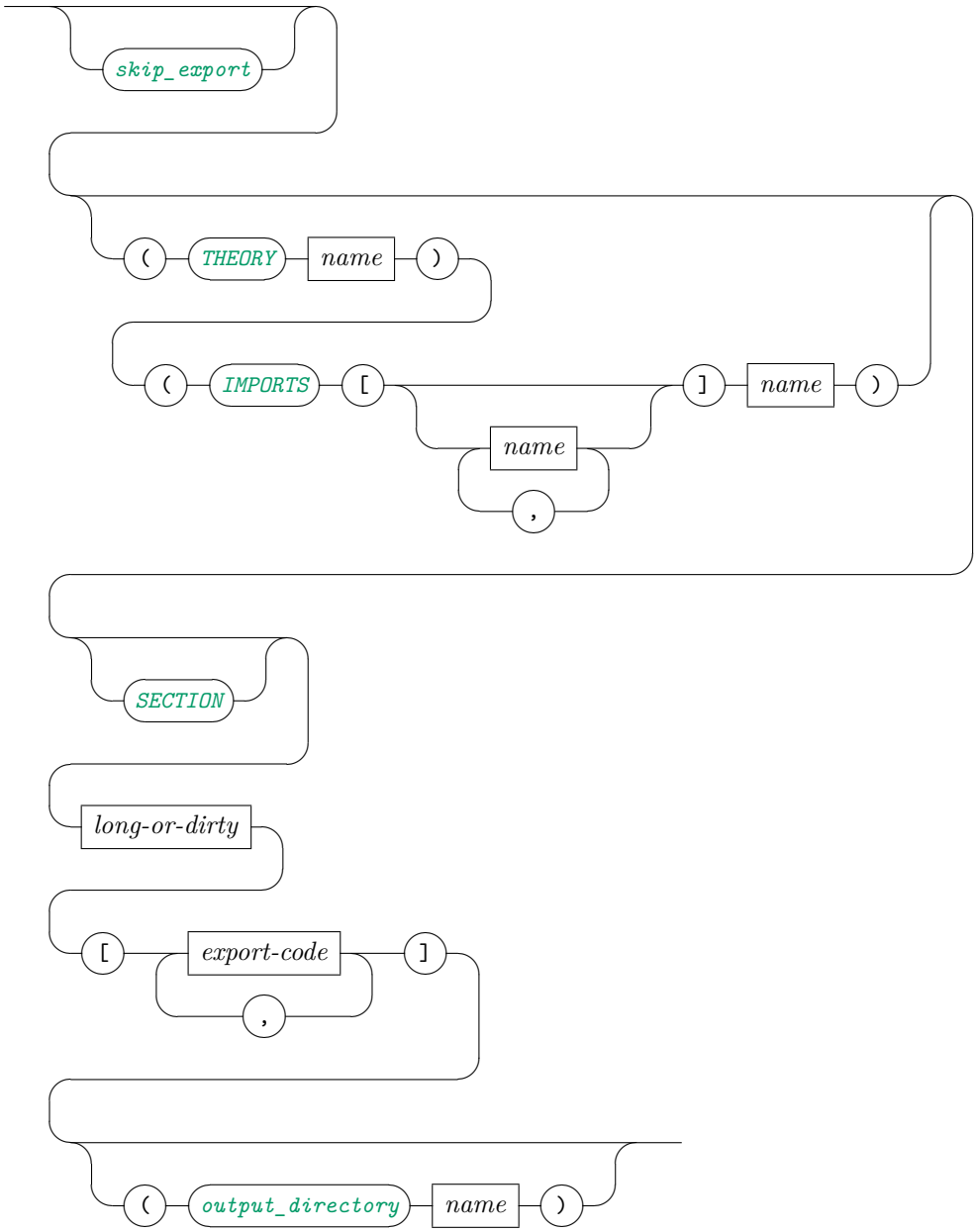
syntax



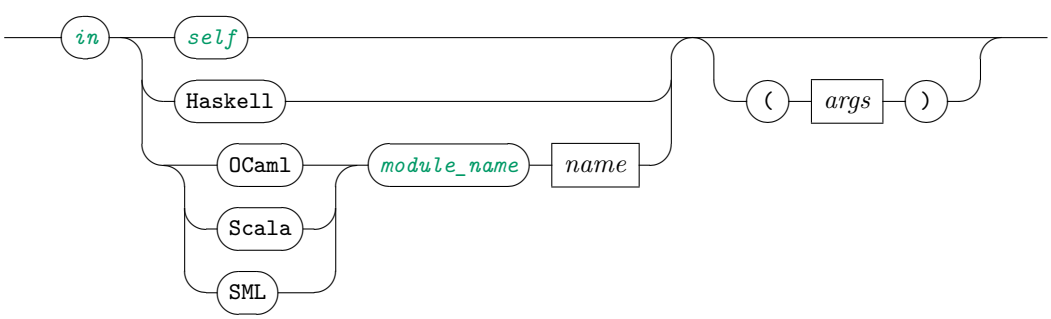
semantics



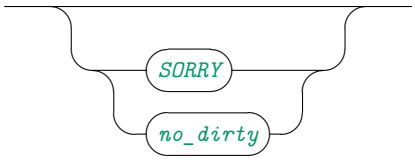
deep-embedding



export-code



long-or-dirty



generation-syntax sets the behavior of all incoming meta-commands. By default, without firstly writing generation-syntax, meta-commands will only print in output what they have parsed, this is similar as giving to generation-syntax a non-empty list having only syntax-print as elements (on the other hand, nothing is printed when an empty list is received). Additionally syntax-print can be followed by an integer indicating the printing depth in output, similar as declaring *ML-print-depth* with an integer, but the global option syntax-print is restricted to meta-commands. Besides the printing of syntaxes, several options are provided to further analyze the semantics of languages being embedded, and tell if their evaluation should occur immediately using the shallow mode, or to only display what would have been evaluated using the deep mode (i.e., to only show the generated Isabelle content in the output window).

Since several occurrences of deep, shallow or syntax-print can appear in the parameterizing list, for each meta-command the overall evaluation respects the order of events given in the list (from head to tail). At the time of writing, it is only possible to evaluate this list sequentially: the execution stops as soon as one first error is raised, thus ignoring remaining events.

generation-syntax deep flush-all performs as side effect the writing of all the generated Isabelle contents to the hard disk (all at the calling time), by iterating the saving for each deep mode in the list. In particular, this is only effective if there is at least one deep mode earlier declared.

As a side note, target languages for the deep mode currently supported are: Haskell, OCaml, Scala and SML. So in principle, all these targets generate the same Isabelle content and exit correctly. However, depending on the intended use, exporting with some targets may be more appropriate than other targets:

- For efficiency reasons, the meta-compiler has implemented a particular optimization for accelerating the process of evaluating incoming meta-commands. By default in Haskell and OCaml, the meta-compiler (at HOL side) is exported only once, during the generation-syntax step. Then all incoming meta-commands are considered as arguments sent to the exported meta-compiler. As a compositionality aspect, these arguments are compiled then linked together with the (already compiled) meta-compiler, but this implies the use of one call of *unsafeCoerce* in Haskell and one *Obj.magic* statement in OCaml (otherwise another solution would be to extract the meta-compiler as a functor). Similar optimizations are not yet implemented for Scala and are only half-implemented for the SML target (which basically performs a step of marshalling to string in Isabelle/ML).

- For safety reasons, it simply suffices to extract all the meta-compiler together with the respective arguments in front of each incoming meta-commands everytime, then the overall needs to be newly compiled everytime. This is the current implemented behavior for Scala. For Haskell, OCaml and SML, it was also the default behavior in a prototyping version of the compiler, as a consequence one can restore that functionality for future versions.

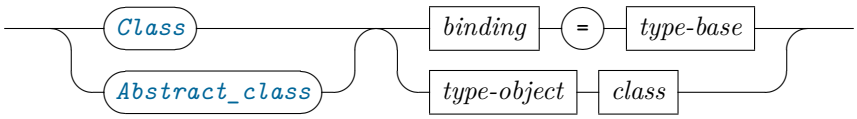
The keyword self is another option to call the own reflected meta-compiler, and execute the full generation without leaving the own Isabelle process being executed.

Concerning the semantics of generated contents, if lemmas and proofs are generated, SORRY allows to explicitly skip the evaluation of all proofs, irrespective of the presence of sorry or not in generated proofs. In any cases, the semantics of sorry has not been overloaded, e.g., red background may appear as usual.

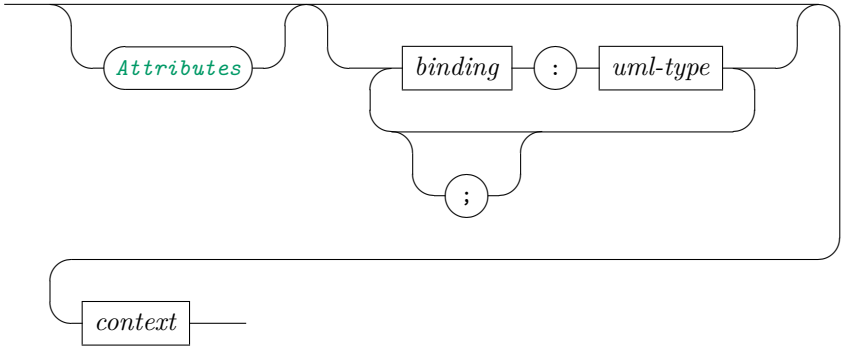
Finally generation-semantics is a container for specifying various options for varying the semantics of languages being embedded. For example, design and analysis are two options for specifying how the modelling of objects will be represented in the Toy Language. Similarly, this would be a typical place for options like *eager* or *lazy* for choosing how the evaluation should happen...

I.2 All Meta Commands of UML/OCL

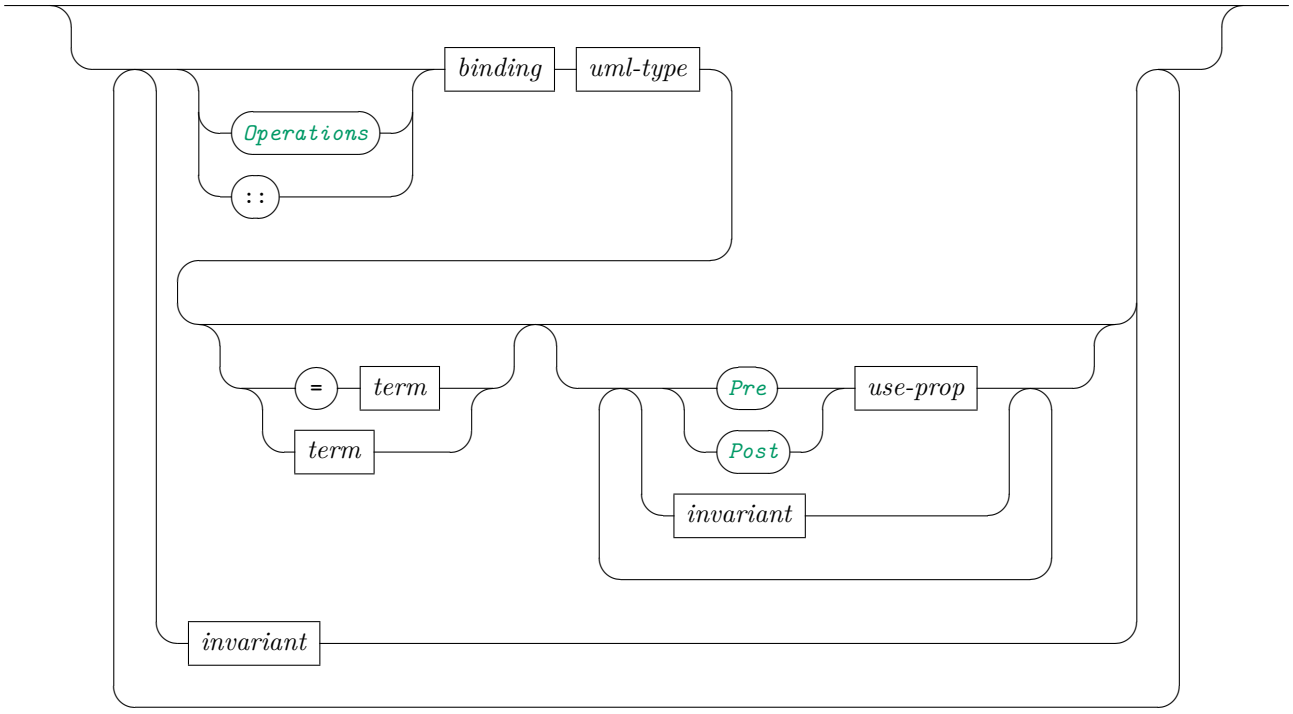
Class : *theory* → *theory*
Abstract-class : *theory* → *theory*



class

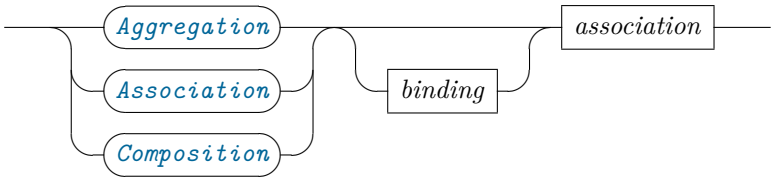


context

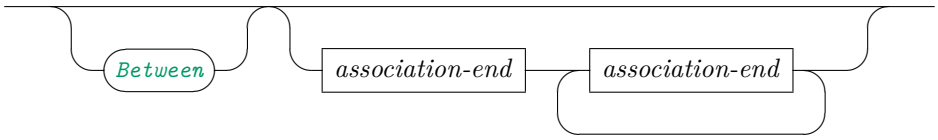


invariant

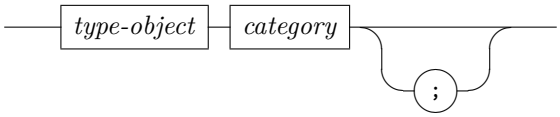
Aggregation : *theory* → *theory*
Association : *theory* → *theory*
Composition : *theory* → *theory*



