Certifications of programs with computational effects

Burak Ekici

UNIVERSITÉ GRENOBLE ALPES

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques et Informatique**

Arrêté ministériel : 7 Août 2006

Présentée par

Burak Ekici

Thèse dirigée par **Jean-Guillaume Dumas**
et par **Dominique Duval**

préparée au sein du **Laboratoire Jean Kuntzmann**
dans l'école doctorale **ED Mathématiques, Sciences et Technologies de l'Information, Informatique**

Certification de programmes avec des effets calculatoires

Thèse soutenue publiquement le **9 Décembre 2015**,
devant le jury composé de :

M. Jean François Monin
Professeur, Université Joseph Fourier, Président
Mme Catherine Dubois
Professeure, ENSIIE, Rapporteur
M. Olivier Laurent
Directeur de Recherche, CNRS, Rapporteur
M. Alan Schmitt
Directeur de Recherche, INRIA, Rapporteur
M. Andrej Bauer
Professeur, Université de Ljubljana, Examinateur
M. Damien Pous
Chercheur, CNRS, Examinateur
M. Jean-Guillaume Dumas
Professeur, Université Joseph Fourier, Directeur de thèse
Mme Dominique Duval
Professeure, Université Joseph Fourier, Co-Directrice de thèse

Annem, babam ve Çağın'a ...

Acknowledgements

When I started doing my PhD in Grenoble in early 2013, I was not able to imagine how enormous load of mathematical formalism I would be struggling with in the following three years: using the decorated logic to formalize computational effects, making the distinction between its syntax and semantics from the categorical viewpoint, proving properties of "effectful" programs using "complete theories" of decorated logics, certifying these proofs in Coq implementations, combining computational effects and all that. I ignorantly believed for a long while that all these can be done in a three-years PhD. I have to admit that it took me a lot to put things in a real basis. At the current stage, if I