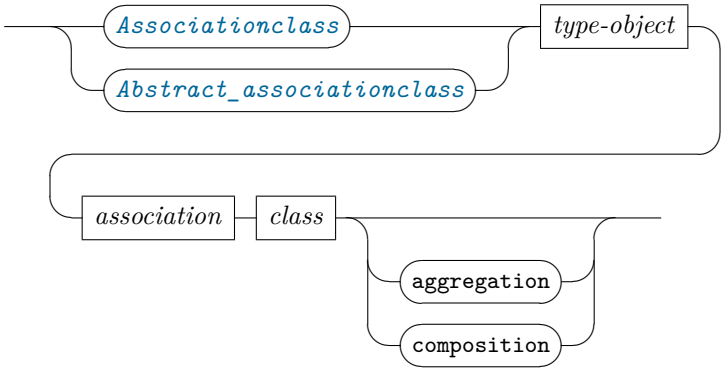
association



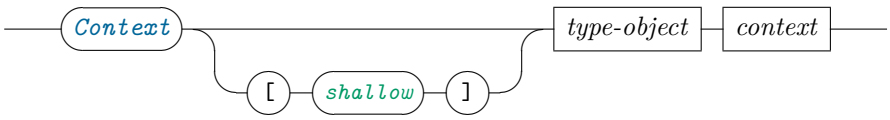
association-end



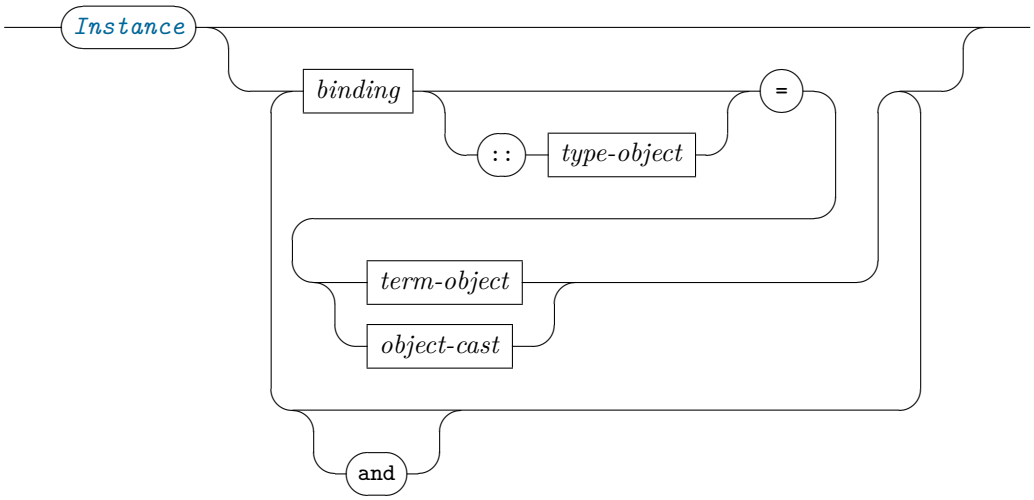
Associationclass : *theory* → *theory*
Abstract-associationclass : *theory* → *theory*



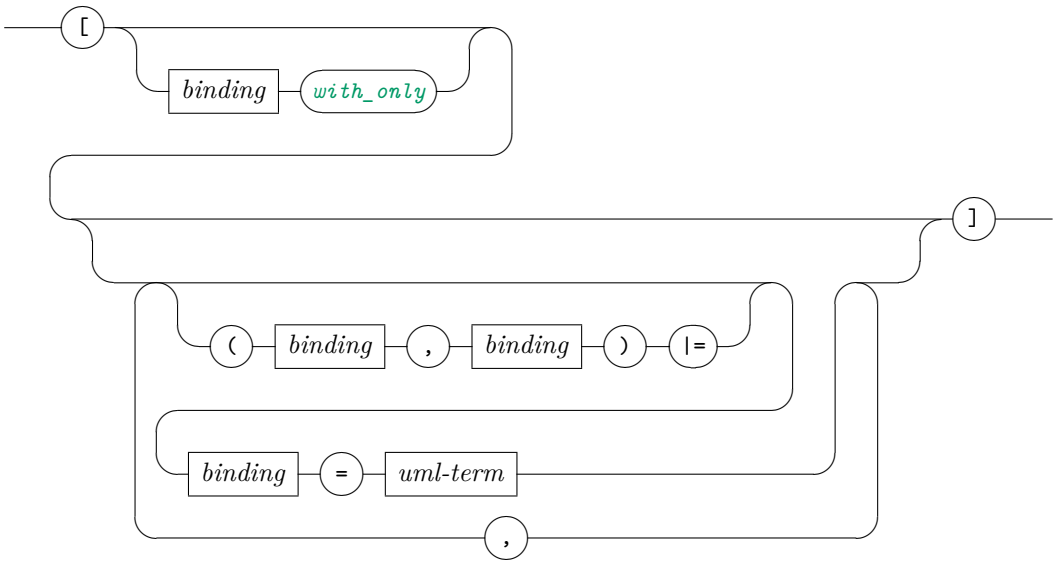
Context : *theory* → *theory*



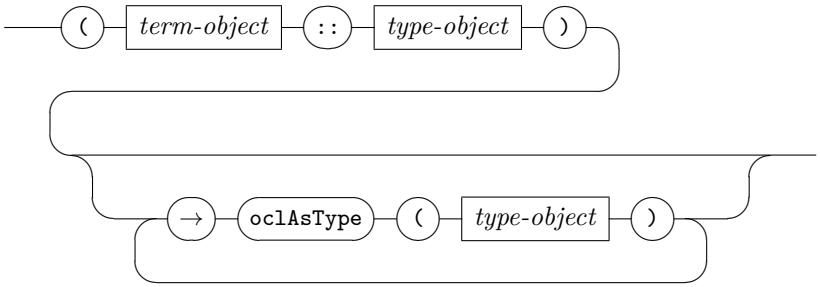
Instance : *theory* → *theory*



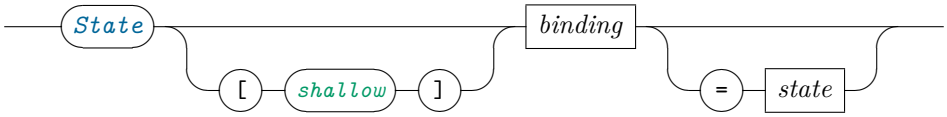
term-object



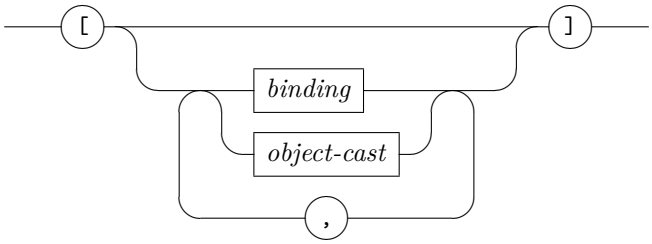
object-cast



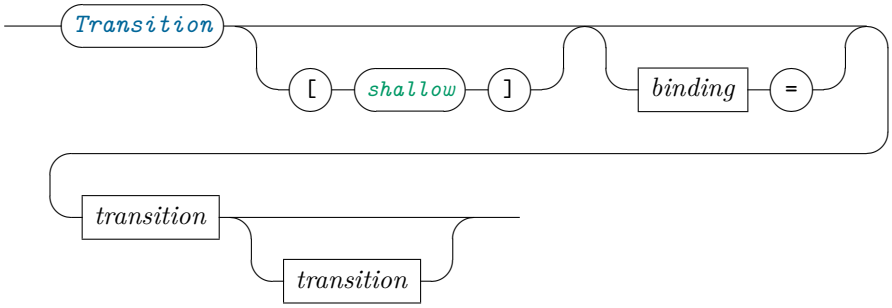
State : *theory* → *theory*



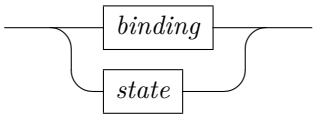
state



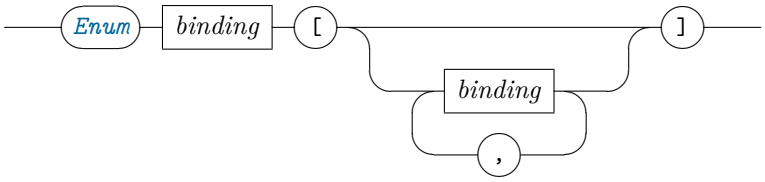
Transition : *theory* → *theory*



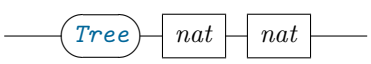
transition



Enum : *theory* → *theory*

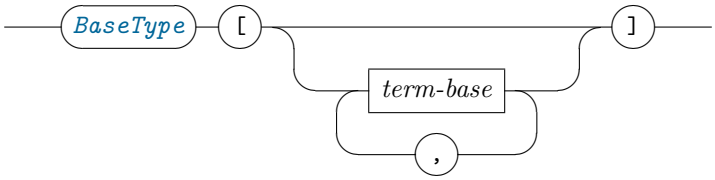


Tree : *theory* → *theory*



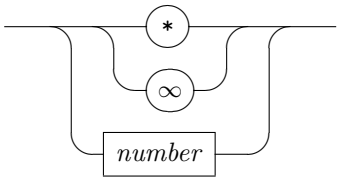
Miscellaneous

BaseType : *theory* → *theory*

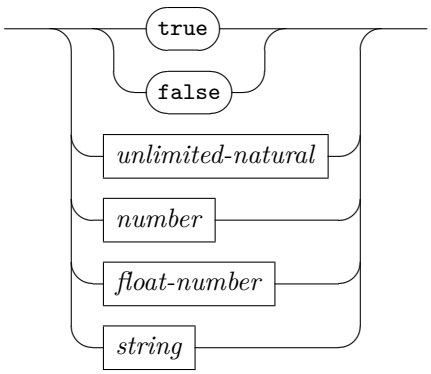


I.3 UML/OCL: Type System

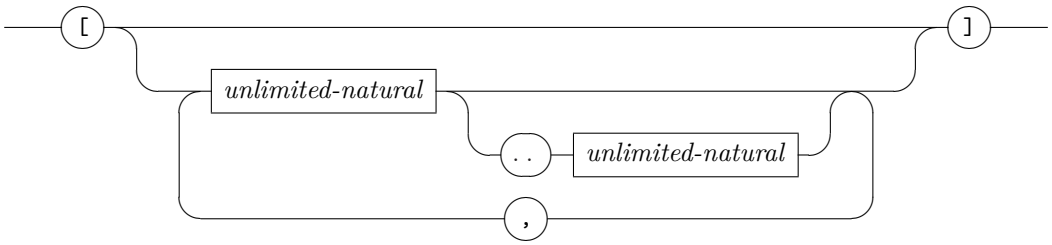
unlimited-natural



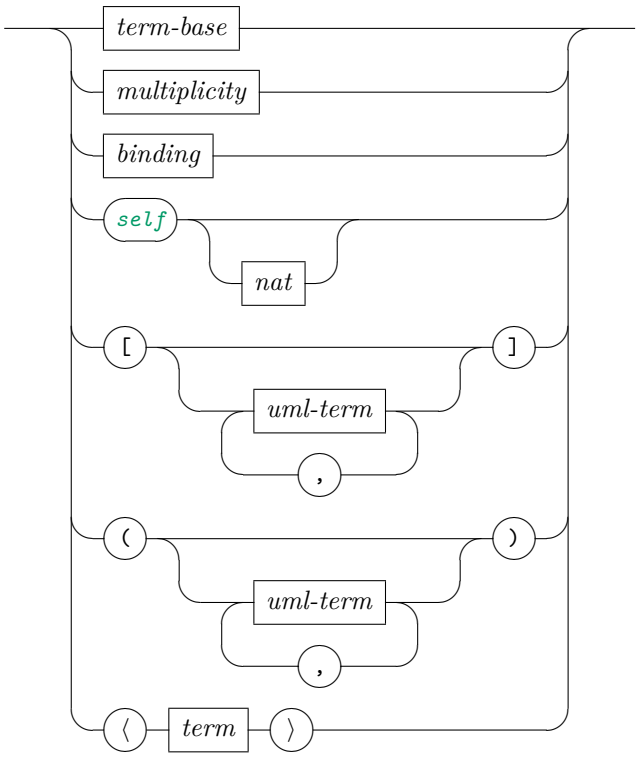
term-base



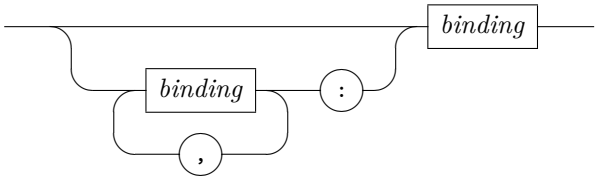
multiplicity



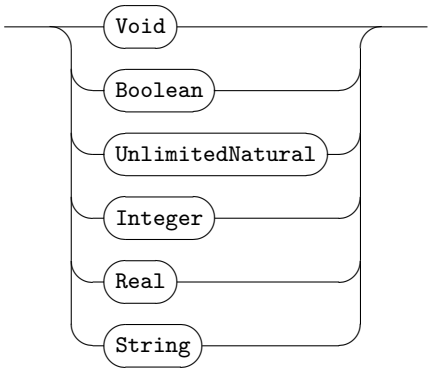
uml-term



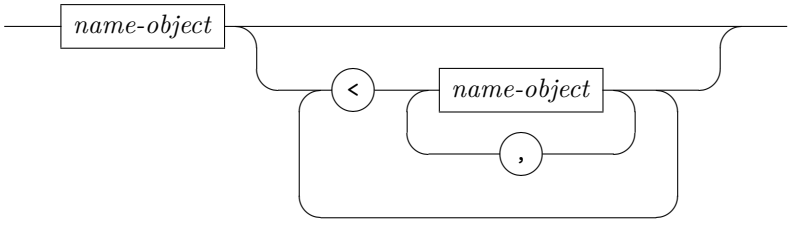
name-object



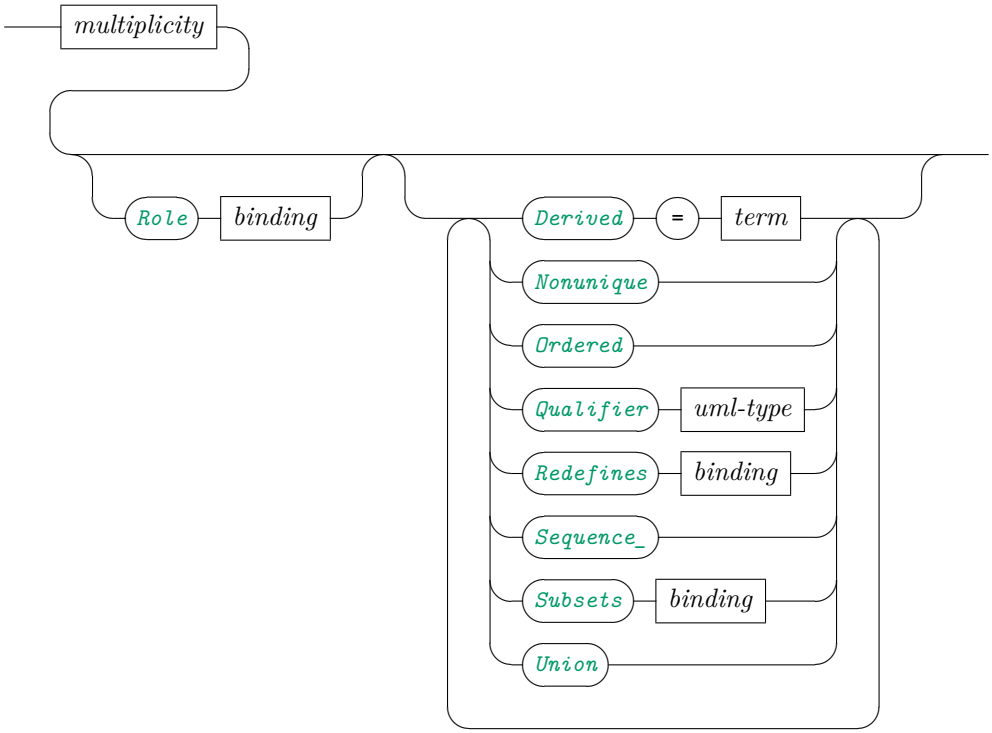
type-base



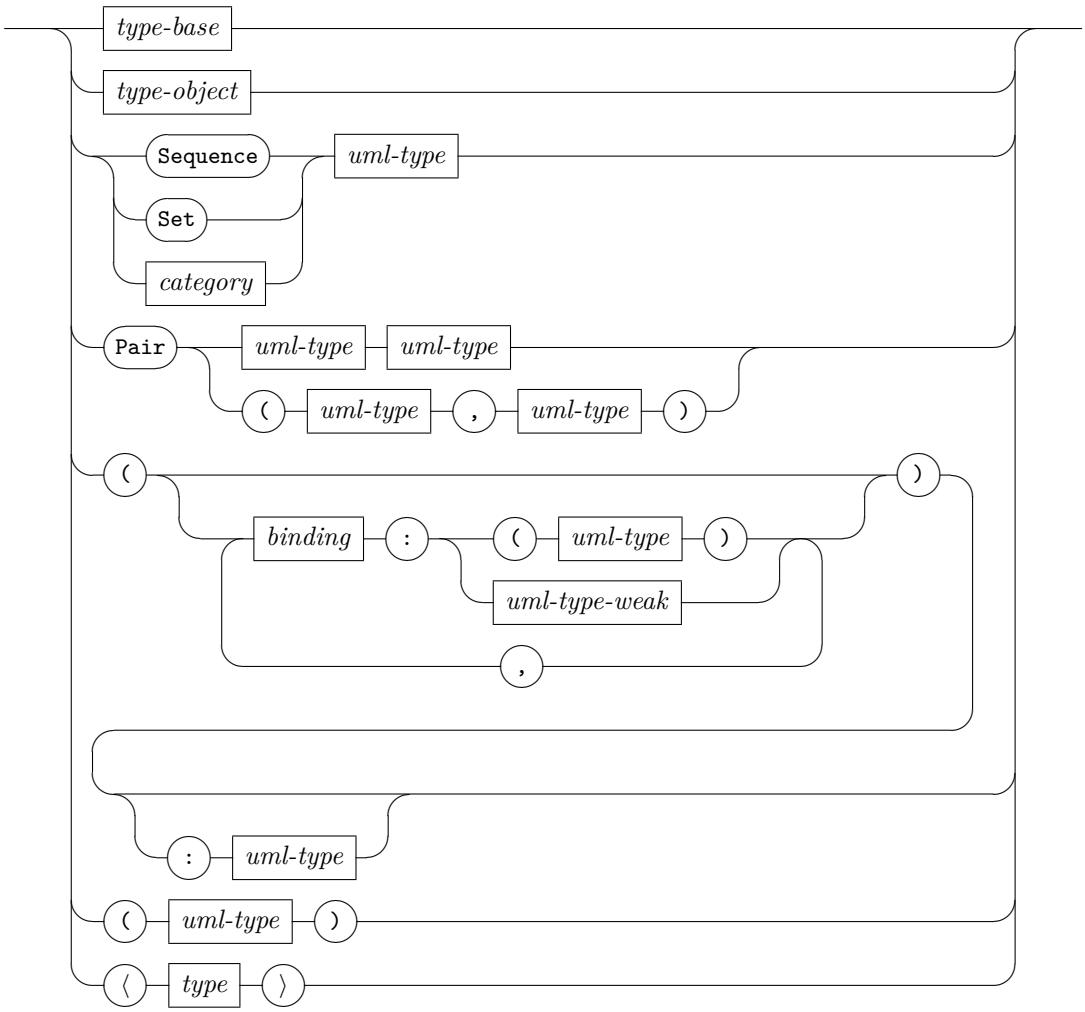
type-object



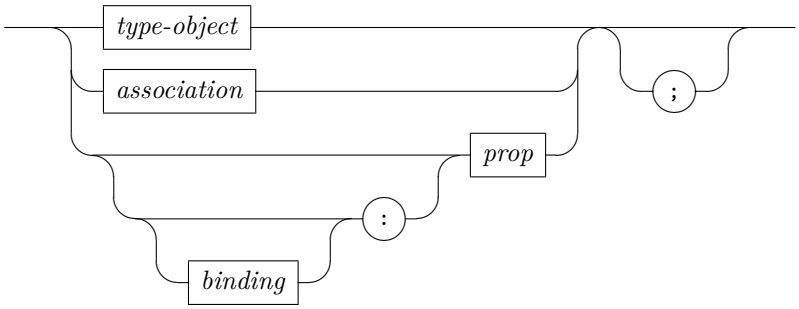
category


multiplicity
Role
binding
Derived
=
term
Nonunique
Ordered
Qualifier
uml-type
Redefines
binding
Sequence_
Subsets
binding
Union

uml-type



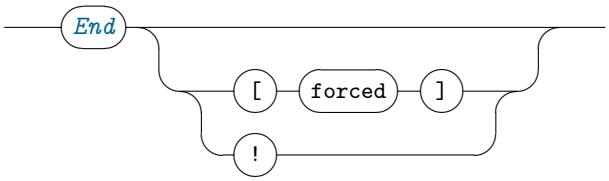
use-prop



uml_type_weak is like uml_type except that type_object can not contain quantified names.

I.4 UML/OCL: Lazy Identity Combinator

End : *theory* → *theory*


End
[
forced
]
!

APPENDIX J

HOL-OCL 2.0: Grammar of Featherweight OCL

This chapter lists the productions of the priority grammar of Featherweight OCL (by not considering the productions initially brought by the theory "Transcendental" from Figure D.2). This chapter has been generated from the output of the command print_syntax [Wen16b] (when that command is called at the end of the theory "UML_Main" from Figure D.2).

(*any* ^{1}) *prop'* ^{1}	(none)	
(*any* ^{1}) *logic* ^{1}	(none)	
(*args*^{1000}) *any*^{0} , *args*^{0}	*args*	
(*args* ^{1}) *any* ^{1}	(none)	
(*cartouche-position*^{1000}) *cartouche*	*position*	
(*id-position*^{1000}) *id*	*position*	
(*idt*^{1000}) (*idt*^{0})	(none)	
(*idt*^{0}) - :: *type*^{0}	*idtypdummy*	
(*idt*^{1000}) -	*idtdummy*	
(*idt*^{0}) *id-position*^{0} :: *type*^{0}	*idtyp*	
(*idt* ^{1}) *id-position* ^{1}	(none)	
(*logic*^{1000}) *op* $\triangleq$	*UML-Logic.StrongEq*	*const*
(*logic*^{1000}) *op* $\triangleq_{pre}$	$UML\text{-}Logic.StrongEq_{pre}$	*const*
(*logic*^{1000}) *op* $\triangleq_{post}$	$UML\text{-}Logic.StrongEq_{post}$	*const*
(*logic*^{1000}) *op* *and*	*UML-Logic.OclAnd*	*const*
(*logic*^{1000}) *op* *or*	*UML-Logic.OclOr*	*const*
(*logic*^{1000}) *op* *implies*	*UML-Logic.OclImplies*	*const*
(*logic*^{1000}) *op* $\doteq$	*UML-Logic.StrictRefEq*	*const*
(*logic*^{1000}) *op* <>	*notequal*	
(*logic*^{1000}) *op* $+_{int}$	$UML\text{-}Integer.OclAdd_{Integer}$	*const*
(*logic*^{1000}) *op* $-_{int}$	$UML\text{-}Integer.OclMinus_{Integer}$	*const*
(*logic*^{1000}) *op* $*_{int}$	$UML\text{-}Integer.OclMult_{Integer}$	*const*

$(logic^{1000})$ op div_{int} — $UML\text{-}Integer.OclDivision_{Integer}$ const

$(logic^{1000})$ op mod_{int} — $UML\text{-}Integer.OclModulus_{Integer}$ const

$(logic^{1000})$ op $<_{int}$ — $UML\text{-}Integer.OclLess_{Integer}$ const

$(logic^{1000})$ op $\leq_{int}$ — $UML\text{-}Integer.OclLe_{Integer}$ const

$(logic^{1000})$ op $+_{string}$ — $UML\text{-}String.OclAdd_{String}$ const

$(logic^{1000})$ op $+_{real}$ — $UML\text{-}Real.OclAdd_{Real}$ const

$(logic^{1000})$ op $-_{real}$ — $UML\text{-}Real.OclMinus_{Real}$ const

$(logic^{1000})$ op $*_{real}$ — $UML\text{-}Real.OclMult_{Real}$ const

$(logic^{1000})$ op div_{real} — $UML\text{-}Real.OclDivision_{Real}$ const

$(logic^{1000})$ op mod_{real} — $UML\text{-}Real.OclModulus_{Real}$ const

$(logic^{1000})$ op $<_{real}$ — $UML\text{-}Real.OclLess_{Real}$ const

$(logic^{1000})$ op $\leq_{real}$ — $UML\text{-}Real.OclLe_{Real}$ const

$(logic^{1000})$ op $\cong$ — $UML\text{-}Bag.ApproxEq$ const

$(logic^{50})$ if $logic^{10}$ $then$ $logic^{10}$ $else$ $logic^{10}$ $endif$ — $UML\text{-}Logic.OclIf$ const

$(logic^{1000})$ $\lceil$ $logic^{0}$ $\rceil$ — $UML\text{-}Types.drop$ const

$(logic^{1000})$ $I[\![$ any^{0} $]\!]$ — $UML\text{-}Types.Sem$ const

$(logic^{100})$ v $logic^{100}$ — $UML\text{-}Logic.valid$ const

$(logic^{100})$ δ $logic^{100}$ — $UML\text{-}Logic.defined$ const

$(logic^{1000})$ not — $UML\text{-}Logic.OclNot$ const

$(logic^{1000})$ $\bot$ — $UML\text{-}Types.bot\text{-}class.bot$ const

$(logic^{1000})$ $\bot$ — $Option.option.None$ const

$(logic^{1000})$ $Pair\{$ $logic^{0}$ $,$ $logic^{0}$ $\}$ — $UML\text{-}Pair.OclPair$ const

$(logic^{1000})$ **0** — $UML\text{-}Integer.OclInt0$ const

$(logic^{1000})$ **1** — $UML\text{-}Integer.OclInt1$ const

$(logic^{1000})$ **2** — $UML\text{-}Integer.OclInt2$ const

$(logic^{1000})$ **3** — $UML\text{-}Integer.OclInt3$ const

$(logic^{1000})$ **4** — $UML\text{-}Integer.OclInt4$ const

$(logic^{1000})$ **5** — $UML\text{-}Integer.OclInt5$ const

$(logic^{1000})$ **6** — $UML\text{-}Integer.OclInt6$ const

$(logic^{1000})$ **7** — $UML\text{-}Integer.OclInt7$ const

$(logic^{1000})$ **8** — $UML\text{-}Integer.OclInt8$ const

$(logic^{1000})$ **9** — $UML\text{-}Integer.OclInt9$ const

$(logic^{1000})$ **10** — $UML\text{-}Integer.OclInt10$ const

(*logic*[1000]) a — *UML-String.OclStringa* const

(*logic*[1000]) b — *UML-String.OclStringb* const

(*logic*[1000]) c — *UML-String.OclStringc* const

(*logic*[1000]) **0.0** — *UML-Real.OclReal0* const

(*logic*[1000]) **1.0** — *UML-Real.OclReal1* const

(*logic*[1000]) **2.0** — *UML-Real.OclReal2* const

(*logic*[1000]) **3.0** — *UML-Real.OclReal3* const

(*logic*[1000]) **4.0** — *UML-Real.OclReal4* const

(*logic*[1000]) **5.0** — *UML-Real.OclReal5* const

(*logic*[1000]) **6.0** — *UML-Real.OclReal6* const

(*logic*[1000]) **7.0** — *UML-Real.OclReal7* const

(*logic*[1000]) **8.0** — *UML-Real.OclReal8* const

(*logic*[1000]) **9.0** — *UML-Real.OclReal9* const

(*logic*[1000]) **10.0** — *UML-Real.OclReal10* const

(*logic*[1000]) π — *UML-Real.OclRealpi* const

(*logic*[1000]) *Bag*{} — *UML-Bag.mtBag* const

(*logic*[1000]) *Bag*{ *args*[0] } — *OclFinbag*

(*logic*[1000]) *Set*{} — *UML-Set.mtSet* const

(*logic*[1000]) *Set*{ *args*[0] } — *OclFinset*

(*logic*[1000]) *Sequence*{} — *UML-Sequence.mtSequence* const

(*logic*[1000]) *Sequence*{ *args*[0] } — *OclFinsequence*

(*logic*[1000]) *cartouche-position*[0] — *cartouche-oclstring*

(*logic*[1000]) -′ — *ocl-denotation*

(*logic*[1000]) *logic*[0] $->asBag_{Pair}()$ — *UML-Library.OclAsBag*$_{Pair}$ const

(*logic*[1000]) *logic*[0] $->asBag_{Set}()$ — *UML-Library.OclAsBag*$_{Set}$ const

(*logic*[1000]) *logic*[0] $->asBag_{Seq}()$ — *UML-Library.OclAsBag*$_{Seq}$ const

(*logic*[1000]) *logic*[0] $->asSequence_{Pair}()$ — *UML-Library.OclAsSeq*$_{Pair}$ const

(*logic*[1000]) *logic*[0] $->asSequence_{Bag}()$ — *UML-Library.OclAsSeq*$_{Bag}$ const

(*logic*[1000]) *logic*[0] $->asSequence_{Set}()$ — *UML-Library.OclAsSeq*$_{Set}$ const

(*logic*[1000]) *logic*[0] $->asSet_{Bag}()$ — *UML-Library.OclAsSet*$_{Bag}$ const

(*logic*[1000]) *logic*[0] $->asSet_{Pair}()$ — *UML-Library.OclAsSet*$_{Pair}$ const

(*logic*[1000]) *logic*[0] $->asSet_{Seq}()$ — *UML-Library.OclAsSet*$_{Seq}$ const

(*logic*[1000]) *logic*[0] $->asPair_{Bag}()$ — *UML-Library.OclAsPair*$_{Bag}$ const

$(logic^{1000})\ logic^{0}\ \mathbf{->asPair_{Set}()}$ — UML-Library.OclAsPair$_{Set}$ const

$(logic^{1000})\ logic^{0}\ \mathbf{->asPair_{Seq}()}$ — UML-Library.OclAsPair$_{Seq}$ const

$(logic^{1000})\ logic^{0}\ \mathbf{->oclAsType_{Int}(Real)}$ — UML-Library.OclAsReal$_{Int}$ const

$(logic^{1000})\ logic^{0}\ \mathbf{->oclAsType_{Real}(Integer)}$ — UML-Library.OclAsInteger$_{Real}$ const

$(logic^{1000})\ logic^{0}\ \mathbf{->oclAsType_{Real}(Boolean)}$ — UML-Library.OclAsBoolean$_{Real}$ const

$(logic^{1000})\ logic^{0}\ \mathbf{->oclAsType_{Int}(Boolean)}$ — UML-Library.OclAsBoolean$_{Int}$ const

$(logic^{1000})\ logic^{0}\ \mathbf{->sum_{Seq}()}$ — UML-Sequence.OclSum const

$(logic^{1000})\ logic^{0}\ \mathbf{->count_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclCount const

$(logic^{1000})\ logic^{0}\ \mathbf{->any_{Seq}()}$ — UML-Sequence.OclANY const

$(logic^{1000})\ logic^{0}\ \mathbf{->notEmpty_{Seq}()}$ — UML-Sequence.OclNotEmpty const

$(logic^{1000})\ logic^{0}\ \mathbf{->isEmpty_{Seq}()}$ — UML-Sequence.OclIsEmpty const

$(logic^{1000})\ logic^{0}\ \mathbf{->size_{Seq}()}$ — UML-Sequence.OclSize const

$(logic^{1000})\ logic^{0}\ \mathbf{->select_{Seq}(}\ id\ \mathbf{|}\ logic^{0}\ \mathbf{)}$ — OclSelectSeq

$(logic^{1000})\ logic^{0}\ \mathbf{->collect_{Seq}(}\ id\ \mathbf{|}\ logic^{0}\ \mathbf{)}$ — OclCollectSeq

$(logic^{1000})\ logic^{0}\ \mathbf{->exists_{Seq}(}\ id\ \mathbf{|}\ logic^{0}\ \mathbf{)}$ — OclExistSeq

$(logic^{1000})\ logic^{0}\ \mathbf{->forAll_{Seq}(}\ id\ \mathbf{|}\ logic^{0}\ \mathbf{)}$ — OclForallSeq

$(logic^{1000})\ logic^{0}\ \mathbf{->iterate_{Seq}(}\ idt^{0}\ \mathbf{;}\ idt^{0}\ \mathbf{=}\ any^{0}\ \mathbf{|}\ any^{0}\ \mathbf{)}$ — OclIterateSeq

$(logic^{1000})\ logic^{0}\ \mathbf{->last_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclLast const

$(logic^{1000})\ logic^{0}\ \mathbf{->first_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclFirst const

$(logic^{1000})\ logic^{0}\ \mathbf{->at_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclAt const

$(logic^{1000})\ logic^{0}\ \mathbf{->union_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclUnion const

$(logic^{1000})\ logic^{0}\ \mathbf{->append_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclAppend const

$(logic^{1000})\ logic^{0}\ \mathbf{->excluding_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclExcluding const

$(logic^{1000})\ logic^{0}\ \mathbf{->including_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclIncluding const

$(logic^{1000})\ logic^{0}\ \mathbf{->prepend_{Seq}(}\ logic^{0}\ \mathbf{)}$ — UML-Sequence.OclPrepend const

$(logic^{1000})\ logic^{0}\ \mathbf{->sum_{Set}()}$ — UML-Set.OclSum const

$(logic^{1000})\ logic^{0}\ \mathbf{->count_{Set}(}\ logic^{0}\ \mathbf{)}$ — UML-Set.OclCount const

$(logic^{1000})\ logic^{0}\ \mathbf{->intersection_{Set}(}\ logic^{0}\ \mathbf{)}$ — UML-Set.OclIntersection const

$(logic^{1000})\ logic^{0}\ \mathbf{->union_{Set}(}\ logic^{0}\ \mathbf{)}$ — UML-Set.OclUnion const

$(logic^{1000})\ logic^{0}\ \mathbf{->excludesAll_{Set}(}\ logic^{0}\ \mathbf{)}$ — UML-Set.OclExcludesAll const

$(logic^{1000})\ logic^{0}\ \mathbf{->includesAll_{Set}(}\ logic^{0}\ \mathbf{)}$ — UML-Set.OclIncludesAll const

$(logic^{1000})\ logic^{0}\ \texttt{->}reject_{Set}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclRejectSet*

$(logic^{1000})\ logic^{0}\ \texttt{->}select_{Set}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclSelectSet*

$(logic^{1000})\ logic^{0}\ \texttt{->}iterate_{Set}(\ idt^{0}\ ;\ idt^{0}\ =\ any^{0}\ |\ any^{0}\ )$ — *OclIterateSet*

$(logic^{1000})\ logic^{0}\ \texttt{->}exists_{Set}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclExistSet*

$(logic^{1000})\ logic^{0}\ \texttt{->}forAll_{Set}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclForallSet*

$(logic^{1000})\ logic^{0}\ \texttt{->}any_{Set}()$ — *UML-Set.OclANY* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}notEmpty_{Set}()$ — *UML-Set.OclNotEmpty* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}isEmpty_{Set}()$ — *UML-Set.OclIsEmpty* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}size_{Set}()$ — *UML-Set.OclSize* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}excludes_{Set}(\ logic^{0}\ )$ — *UML-Set.OclExcludes* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}includes_{Set}(\ logic^{0}\ )$ — *UML-Set.OclIncludes* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}excluding_{Set}(\ logic^{0}\ )$ — *UML-Set.OclExcluding* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}including_{Set}(\ logic^{0}\ )$ — *UML-Set.OclIncluding* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}sum_{Bag}()$ — *UML-Bag.OclSum* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}count_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclCount* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}intersection_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclIntersection* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}union_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclUnion* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}excludesAll_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclExcludesAll* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}includesAll_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclIncludesAll* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}reject_{Bag}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclRejectBag*

$(logic^{1000})\ logic^{0}\ \texttt{->}select_{Bag}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclSelectBag*

$(logic^{1000})\ logic^{0}\ \texttt{->}iterate_{Bag}(\ idt^{0}\ ;\ idt^{0}\ =\ any^{0}\ |\ any^{0}\ )$ — *OclIterateBag*

$(logic^{1000})\ logic^{0}\ \texttt{->}exists_{Bag}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclExistBag*

$(logic^{1000})\ logic^{0}\ \texttt{->}forAll_{Bag}(\ \boxed{id}\ |\ logic^{0}\ )$ — *OclForallBag*

$(logic^{1000})\ logic^{0}\ \texttt{->}any_{Bag}()$ — *UML-Bag.OclANY* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}notEmpty_{Bag}()$ — *UML-Bag.OclNotEmpty* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}isEmpty_{Bag}()$ — *UML-Bag.OclIsEmpty* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}size_{Bag}()$ — *UML-Bag.OclSize* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}excludes_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclExcludes* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}includes_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclIncludes* *const*

$(logic^{1000})\ logic^{0}\ \texttt{->}excluding_{Bag}(\ logic^{0}\ )$ — *UML-Bag.OclExcluding* *const*

$(logic^{1000})$	$logic^{0}$ $->including_{Bag}($ $logic^{0}$ $)$	*UML-Bag.OclIncluding*	*const*
$(logic^{30})$	$logic^{30}$ $\cong$ $logic^{31}$	*UML-Bag.ApproxEq*	*const*
$(logic^{35})$	$logic^{36}$ $\leq_{real}$ $logic^{36}$	$UML\text{-}Real.OclLe_{Real}$	*const*
$(logic^{35})$	$logic^{36}$ $<_{real}$ $logic^{36}$	$UML\text{-}Real.OclLess_{Real}$	*const*
$(logic^{45})$	$logic^{46}$ mod_{real} $logic^{46}$	$UML\text{-}Real.OclModulus_{Real}$	*const*
$(logic^{45})$	$logic^{46}$ div_{real} $logic^{46}$	$UML\text{-}Real.OclDivision_{Real}$	*const*
$(logic^{45})$	$logic^{46}$ $*_{real}$ $logic^{46}$	$UML\text{-}Real.OclMult_{Real}$	*const*
$(logic^{41})$	$logic^{42}$ $-_{real}$ $logic^{42}$	$UML\text{-}Real.OclMinus_{Real}$	*const*
$(logic^{40})$	$logic^{41}$ $+_{real}$ $logic^{41}$	$UML\text{-}Real.OclAdd_{Real}$	*const*
$(logic^{40})$	$logic^{41}$ $+_{string}$ $logic^{41}$	$UML\text{-}String.OclAdd_{String}$	*const*
$(logic^{35})$	$logic^{36}$ $\leq_{int}$ $logic^{36}$	$UML\text{-}Integer.OclLe_{Integer}$	*const*
$(logic^{35})$	$logic^{36}$ $<_{int}$ $logic^{36}$	$UML\text{-}Integer.OclLess_{Integer}$	*const*
$(logic^{45})$	$logic^{46}$ mod_{int} $logic^{46}$	$UML\text{-}Integer.OclModulus_{Integer}$	*const*
$(logic^{45})$	$logic^{46}$ div_{int} $logic^{46}$	$UML\text{-}Integer.OclDivision_{Integer}$	*const*
$(logic^{45})$	$logic^{46}$ $*_{int}$ $logic^{46}$	$UML\text{-}Integer.OclMult_{Integer}$	*const*
$(logic^{41})$	$logic^{42}$ $-_{int}$ $logic^{42}$	$UML\text{-}Integer.OclMinus_{Integer}$	*const*
$(logic^{40})$	$logic^{41}$ $+_{int}$ $logic^{41}$	$UML\text{-}Integer.OclAdd_{Integer}$	*const*
$(logic^{1000})$	$logic^{0}$ $.Second()$	*UML-Pair.OclSecond*	*const*
$(logic^{1000})$	$logic^{0}$ $.First()$	*UML-Pair.OclFirst*	*const*
$(logic^{40})$	$logic^{41}$ $<>$ $logic^{41}$	*notequal*	
$(logic^{30})$	$logic^{30}$ $\doteq$ $logic^{31}$	*UML-Logic.StrictRefEq*	*const*
$(logic^{50})$	$logic^{0}$ $\models_{post}$ $logic^{0}$	*UML-Logic.OclValid-at-post*	*const*
$(logic^{50})$	$logic^{0}$ $\models_{pre}$ $logic^{0}$	*UML-Logic.OclValid-at-pre*	*const*
$(logic^{50})$	$logic^{0}$ $\mid\neq$ $logic^{0}$	*OclNonValid*	
$(logic^{50})$	$logic^{0}$ $\models$ $logic^{0}$	*UML-Logic.OclValid*	*const*
$(logic^{25})$	$logic^{25}$ $implies$ $logic^{26}$	*UML-Logic.OclImplies*	*const*
$(logic^{25})$	$logic^{25}$ or $logic^{26}$	*UML-Logic.OclOr*	*const*
$(logic^{30})$	$logic^{30}$ and $logic^{31}$	*UML-Logic.OclAnd*	*const*
$(logic^{30})$	$logic^{30}$ $\triangleq_{post}$ $logic^{31}$	$UML\text{-}Logic.StrongEq_{post}$	*const*
$(logic^{30})$	$logic^{30}$ $\triangleq_{pre}$ $logic^{31}$	$UML\text{-}Logic.StrongEq_{pre}$	*const*
$(logic^{30})$	$logic^{30}$ $\triangleq$ $logic^{31}$	*UML-Logic.StrongEq*	*const*
$(logic^{1000})$	$logic^{0}$ $.allInstances()$	*UML-State.OclAllInstances-at-post*	*const*

$(logic^{1000})$	$logic^{0}$ $.allInstances@pre()$	$UML\text{-}State.OclAllInstances\text{-}at\text{-}pre$ $const$
$(logic^{1000})$	$logic^{0}$ $.oclIsNew()$	$UML\text{-}State.OclIsNew$ $const$
$(logic^{1000})$	$logic^{0}$ $.oclIsDeleted()$	$UML\text{-}State.OclIsDeleted$ $const$
$(logic^{1000})$	$logic^{0}$ $.oclIsMaintained()$	$UML\text{-}State.OclIsMaintained$ $const$
$(logic^{1000})$	$logic^{0}$ $.oclIsAbsent()$	$UML\text{-}State.OclIsAbsent$ $const$
$(logic^{1000})$	$logic^{0}$ $->oclIsModifiedOnly()$	$UML\text{-}State.OclIsModifiedOnly$ $const$
$(logic^{1000})$	$logic^{0}$ $@pre$ $logic^{0}$	$UML\text{-}State.OclSelf\text{-}at\text{-}pre$ $const$
$(logic^{1000})$	$logic^{0}$ $@post$ $logic^{0}$	$UML\text{-}State.OclSelf\text{-}at\text{-}post$ $const$
$(type^{1000})$	$\langle$ $type^{0}$ $\rangle_{\perp}$	$Option.option$ $type$
$(type^{1000})$	$Pair($ $type^{0}$ $,$ $type^{0}$ $)$	$UML\text{-}Types.Pair_{base}$ $type$
$(type^{1000})$	$Set($ $type^{0}$ $)$	$UML\text{-}Types.Set_{base}$ $type$
$(type^{1000})$	$Bag($ $type^{0}$ $)$	$UML\text{-}Types.Bag_{base}$ $type$
$(type^{1000})$	$Sequence($ $type^{0}$ $)$	$UML\text{-}Types.Sequence_{base}$ $type$

APPENDIX

Defining Isar_HOL syntax “from null”

The possibility to embed an arbitrary language $\mathcal{L}$ in Isabelle depends on the capacity of the proving system to parse new syntax and reconfigure commands, so that most keywords of $\mathcal{L}$ can be represented in the system. In this part, we detail certain flexibility of the framework by particularly presenting how to type the `invalid` and `null` characters in Isabelle/jEdit. As illustration, Figure K.1 is exactly similar as Figure 5.8 except that all commands have been syntactically renamed.

Despite the similarity between “U+0435”[1] and “e” (i. e. “U+0065”[2]), here theory and lemma have not been overloaded using particular equal glyphs like in Section 6.5: datatype was renamed into “theory Scratch3␤␤theory ‹”, and fun renamed into “›␤␤lemma” (where we represent the newline symbol by “␤”). So spaces can occur in the name of commands at any positions. Not only have commands been renamed but green syntactic entities are also flexible (i. e., the portions of code that can be divided inside each green area of Figure 5.9). For example, after theory, imports is usually written before keywords, but it is in the source code where the order is established, and such modifications can be easily experimented with the command ML for example. As another example, the need to write “*end*” at the end of the theory does not depend on the presence or not of the keyword begin at the beginning of the theory: even if the color of this last appears in green in Isabelle/jEdit, “*end*” has been internally defined with `Outer_Syntax.command` in the source of Isabelle. Additionally Figure K.1 also shows that one can replace green keywords by other green keywords: as example where is replaced by assumes, and “|” by shows. Even if the color of “|” is black, it can be considered as a green entity (so do all other entities which are not commands).

Since names of commands can include certain interleaving of white spaces, one possibility to detect the boundaries of a command is to hover with the mouse over a visible part of that command, and observe a lighten continuous region appearing in the background. For instance, by doing so, we can observe “\includegraphics” is a single word with particularly no white spaces around it. After \includegraphics, the two cartouches are not blue: they do not belong to

[1] `http://unicode.org/cldr/utility/character.jsp?a=0435`
[2] `http://unicode.org/cldr/utility/character.jsp?a=0065`

```
theory Scratch3  imports  Scratch2
                 keywords "." :: diag  begin
theory Scratch3

theory ‹LIST = NIL | CONS nat LIST›

lemma    height :: "LIST ⇒ nat"
assumes "height NIL        = 0              "
shows   "height (CONS _ t) = Suc (height t)"

\includegraphics[[ML_source_trace]]‹
     val NIL    = @{code NIL}
     val height = @{code height}
     val _      = height NIL ›‹
     Outer_Syntax.command @{command_keyword "."}
     " reads and prints an arbitrary HOL term"
     (Parse.term >> (Isar_Cmd.print_term o pair [])) ›

"height a + height b = height b + height a"

end
find_theorems
```

Figure K.1: Syntactically renaming commands of Figure 5.8

the syntax of a potential blue command, they are in fact categorized as part of a larger green area.

Since these two cartouches are following each other closely (without apparent white spaces), the number of arguments of \includegraphics is not clear. As it could be attentively noticed, it is sure that \includegraphics does not take four arguments since there is a measurable dot in blue separating "`height` a + `height` b = `height` b + `height` a" with the content occurring before this dot. The current size of the command "." has in fact been visually attained by combining several special effects:

- The glyph of the symbol "·" (i. e. "U+22C5"[3]) has one of the smallest number of non-null pixels among other glyphs.

- The special symbol "⇩" (i. e. "U+21E9"[4]) acts as an operator. It can be applied in front of most symbols, and only at most one time in the current version of Isabelle. As side effect, it attempts to reduce the given typographic text to the subscript level (each symbol of the given text has to be applied one by one because this operator only takes one symbol as argument). In the picture, it is prefixed one time to "·" (without intercalating any white spaces).

[3] `http://unicode.org/cldr/utility/character.jsp?a=22C5`
[4] `http://unicode.org/cldr/utility/character.jsp?a=21E9`

More generally, whereas "⇩" has a proper meaning in Isabelle/jEdit (among other special symbols which can for example portray a juxtaposed letter in superscript or in bold), there are many symbols in Unicode acting as an operator, perhaps independently of the own rendering of the editing software, thus also many symbols affecting the preceding or succeeding symbol across font families. As example, " ̇" (i. e. "U+05B9"[5]) is a suffix operator which draws a dot on the position of the neighbour symbol as follows: "`012_̇_56`". Consequently, this dot can only be perceived if no neighbours are overlapping and accidentally hiding the dot. Variations of "U+05B9" actually exist in numerous forms: symbols drawing a dot on the top, or drawing a dot on the bottom.

Ultimately, this leads the following part to answer why \includegraphics is resembling to a meta-command. As a matter of fact, we precisely focus on variations of symbols depicting the absence of pixels, i. e. symbols expressing white spaces. Several evidences seem to indicate that the TrueType file format[6] and OpenType[7] had early taken into account the notion of invalidity, but also the notion of nullity[8,9]. Although invalidity is required to state which symbol to use by default when an encountered code-point does not have an associated known glyph registered, the presence of `null` is related to some algorithmic considerations in the domain of typesetting: in typography, `null` is assimilated as a completely blank symbol having a width equal to zero[10]. Historically, the font currently loaded by default in Isabelle/jEdit has as ancestor a font close to the group "Bitstream Vera"[11]. The last version 1.10 dates back from 2003[12]. At that time, Unicode characters was not natively integrated in Bitstream Vera, so it is not incorrect for non-Unicode fonts to support a range of code-point different than the actual Unicode range. Non-Unicode ranges could then be smaller, for Bitstream Vera, it was (an approximated size of) 0xFFFF characters. However the presence of `invalid` and `null` in Bitstream Vera may have been originated from specifications of the TrueType file format. `invalid` and `null` have been respectively located there at positions 0x10000 and 0x10001, while their respective glyphs might have also appeared in other positions: for example in Unicode, "U+0000" seems to also designate `null`. Due to the support of Unicode in Isabelle/jEdit, a copy of the Bitstream Vera font was taken and extended to support a wider range of characters (the increase was up to approximately 0x10FFFF characters). However, as side effect occurring during the extension, Isabelle has kept preserved at their respective positions the glyphs of both `invalid` and `null`. So `invalid` and `null` have always existed at positions 0x10000 and 0x10001 since at least 2009 (although 0x10000 and 0x10001 are normally reserved for other symbols in Unicode).[13] To conclude, ML actually

[5] `http://unicode.org/cldr/utility/character.jsp?a=05B9`

[6] `https://en.wikipedia.org/w/index.php?title=TrueType`

[7] `https://en.wikipedia.org/w/index.php?title=OpenType`

[8] `https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6post.html`

[9] `http://www.microsoft.com/typography/otspec/recom.htm`

[10] `https://en.wikipedia.org/w/index.php?title=Zero-width_space`

[11] `https://en.wikipedia.org/w/index.php?title=Bitstream_Vera`

[12] `http://ftp.gnome.org/pub/GNOME/sources/ttf-bitstream-vera/1.10/`

[13] Moreover `invalid` and `null` are *activated* at positions 0x10000 and 0x10001, because code fonts have the possibility to be deactivated: when encountering a deactivated font, `invalid` is generally automatically used instead of that font. For example in Isabelle, "U+0000", also

exists in Figure K.1 but is invisible: we have redefine the command ML to be the ghost symbol "U+10001"[14], also known as `null`.

As one corollary, since any non-empty appending of `null` with itself always produces in Isabelle/jEdit a shape similar as the empty string, to be strict, we need to give more precision about the number of symbols U+10001 we have actually used to overload the command ML. Indeed, one can use this feature to create an army of commands which are at the same time all different and all invisible. Moreover, the hovering with the mouse in this case is unable to detect `null`, and more generally any (non-empty) repetition of `null`.[15,16]

The last two commands *end* and *find_theorems* in Figure K.1 have been syntactically defined by inserting a number of symbols `null` somewhere. This is one way to visually give the impression that permutations of commands seem possible (even if we are using a version of Isabelle after 2014). As remark, parsing errors normally prevent the juxtaposition of two arbitrary commands, for example if these two commands only contain characters from the ASCII set. So it suffices to insert a non usual symbol inside one of these two commands (not necessarily at the beginning or the end, somewhere in the middle is accepted). This is how one can juxtapose *end* and *find_theorems* together (i. e. "*endfind_theorems*") to give the illusion of having a single command, even if at run-time at least two commands will be executed.

Finally as exercise, it would remain to determine how feasible one can dynamically change the color of blue commands to green (so that *find_theorems* would appear in green), be it for an entire word or for some particular sub-words, such as `invalid` and `null`.

known as `null`, is represented with the `invalid` shape because "U+0000" is deactivated.

[14] `http://unicode.org/cldr/utility/character.jsp?a=10001`

[15] In Isabelle/jEdit one can rely on other mechanisms to visualize the space occupied by commands, for instance the vertical bar on the left, usually used to collapse commands, can indicate their presence. However nothing is drawn if the command only occupies a single line or contains certain unusual characters...

[16] When applying the "⇩" operator in front of `null`, we obtain a shape having a positive width: the invisible property becomes lost.

Bibliography

[ACHA90] Stuart F Allen, Robert L Constable, Douglas J Howe, and William E Aitken. The semantics of reflected proof. In *Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 95–105. IEEE, 1990. 67

[ACM94] Catia M. Angelo, Luc J. M. Claesen, and Hugo De Man. Degrees of formality in shallow embedding hardware description languages in HOL. In Joyce and Seger [JS94], pages 89–100. 22

[ADEM14] Marcos Arjona, Carolina Dania, Marina Egea, and Antonio Maña. Validation of a security metamodel for the development of cloud applications. In Achim D. Brucker, Carolina Dania, Geri Georg, and Martin Gogolla, editors, *Proceedings of the MODELS 2014 OCL Workshop (OCL 2014)*, CEUR Workshop Proceedings. CEUR-WS.org, 2014. 48, 142

[AGH00] Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language, Third Edition*. Addison-Wesley, 2000. 11

[AJ04] Klaus Aehlig and Felix Joachimski. Operational aspects of untyped normalisation by evaluation. *Mathematical Structures in Computer Science*, 14(4):587–611, 2004. 51

[AJGL11] Jorge Soto Andrade, Sebastián Jaramillo, Claudio Gutiérrez, and Juan-Carlos Letelier. Ouroboros avatars: A mathematical exploration of self-reference and metabolic closure. In Tom Lenaerts, Mario Giacobini, Hugues Bersini, Paul Bourgine, Marco Dorigo, and René Doursat, editors, *Advances in Artificial Life: 20th Anniversary Edition - Back to the Origins of Alife, ECAL 2011, Paris, France, August 8-12, 2011*, pages 763–770. MIT Press, 2011. 77

[And02] Peter B. Andrews. *Introduction to Mathematical Logic and Type Theory: To Truth through Proof*. Kluwer Academic Publishers, Dordrecht, 2nd edition, 2002. 21

[App16] Apple Inc. The swift programming language. `https://swift.org/documentation/TheSwiftProgrammingLanguage(Swift2.2).epub`, 2016. Swift 2.2 Edition. 11

[Bal14] Clemens Ballarin. Locales: A module system for mathematical theories. *J. Autom. Reasoning*, 52(2):123–153, 2014. 130, 147

[Bal16] Clemens Ballarin. *Tutorial to Locales and Locale Interpretation*, 2016. `http://isabelle.in.tum.de/doc/locales.pdf`. 130, 147

[Bar91] Henk Barendregt. Introduction to generalized type systems. *J. Funct. Program.*, 1(2):125–154, 1991. 76

[Bar10] Bruno Barras. Sets in coq, coq in sets. *J. Formalized Reasoning*, 3(1):29–48, 2010. 142

[Bas93] David A Basin. Metalogical frameworks. *Logical Environments*, pages 1–29, 1993. 67

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer-Verlag, Heidelberg, 2004. 12

[BCC+13] Achim D. Brucker, Dan Chiorean, Tony Clark, Birgit Demuth, Martin Gogolla, Dimitri Plotnikov, Bernhard Rumpe, Edward D. Willink, and Burkhart Wolff. Report on the Aachen OCL meeting. In Jordi Cabot, Martin Gogolla, Istvan Rath, and Edward Willink, editors, *Proceedings of the* MoDELS *2013 OCL Workshop (OCL 2013)*, volume 1092 of *CEUR Workshop Proceedings*, pages 103–111. CEUR-WS.org, 2013. 30

[BCF+13] Patrick Baudin, Pascal Cuoq, Jean-Christophe Filliâtre, Claude Marché, Benjamin Monate, Yannick Moy, and Virgile Prevosto. *ACSL: ANSI/ISO C Specification Language*, 2013. Version 1.8. 12, 49, 142

[BCM04] David A. Basin, Manuel Clavel, and José Meseguer. Reflective metalogical frameworks. *ACM Trans. Comput. Log.*, 5(3):528–576, 2004. 59

[BDP+16] Jasmin Christian Blanchette, Martin Desharnais, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. *Defining (Co)datatypes in Isabelle/HOL*, 2016. `http://isabelle.in.tum.de/doc/datatypes.pdf`. 23

[BDW06a] Achim D. Brucker, Jürgen Doser, and Burkhart Wolff. An MDA framework supporting OCL. *Electronic Communications of the EASST*, 5, 2006. 94

[BDW06b] Achim D. Brucker, Jürgen Doser, and Burkhart Wolff. A model transformation semantics and analysis methodology for SecureUML. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *MoDELS 2006: Model Driven Engineering Languages and Systems*, number 4199 in Lecture Notes in Computer Science, pages 306–320. Springer-Verlag, 2006. An extended version of this paper is available as ETH Technical Report, no. 524. 102

[BDW06c] Achim D. Brucker, Jürgen Doser, and Burkhart Wolff. Semantic issues of OCL: Past, present, and future. *Electronic Communications of the EASST*, 5, 2006. 14

[BGG+93] Richard Boulton, Andrew Gordon, Michael J. C. Gordon, John Harrison, John Herbert, and John Van Tassel. Experience with embedding hardware description languages in HOL. In Victoria Stavridou, Thomas F. Melham, and Raymond T. Boute, editors, *Proceedings of*

the the International Conference on Theorem Provers in Circuit Design: Theory, Practice and Experience, volume A-10 of *IFIP Transactions*, pages 129–156, Nijmegen, The Netherlands, 1993. North-Holland Publishing Co. 22

[BHL+14] Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. Truly modular (co)datatypes for isabelle/hol. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 93–110. Springer, 2014. 23, 74, 81

[BHOG01] Franck Barbier, Brian Henderson-Sellers, Andreas L. Opdahl, and Martin Gogolla. The whole-part relationship in the unified modeling language: A new approach. In *Unified Modeling Language: Systems Analysis, Design and Development Issues*, pages 185–209. IGI Global, Hershey, PA, USA, 2001. 34

[BKLW10] Achim D. Brucker, Matthias P. Krieger, Delphine Longuet, and Burkhart Wolff. A specification-based test case generation method for UML/OCL. In Jürgen Dingel and Arnor Solberg, editors, *MoDELS Workshops*, number 6627 in Lecture Notes in Computer Science, pages 334–348. Springer-Verlag, 2010. Selected best papers from all satellite events of the MoDELS 2010 conference. Workshop on OCL and Textual Modelling. 30

[BKW09] Achim D. Brucker, Matthias P. Krieger, and Burkhart Wolff. Extending OCL with null-references. In Sudipto Gosh, editor, *Models in Software Engineering*, number 6002 in Lecture Notes in Computer Science, pages 261–275. Springer-Verlag, 2009. Selected best papers from all satellite events of the MoDELS 2009 conference. 112

[Bla16] Jasmin Christian Blanchette. *Hammering Away: A User's Guide to Sledgehammer for Isabelle/HOL*, 2016. `http://isabelle.in.tum.de/doc/sledgehammer.pdf`. 91

[BLS05] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In Gilles Barthe, Lilian Burdy, Marieke Huisman, Jean-Louis Lanet, and Traian Muntean, editors, *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices (CASSIS)*, volume 3362 of *Lecture Notes in Computer Science*, pages 49–69, Heidelberg, May 25 2005. Springer-Verlag. 12, 49, 142

[BLTW13] Achim D. Brucker, Delphine Longuet, Frédéric Tuong, and Burkhart Wolff. On the semantics of object-oriented data structures and path expressions. In Jordi Cabot, Martin Gogolla, István Ráth, and Edward D. Willink, editors, *Proceedings of the* MoDELS *2013 OCL Workshop (OCL 2013)*, volume 1092 of *CEUR Workshop Proceedings*,

pages 23–32. CEUR-WS.org, 2013. An extended version of this paper is available as LRI Technical Report 1565. 30

[BLW08] Sascha Böhme, K. Rustan M. Leino, and Burkhart Wolff. HOL-Boogie—an interactive prover for the Boogie program-verifier. In Otmane Aït Mohamed, César Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics*, volume 5170 of *Lecture Notes in Computer Science*, pages 150–166, Heidelberg, August 2008. Springer-Verlag. 49

[BM79] Robert S Boyer and J Strother Moore. Metafunctions: Proving them correct and using them efficiently as new proof procedures. Technical report, DTIC Document, 1979. 67

[Bou97] Samuel Boutin. Using reflection to build efficient and certified decision procedures. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software, Third International Symposium, TACS '97, Sendai, Japan, September 23-26, 1997, Proceedings*, volume 1281 of *Lecture Notes in Computer Science*, pages 515–529. Springer, 1997. 67

[Bru07] Achim D. Brucker. *An Interactive Proof Environment for Object-oriented Specifications*. PhD thesis, ETH Zurich, March 2007. ETH Dissertation No. 17097. 33

[BRW03] Achim D. Brucker, Frank Rittinger, and Burkhart Wolff. HOL-Z 2.0: A proof environment for Z-specifications. *Journal of Universal Computer Science*, 9(2):152–172, February 2003. 22

[BS91] Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed lambda-calculus. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 203–211. IEEE Computer Society, 1991. 51

[BT07] Clark Barrett and Cesare Tinelli. Cvc3. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, 2007. 43

[BTT15] Bruno Barras, Carst Tankink, and Enrico Tassi. Asynchronous processing of coq documents: From the kernel up to the user interface. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2015. 13

[BTW14] Achim D. Brucker, Frédéric Tuong, and Burkhart Wolff. Featherweight OCL: A proposal for a machine-checked formal semantics for OCL 2.5. *Archive of Formal Proofs*, January 2014. `http://www.isa-afp.org/entries/Featherweight_OCL.shtml`, Formal proof development. 14, 30, 37, 43, 46, 47, 60, 94, 132, 140, 148

[BW97] Bruno Barras and Benjamin Werner. Coq in coq. Technical report, Inria, 1997. 142

[BW99] Stefan Berghofer and Markus Wenzel. Inductive datatypes in HOL - lessons learned in formal-logic engineering. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings*, volume 1690 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 1999. 23, 81

[BW01] Gertrud Bauer and Markus Wenzel. Calculational reasoning revisited (an isabelle/isar experience). In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings*, volume 2152 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2001. 61

[BW02a] Achim D. Brucker and Burkhart Wolff. HOL-OCL: Experiences, consequences and design choices. In Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook, editors, *UML 2002: Model Engineering, Concepts and Tools*, number 2460 in Lecture Notes in Computer Science, pages 196–211. Springer-Verlag, 2002. 14

[BW02b] Achim D. Brucker and Burkhart Wolff. A proposal for a formal OCL semantics in Isabelle/HOL. In Víctor A. Carreño, César A. Muñoz, and Sophiène Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs)*, number 2410 in Lecture Notes in Computer Science, pages 99–114. Springer-Verlag, 2002. 30

[BW06] Achim D. Brucker and Burkhart Wolff. The HOL-OCL book. Technical Report 525, ETH Zurich, 2006. 45

[BW08a] Achim D. Brucker and Burkhart Wolff. HOL-OCL – A Formal Proof Environment for UML/OCL. In José Fiadeiro and Paola Inverardi, editors, *Fundamental Approaches to Software Engineering (FASE08)*, number 4961 in Lecture Notes in Computer Science, pages 97–100. Springer-Verlag, 2008. 14, 45, 63, 141

[BW08b] Achim D. Brucker and Burkhart Wolff. An extensible encoding of object-oriented data models in HOL. *Journal of Automated Reasoning*, 41:219–249, 2008. 33, 46, 47, 141, 142

[BW08c] Achim D. Brucker and Burkhart Wolff. Extensible universes for object-oriented data models. In Jan Vitek, editor, *ECOOP 2008 – Object-Oriented Programming*, number 5142 in Lecture Notes in Computer Science, pages 438–462. Springer-Verlag, 2008. 142

[BW09] Achim D. Brucker and Burkhart Wolff. Semantics, calculi, and analysis for object-oriented specifications. *Acta Informatica*, 46(4):255–284, July 2009. 22, 127, 141

[CDH+09] Ernie Cohen, Markus Dahlweid, Mark A. Hillebrand, Dirk Leinenbach, Michal Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC: A practical system for verifying concurrent C. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5674 of *Lecture Notes in Computer Science*, pages 23–42. Springer-Verlag, 2009. 49

[CFC58] Haskell B. Curry, Robert Feys, and William Craig. *Combinatory Logic*, volume 1. North-Holland, 1958. §9E. 64

[Che76] Peter Pin-Shan Chen. The entity-relationship model – toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, March 1976. 34

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, June 1940. 21

[CKM+02] Steve Cook, Anneke Kleppe, Richard Mitchell, Bernhard Rumpe, Jos Warmer, and Alan Wills. The amsterdam manifesto on OCL. In Tony Clark and Jos Warmer, editors, *Object Modeling with the OCL: The Rationale behind the Object Constraint Language*, volume 2263 of *Lecture Notes in Computer Science*, pages 115–149, Heidelberg, 2002. Springer-Verlag. 30

[CM96] Manuel G. Clavel and José Meseguer. Axiomatizing reflective logics and languages. In *Proceedings of Reflection'96*, pages 263–288, 1996. 67

[CN05] Amine Chaieb and Tobias Nipkow. Verifying and reflecting quantifier elimination for presburger arithmetic. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*, pages 367–380. Springer, 2005. 62, 67

[Coq16] *The Coq proof assistant reference manual*, 2016. 89

[Cos02] Stefania Costantini. Meta-reasoning: A survey. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, volume 2408 of *Lecture Notes in Computer Science*, pages 253–288. Springer, 2002. 66

[CR05] Patrice Chalin and Frédéric Rioux. Non-null references by default in the Java modeling language. In *SAVCBS '05: Proceedings of the 2005 conference on Specification and verification of component-based systems*, page 9, New York, NY USA, 2005. ACM Press. 143

[Cut80] Nigel Cutland. *Computability: An introduction to recursive function theory*. Cambridge university press, 1980. 77

[Dan98] Olivier Danvy. Type-directed partial evaluation. In John Hatcliff, Torben Æ. Mogensen, and Peter Thiemann, editors, *Partial Evaluation - Practice and Theory, DIKU 1998 International Summer School, Copenhagen, Denmark, June 29 - July 10, 1998*, volume 1706 of *Lecture Notes in Computer Science*, pages 367–411. Springer, 1998. 51

[dB80] N.J. de Bruijn. A survey of the project Automath. In J.P. Seldin and J.R. Hindley, editors, *to H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980. 64

[DC13] Carolina Dania and Manuel Clavel. OCL2FOL+: coping with undefinedness. In Jordi Cabot, Martin Gogolla, István Ráth, and Edward D. Willink, editors, *Proceedings of the MODELS 2013 OCL Workshop co-located with the 16th International ACM/IEEE Conference on Model Driven Engineering Languages and Systems (MODELS 2013), Miami, USA, September 30, 2013.*, volume 1092 of *CEUR Workshop Proceedings*, pages 53–62. CEUR-WS.org, 2013. 48, 142

[Del00] David Delahaye. A tactic language for the system coq. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000. 74

[DF00] Peter Dybjer and Andrzej Filinski. Normalization and partial evaluation. In Gilles Barthe, Peter Dybjer, Luis Pinto, and João Saraiva, editors, *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science*, pages 137–192. Springer, 2000. 51

[DM82] Luís Damas and Robin Milner. Principal type-schemes for functional programs. In Richard A. DeMillo, editor, *Conference Record of the Ninth Annual ACM Symposium on Principles of Programming Languages, Albuquerque, New Mexico, USA, January 1982*, pages 207–212. ACM Press, 1982. 19

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Heidelberg, 2008. Springer-Verlag. 43

[DN66] Ole-Johan Dahl and Kristen Nygaard. SIMULA - an algol-based simulation language. *Commun. ACM*, 9(9):671–678, 1966. 11

[Dre16] Dresden OCL. `http://www.dresden-ocl.org/`, 2016. 47

[ELN+14] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified exe-

cutable LTL model checker. *Archive of Formal Proofs*, 2014, 2014. 136

[GBR07] Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A UML-based specification environment for validating UML and OCL. *Sci. Comput. Program.*, 69(1-3):27–34, 2007. 15

[GMW79] Michael J. C. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *LNCS*. Springer, 1979. 12, 62, 70

[Gor00] Mike Gordon. From LCF to HOL: a short history. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, pages 169–185. MIT Press, Cambridge, Massachusetts, 2000. 70

[GR02] Martin Gogolla and Mark Richters. Expressing UML class diagrams properties with OCL. In Tony Clark and Jos Warmer, editors, *Object Modeling with the OCL: The Rationale behind the Object Constraint Language*, volume 2263 of *Lecture Notes in Computer Science*, pages 85–114, Heidelberg, 2002. Springer-Verlag. 102

[Gun92] Elsa L. Gunter. Why we can't have sml-style datatype declarations in HOL. In Luc J. M. Claesen and Michael J. C. Gordon, editors, *Higher Order Logic Theorem Proving and its Applications, Proceedings of the IFIP TC10/WG10.2 Workshop HOL'92, Leuven, Belgium, 21-24 September 1992*, volume A-20 of *IFIP Transactions*, pages 561–568. North-Holland/Elsevier, 1992. 74

[HÖ94] Reiner Hähnle. Efficient deduction in many-valued logics. In *International Symposium on Multiple-Valued Logics (ISMVL)*, pages 240–249, Los Alamitos, CA, USA, 1994. IEEE Computer Society. 48

[Haf09] Florian Haftmann. *Code generation from specifications in higher-order logic*. PhD thesis, Technical University Munich, 2009. 24, 70

[Haf16] Florian Haftmann. *Code generation from Isabelle theories*, 2016. `http://isabelle.in.tum.de/doc/codegen.pdf`. 24, 70, 85

[Har95] John Harrison. Metatheory and reflection in theorem proving: A survey and critique. Technical report, Citeseer, 1995. 67

[Har14] John Harrison. *HOL Light Tutorial*, November 2014. 12

[HCH+98] Ali Hamie, Franco Civello, John Howse, Stuart Kent, and Richard Mitchell. Reflections on the Object Constraint Language. In Jean Bézivin and Pierre-Alain Muller, editors, *The Unified Modeling Language. «UML»'98: Beyond the Notation*, volume 1618 of *Lecture Notes in Computer Science*, pages 162–172, Heidelberg, 1998. Springer-Verlag. 30

[HHJW07] Paul Hudak, John Hughes, Simon L. Peyton Jones, and Philip Wadler. A history of haskell: being lazy with class. In Barbara G. Ryder and Brent Hailpern, editors, *Proceedings of the Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III), San Diego, California, USA, 9-10 June 2007*, pages 1–55. ACM, 2007. 19

[Hin69] Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the american mathematical society*, 146:29–60, 1969. 19

[HN10] Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In Matthias Blume, Naoki Kobayashi, and Germán Vidal, editors, *Functional and Logic Programming, 10th International Symposium, FLOPS 2010, Sendai, Japan, April 19-21, 2010. Proceedings*, volume 6009 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2010. 24, 70

[How80] W.A. Howard. The formulae-as-types notion of constructions. In J.P. Seldin and J.R. Hindley, editors, *to H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980. Unpublished 1969 Manuscript. 64

[Hue92] Gérard P. Huet. The gallina specification language: A case study. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, volume 652 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 1992. 12

[JS94] Jeffrey J. Joyce and Carl-Johan H. Seger, editors. *Higher Order Logic Theorem Proving and Its Applications (HUG)*, volume 780 of *Lecture Notes in Computer Science*, Heidelberg, 1994. Springer-Verlag. 22, 355

[KAE+10] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: formal verification of an operating-system kernel. *Commun. ACM*, 53(6):107–115, 2010. 22, 63

[Kay93] Alan C. Kay. The early history of smalltalk. In John A. N. Lee and Jean E. Sammet, editors, *History of Programming Languages Conference (HOPL-II), Preprints, Cambridge, Massachusetts, USA, April 20-23, 1993*, pages 69–95. ACM, 1993. 11

[KG12] Mirco Kuhlmann and Martin Gogolla. From UML and OCL to relational logic and back. In Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings*, volume 7590 of *Lecture Notes in Computer Science*, pages 415–431. Springer-Verlag, 2012. 47

[KK08] Matthias P. Krieger and Alexander Knapp. Executing underspecified ocl operation contracts with a sat solver. In *Proceedings of the the OCL 2008 Workshop*, 2008. `http://www.fots.ua.ac.be/events/ocl2008/`. 47

[KL12] Jason Koenig and K. Rustan M. Leino. Getting started with Dafny: A guide. In Tobias Nipkow, Orna Grumberg, and Benedikt Hauptmann, editors, *Software Safety and Security: Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series D: Information and Communication Security*, pages 152–181. IOS Press, 2012. Summer School Marktoberdorf 2011 lecture notes. A version of this tutorial is available online at `http://rise4fun.com/dafny`. 12

[Kle38] Stephen Cole Kleene. On notation for ordinal numbers. *J. Symb. Log.*, 3(4):150–155, 1938. 77

[Kra06] Alexander Krauss. Partial recursive functions in higher-order logic. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 589–603. Springer, 2006. 24

[Kra16] Alexander Krauss. *Defining Recursive Functions in Isabelle/HOL*, 2016. `http://isabelle.in.tum.de/doc/functions.pdf`. 24

[LDF+14] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The OCaml system release 4.02: Documentation and user's manual. Interne, Inria, September 2014. 24

[Ler09] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009. 61, 142

[LNS00] K. Rustan M. Leino, Greg Nelson, and James B. Saxe. ESC/Java user's manual. Technical Report SRC-2000-002, Compaq Systems Research Center, October 2000. 49

[LPC+13] Gary T. Leavens, Erik Poll, Curtis Clifton, Yoonsik Cheon, Clyde Ruby, David R. Cok, Peter Müller, Joseph Kiniry, Patrice Chalin, Daniel M. Zimmermann, and Werner Dietl. JML reference manual, May 2013. Revision 2344. Available from `http://www.jmlspecs.org`. 12, 49, 142

[LTW14] Delphine Longuet, Frédéric Tuong, and Burkhart Wolff. Towards a tool for featherweight OCL: A case study on semantic reflection. In Achim D. Brucker, Carolina Dania, Geri Georg, and Martin Gogolla, editors, *Proceedings of the* MoDELS *2014 OCL Workshop (OCL 2014)*, volume 1285 of *CEUR Workshop Proceedings*, pages 43–52. CEUR-WS.org, 2014. 30

[MC99] Luis Mandel and Marìa Victoria Cengarle. On the expressive power of OCL. In Jeannette M. Wing, Jim Woodcock, and Jim Davies,

editors, *World Congress on Formal Methods in the Development of Computing Systems (FM)*, volume 1708 of *Lecture Notes in Computer Science*, pages 854–874, Heidelberg, 1999. Springer-Verlag. 30

[McC65] John McCarthy. *LISP 1.5 programmer's manual*. MIT press, 1965. 12

[Mel91] Thomas F. Melham. A package for inductive relation definitions in HOL. In Myla Archer, Jeffrey J. Joyce, Karl N. Levitt, and Phillip J. Windley, editors, *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications, August 1991, Davis, California, USA*, pages 350–357. IEEE Computer Society, 1991. 12, 57

[Mes92] José Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992. 76

[Mey97] Bertrand Meyer. *Object-oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition, 1997. 48, 142

[Mil78] Robin Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978. 19

[Mil97] Robin Milner. *The definition of standard ML: revised*. MIT press, 1997. 22

[ML75] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975. 51

[ML84] Per Martin-Lef. Intuitionistic type theory. *Naples: Bibliopolis*, 1984. 76

[Mog91] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. 62

[MP08] Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008. 91

[MW10] David C. J. Matthews and Makarius Wenzel. Efficient parallel programming in poly/ml and isabelle/ml. In Leaf Petersen and Enrico Pontelli, editors, *Proceedings of the POPL 2010 Workshop on Declarative Aspects of Multicore Programming, DAMP 2010, Madrid, Spain, January 19, 2010*, pages 53–62. ACM, 2010. 74

[MWM14] Daniel Matichuk, Makarius Wenzel, and Toby C. Murray. An isabelle proof method language. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 390–405. Springer, 2014. 63, 74

[NFWP15] Yakoub Nemouchi, Abderrahmane Feliachi, Burkhart Wolff, and Cyril Proch. Isabelle in certification processes. Technical Report 1583, LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay, December 2015. `http://www.lri.fr/~bibli/Rapports-internes/2015/RR1583.pdf`. 27, 85, 98

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 2002. 19, 30, 33

[NPW09] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. Isabelle's logic: HOL, 2009. 12, 14

[NS14] Michael Norrish and Konrad Slind. *The HOL System Tutorial*, November 2014. 12

[Oa04] Martin Odersky and al. An overview of the scala programming language. Technical Report IC/2004/64, EPFL Lausanne, Switzerland, 2004. 11

[Obj97] Object constraint language specification (version 1.1), September 1997. Available as OMG document ad/97-08-08. 30

[Obj06] UML 2.0 OCL specification, April 2006. Available as OMG document formal/06-05-01. 30

[Obj11a] UML 2.4.1: Infrastructure specification, August 2011. Available as OMG document formal/2011-08-05. 29, 33

[Obj11b] UML 2.4.1: Superstructure specification, August 2011. Available as OMG document formal/2011-08-06. 29, 33

[Obj12] UML 2.3.1 OCL specification, February 2012. Available as OMG document formal/2012-01-01. 14, 29, 30, 31, 45

[Pap16] Papyrus UML. `http://www.papyrusuml.org`, 2016. 47

[Pau16] Lawrence C. Paulson. *Isabelle's Logics*, 2016. `http://isabelle.in.tum.de/doc/logics.pdf`. 73

[PB10] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, volume 2 of *EPiC Series*, pages 1–11. EasyChair, 2010. 91

[PS07] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings*, volume 4732 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2007. 91

[Ren06] Renesas Electronics. *SH-4 Software Manual, Renesas 32-Bit RISC, Rev.6.00*. Renesas Electronics, 2006. 142

[RG02] Mark Richters and Martin Gogolla. OCL: Syntax, semantics, and tools. In Tony Clark and Jos Warmer, editors, *Object Modeling with the OCL: The Rationale behind the Object Constraint Language*, volume 2263 of *Lecture Notes in Computer Science*, pages 42–68, Heidelberg, 2002. Springer-Verlag. 34, 47

[Ric02] Mark Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, Logos Verlag, Berlin, BISS Monographs, No. 14, 2002. 29, 30, 45

[RL14] Martin Ring and Christoph Lüth. Collaborative interactive theorem proving with clide. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 467–482. Springer, 2014. 53

[Sch08] Norbert Schirmer. A sequential imperative programming language syntax, semantics, hoare logics and verification environment. *Archive of Formal Proofs*, February 2008. `http://www.isa-afp.org/entries/Simpl.shtml`, Formal proof development. 22

[Shi13] Xiaomu Shi. *Certification of an Instruction Set Simulator*. Theses, Université de Grenoble, July 2013. 141

[Smi82] Brian Cantwell Smith. *Reflections and semantics in a procedural language*. Massachusetts Institute of Technology, Laboratory for Computer Science, 1982. 66, 87

[Smi84] Brian Cantwell Smith. Reflection and semantics in lisp. In Ken Kennedy, Mary S. Van Deusen, and Larry Landweber, editors, *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages, Salt Lake City, Utah, USA, January 1984*, pages 23–35. ACM Press, 1984. 66, 87

[SMTB11] Xiaomu Shi, Jean-François Monin, Frédéric Tuong, and Frédéric Blanqui. First steps towards the certification of an ARM simulator using compcert. In *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, pages 346–361, 2011. 141

[SS12] Bas R. Steunebrink and Jürgen Schmidhuber. Towards an actual gödel machine implementation: A lesson in self-reflective systems. In *Theoretical Foundations of Artificial General Intelligence*, pages 173–195. Springer, 2012. 87

[Ste02] Mark-Oliver Stehr. *Towards a unified language based on equational logic, rewriting logic, and type theory*. PhD thesis, Universität Hamburg, 2002. 76

[Str86] Bjarne Stroustrup. C++ programming language. *IEEE Software*, 3(1):71–72, 1986. 11

[SU06] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149. Elsevier, 2006. 38

[TPB12] Dmitriy Traytel, Andrei Popescu, and Jasmin Christian Blanchette. Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 596–605. IEEE Computer Society, 2012. 74

[TW15] Frédéric Tuong and Burkhart Wolff. A meta-model for the isabelle API. *Archive of Formal Proofs*, September 2015. `http://www.isa-afp.org/entries/Isabelle_Meta_Model.shtml`, Formal proof development. 15, 31, 64, 65, 66, 67, 71, 85, 93, 287, 313, 325

[VGPA00] Kumar Neeraj Verma, Jean Goubault-Larrecq, Sanjiva Prasad, and S. Arun-Kumar. Reflecting bdds in coq. In Jifeng He and Masahiko Sato, editors, *Advances in Computing Science - ASIAN 2000, 6th Asian Computing Science Conference, Penang, Malaysia, November 25-27, 2000, Proceedings*, volume 1961 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2000. 67

[vR07] Guido van Rossum. Python programming language. In Jeff Chase and Srinivasan Seshan, editors, *Proceedings of the 2007 USENIX Annual Technical Conference, Santa Clara, CA, USA, June 17-22, 2007*. USENIX, 2007. 19

[WB89] Philip Wadler and Stephen Blott. How to make ad-hoc polymorphism less ad-hoc. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 60–76. ACM Press, 1989. 20

[WC07] Makarius Wenzel and Amine Chaieb. SML with antiquotations embedded into Isabelle/Isar. In J. Carette and F. Wiedijk, editors, *Programming Languages for Mechanized Mathematics Workshop (CALCULEMUS 2007)*, number 07-10 in RISC-Linz Report. RISC, June 2007. 62, 68, 70, 142

[Wen97] Markus Wenzel. Type classes and overloading in higher-order logic. In Elsa L. Gunter and Amy P. Felty, editors, *Theorem Proving in Higher Order Logics, 10th International Conference, TPHOLs'97, Murray Hill, NJ, USA, August 19-22, 1997, Proceedings*, volume 1275 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 1997. 142

[Wen99] Markus Wenzel. Isar - A generic interpretative approach to readable formal proof documents. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry, editors,

Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings, volume 1690 of *Lecture Notes in Computer Science*, pages 167–184. Springer, 1999. 61

[Wen02] Markus M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, TU München, München, February 2002. 12, 15, 21

[Wen09] M. Wenzel. Parallel proof checking in Isabelle/Isar. In G. Dos Reis and L. Théry, editors, *ACM SIGSAM Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS 2009)*. ACM Digital Library, 2009. 26, 74

[Wen12] Makarius Wenzel. Asynchronous proof processing with isabelle/scala and isabelle/jedit. *Electr. Notes Theor. Comput. Sci.*, 285:101–114, 2012. 24

[Wen14] Makarius Wenzel. Asynchronous user interaction and tool integration in isabelle/pide. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 515–530. Springer, 2014. 26, 53, 74

[Wen16a] Makarius Wenzel. *The Isabelle/Isar Implementation*, 2016. `http://isabelle.in.tum.de/doc/implementation.pdf`. 68, 70

[Wen16b] Makarius Wenzel. *The Isabelle/Isar Reference Manual*, 2016. `http://isabelle.in.tum.de/doc/isar-ref.pdf`. 21, 23, 24, 45, 53, 57, 70, 81, 85, 88, 94, 95, 98, 283, 343

[Wen16c] Makarius Wenzel. *Isabelle/jEdit*, 2016. `http://isabelle.in.tum.de/doc/jedit.pdf`. 24

[Wey80] Richard W Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial intelligence*, 13(1-2):133–170, 1980. 67

[WW02] Markus Wenzel and Freek Wiedijk. A comparison of mizar and isar. *J. Autom. Reasoning*, 29(3-4):389–411, 2002. 61

[WW07] Makarius Wenzel and Burkhart Wolff. Building formal method tools in the Isabelle/Isar framework. In Klaus Schneider and Jens Brandt, editors, *TPHOLS 2007*, number 4732 in Lecture Notes in Computer Science, pages 352–367. Springer-Verlag, Heidelberg, 2007. 26, 141

Abstract

Les langages de spécifications basés et orientés objets (comme UML/OCL, JML, Spec#, ou Eiffel) permettent la création et destruction, la conversion et tests de types dynamiques d'objets statiquement typés. Par dessus, les invariants de classes et les opérations de contrat peuvent y être exprimés; ces derniers représentent les éléments clés des spécifications orientées objets. Une sémantique formelle des structures de données orientées objets est complexe: des descriptions imprécises mènent souvent à différentes interprétations dans les outils qui en résultent.

Dans cette thèse, nous démontrons comment dériver un environnement de preuves moderne comme un *méta-outil* pour la définition et l'analyse de sémantique formelle de langages de spécifications orientés objets. Étant donné une représentation d'un langage particulier plongé en Isabelle/HOL, nous construisons pour ce langage un environnement étendu d'Isabelle, à travers une *méthode* de génération de code particulière, qui implique notamment plusieurs variantes de génération de code. Le résultat supporte l'édition asynchrone, la vérification de types, et les activités de déduction formelle, tous "hérités" d'Isabelle.

En application de cette méthode, nous obtenons un *outil de modélisation orienté objet* pour du UML/OCL textuel. Nous intégrons également des idiomes non nécessairement présent dans UML/OCL— en d'autres termes, nous développons un support pour des dialectes d'UML/OCL à domaine spécifique.

En tant que construction méta, nous définissons un méta-modèle d'une partie d'UML/OCL en HOL, un méta-modèle d'une partie de l'API d'Isabelle en HOL, et une fonction de traduction entre eux en HOL. Le méta-outil va alors exploiter deux procédés de générations de code pour produire soit du *code raisonnablement efficace*, soit du *code raisonnablement lisible*. Cela fournit donc deux modes d'animations pour inspecter plus en détail la sémantique d'un langage venant d'être plongé: en chargeant à vitesse réelle sa sémantique, ou simplement en retardant à un autre niveau "méta" l'expérimentation précédente pour un futur instant de typage en Isabelle, que ce soit pour des raisons de performances, de tests ou de prototypages.

Remarquons que la génération de "code raisonnablement efficace", et de "code raisonnablement lisible" incluent la génération de *code tactiques* qui prouvent une collection de théorèmes formant une théorie de types de données orientés objets d'un modèle dénotationnel: étant donné un modèle de classe UML/OCL, les preuves des propriétés pertinentes aux conversions, tests de types, constructeurs et sélecteurs sont traitées automatiquement. Cette fonctionnalité est similaire aux *paquets de théories de types de données* présents au sein d'autres prouveurs de la famille HOL, à l'exception que certaines motivations ont conduit ce travail présent à programmer des tactiques haut-niveaux en HOL lui-même.

Ce travail prend en compte les plus récentes avancées du standard d'UML/OCL 2.5. Par conséquent, tous les types UML/OCL ainsi que les types logiques distinguent deux éléments d'exception différents: `invalid` (exception) et `null` (élément non-existant). Cela entraîne des conséquences sur les propriétés aussi

bien logiques qu'algébriques des structures orientées objets résultant des modèles de classes.

Étant donné que notre construction est réduite à une séquence d'extension conservative de théorie, notre approche peut garantir la correction logique du langage entier considéré, et fournit une méthodologie pour étendre formellement des langages à domaine spécifique.

Mots-clés

Structures de données orientés objets, Chemins d'expression, Featherweight OCL, Null, Invalid, Sémantique formelle, Isabelle, Réflexion, UML, OCL.

Résumé

Nous avons présenté HOL-OCL 2.0, basé sur une librairie cœur Featherweight OCL, une sémantique pour UML/OCL formellement vérifiée par machine en Isabelle/HOL. HOL-OCL 2.0 comprend un méta-outil pour construire des outils sémantiques adaptés pour des langages à domaines spécifiques textuels. Le méta-outil s'appuie fondamentalement sur le générateur de code d'Isabelle, ainsi que sur les théories d'Isabelle, pour définir une transformation de modèle en Isabelle/Isar_HOL depuis un méta-modèle d'UML (modèles de classes, plus invariants OCL et contrats) vers un méta-modèle d'Isar_HOL. En comparaison avec les implémentations conventionnelles de *générateurs de code* pour OCL, le méta-outil résultant n'est clairement pas compétitif en termes de tailles de modèles compilées, dans un certain sens, nous argumentons que cette comparaison n'est pas équitable puisque ces outils ne s'occupent pas à construire la *théorie sémantique* sous-jacente d'UML et OCL en HOL de manière à pouvoir y bâtir par dessus des preuves formelles. Notre outil est unique parce qu'il produit effectivement deux manières de charger les productions de théorèmes résultant des modèles de classes: de manière native au moment de l'exécution, avec une interaction directe avec le noyau d'Isabelle (en shallow-mode); ou comme un certificat Isabelle à charger par la suite comme un logique-objet (en deep-mode).

Construit à partir d'une librairie d'opérations pour types de base et types collections prenant en charge les éléments d'exception `invalid` et `null`, HOL-OCL 2.0 permet la spécification de programmes basés sur des structures de données orientés objets. Notre travail en précise la notion et apporte une comparaison avec d'autres langages de spécification orientés objets tels que Eiffel, Spec# ou JML. Comme innovation particulière, notre approche concerne les théories de types de données, qui sont construites à partir de définitions axiomatiques et élaborées autour d'un univers d'objets typés, permettant la *dérivation* automatique de la totalité de ces règles et garantissant la consistance logique de l'ensemble.[1] Étant donné que l'environnement d'HOL-OCL 2.0 instancie dynamiquement et décharge ces règles durant l'activité de modélisation orientée objet (typiquement celles présentées dans Chapitre 7), notre approche est, comme nous le pensons, pertinente pour d'autres méthodes de vérifications orientées objets qui axiomatisent leurs théories sous-jacentes, et donc soulèvent des questions sur la portée de l'ensemble.

Dues aux techniques de parallélisation héritées d'Isabelle, HOL-OCL 2.0— pour lequel nous voyons encore un large potentiel d'optimisations— reste raisonnablement utilisable dans un contexte interactif pour des modèles de classes de taille moyenne. Comme le montre notre implémentation, la génération automatique avec preuves de théorie de types de données s'intègre aisément dans un milieu interactif: en reprenant l'exemple Flight étudié, 2301 définitions et lemmes sont générés en 1 seconde en deep-mode, alors que leurs preuves terminent de manière asynchrone en shallow-mode 2 minutes plus tard (dans un

[1] Nos deux exemples Annexe B et Annexe C esquissent comment cette construction peut être effectuée par un processus automatique.

fil d'exécution d'arrière plan). Encore une fois, les lemmes non-dépendant les uns des autres peuvent être activés ou désactivés: par défaut ils sont tous prouvés.

Il s'agit de notre but ultime de complémenter HOL-OCL 2.0 avec les types modèles comportementaux d'UML les plus communs, notamment les présentations textuelles de machines à état et diagrammes de séquences. L'environnement résultant pourrait servir comme démonstrateur de techniques formels pour UML et avantager les partenaires industriels actifs dans le domaine des systèmes embarqués.

Notre travail sur HOL-OCL 2.0 se situe dans le cadre d'une initiative de normalisation impliquant les méthodes formelles pour UML/OCL. En particulier, une sémantique formelle a été développée dans cette thèse pour un sous-ensemble du langage basé sur des définitions sémantiques dénotationnelles. L'ensemble des règles, nécessaires aux différentes techniques de preuves interactives et automatiques, a été dérivé avec un assistant de preuve interactif, apportant en même temps des éléments clés pour prouver des méta-lemmes et méga-lemmes. Étant donné que notre approche peut garantir leurs consistances logiques, non seulement pour les milliers de théorèmes générés, mais précisément pour le fondement de la librarie cœur de Featherweight OCL en lui-même, nous estimons que cette expérience peut servir à des efforts similaires de normalisation de langages de programmation "réels", ou au moins montrer que ce type de travail est de nos jours absolument réalisable avec des bénéfices notables. Un nombre de points problématiques ont été détectés, aussi bien des incohérences que des lacunes formelles, et nos propositions pour les résoudre correctement ont finalement été reçues dans le processus de normalisation. En définitive, nous tenons à fournir une sémantique formellement vérifiée par machine pour être incluse au sein du document standard d'OCL, c.-à-d., remplacer l'actuel Annexe A. Cet effort tend par la suite à stimuler le développement d'outils spécifique, vu qu'une sémantique clarifiée favorise le développement, par exemple, de schémas de compilations optimisées acceptant une logique OCL quatre valuées vers de récents solveurs SMT.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Construction de Logiques-Objet Sémantiquement Correct pour des Langages à Domaines Spécifiques Basés sur UML/OCL

Mots Clés : Structures de données orientés objets, Chemins d'expression, Featherweight OCL, Null, Invalid, Sémantique formelle, Isabelle, Réflexion, UML, OCL.

Résumé : Les langages de spécifications basés et orientés objets (comme UML/OCL, JML, Spec#, ou Eiffel) permettent la création et destruction, la conversion et tests de types dynamiques d'objets statiquement typés. Par dessus, les invariants de classes et les opérations de contrat peuvent y être exprimés; ces derniers représentent les éléments clés des spécifications orientées objets. Une sémantique formelle des structures de données orientées objets est complexe: des descriptions imprécises mènent souvent à différentes interprétations dans les outils qui en résultent.

Dans cette thèse, nous démontrons comment dériver un environnement de preuves moderne comme un *méta-outil* pour la définition et l'analyse de sémantique formelle de langages de spécifications orientés objets. Étant donné une représentation d'un langage particulier plongé en Isabelle/HOL, nous construisons pour ce langage un environnement étendu d'Isabelle, à travers une *méthode* de génération de code particulière, qui implique notamment plusieurs variantes de génération de code. Le résultat supporte l'édition asynchrone, la vérification de types, et les activités de déduction formelle, tous "hérités" d'Isabelle.

En application de cette méthode, nous obtenons un *outil de modélisation orienté objet* pour du UML/OCL textuel. Nous intégrons également des idiomes non nécessairement présent dans UML/OCL— en d'autres termes, nous développons un support pour des dialectes d'UML/OCL à domaine spécifique.

En tant que construction méta, nous définissons un méta-modèle d'une partie d'UML/OCL en HOL, un méta-modèle d'une partie de l'API d'Isabelle en HOL, et une fonction de traduction entre eux en HOL. Le méta-outil va alors exploiter deux procédés de générations de code pour produire soit du *code raisonnablement efficace*, soit du *code raisonnablement lisible*. Cela fournit donc deux modes d'animations pour inspecter plus en détail la sémantique d'un langage venant d'être plongé: en chargeant à vitesse réelle sa sémantique, ou simplement en retardant à un autre niveau "méta" l'expérimentation précédente pour un futur instant de typage en Isabelle, que ce soit pour des raisons de performances, de tests ou de prototypages.

Remarquons que la génération de "code raisonnablement efficace", et de "code raisonnablement lisible" incluent la génération de *code tactiques* qui prouvent une collection de théorèmes formant une théorie de types de données orientés objets d'un modèle dénotationnel: étant donné un modèle de classe UML/OCL, les preuves des propriétés pertinentes aux conversions, tests de types, constructeurs et sélecteurs sont traitées automatiquement. Cette fonctionnalité est similaire aux *paquets de théories de types de données* présents au sein d'autres prouveurs de la famille HOL, à l'exception que certaines motivations ont conduit ce travail présent à programmer des tactiques haut-niveaux en HOL lui-même.

Ce travail prend en compte les plus récentes avancées du standard d'UML/OCL 2.5. Par conséquent, tous les types UML/OCL ainsi que les types logiques distinguent deux éléments d'exception différents: `invalid` (exception) et `null` (élément non-existant). Cela entraîne des conséquences sur les propriétés aussi bien logiques qu'algébriques des structures orientées objets résultant des modèles de classes.

Étant donné que notre construction est réduite à une séquence d'extension conservative de théorie, notre approche peut garantir la correction logique du langage entier considéré, et fournit une méthodologie pour étendre formellement des langages à domaine spécifique.

Title: Constructing Semantically Sound Object-Logics for UML/OCL Based Domain-Specific Languages

Keywords: Object-oriented Data Structures, Path Expressions, Featherweight OCL, Null, Invalid, Formal Semantics, Isabelle, Reflection, UML, OCL.

Abstract: Object-based and object-oriented specification languages (like UML/OCL, JML, Spec#, or Eiffel) allow for the creation and destruction, casting and test for dynamic types of statically typed objects. On this basis, class invariants and operation contracts can be expressed; the latter represent the key elements of object-oriented specifications. A formal semantics of object-oriented data structures is complex: imprecise descriptions can often imply different interpretations in resulting tools.

In this thesis we demonstrate how to turn a modern proof environment into a *meta-tool* for definition and analysis of formal semantics of object-oriented specification languages. Given a representation of a particular language embedded in Isabelle/HOL, we build for this language an extended Isabelle environment by using a particular *method* of code generation, which actually involves several variants of code generation. The result supports the asynchronous editing, type-checking, and formal deduction activities, all "inherited" from Isabelle.

Following this method, we obtain an *object-oriented modelling tool* for textual UML/OCL. We also integrate certain idioms not necessarily present in UML/OCL— in other words, we develop support for domain-specific dialects of UML/OCL.

As a meta construction, we define a meta-model of a part of UML/OCL in HOL, a meta-model of a part of the Isabelle API in HOL, and a translation function between both in HOL. The meta-tool will then exploit two kinds of code generation to produce either *fairly efficient code*, or *fairly readable code*. Thus, this provides two animation modes to inspect in more detail the semantics of a language being embedded: by loading at a native speed its semantics, or just delay at another "meta"-level the previous experimentation for another type-checking time in Isabelle, be it for performance, testing or prototyping reasons.

Note that generating "fairly efficient code", and "fairly readable code" include the generation of *tactic code* that proves a collection of theorems forming an object-oriented datatype theory from a denotational model: given a UML/OCL class model, the proof of the relevant properties for casts, type-tests, constructors and selectors are automatically processed. This functionality is similar to the *datatype theory packages* in other provers of the HOL family, except that some motivations have conducted the present work to program high-level tactics in HOL itself.

This work takes into account the most recent developments of the UML/OCL 2.5 standard. Therefore, all UML/OCL types including the logic types distinguish two different exception elements: `invalid` (exception) and `null` (non-existing element). This has far-reaching consequences on both the logical and algebraic properties of object-oriented data structures resulting from class models.

Since our construction is reduced to a sequence of conservative theory extensions, the approach can guarantee logical soundness for the entire considered language, and provides a methodology to soundly extend domain-specific languages.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France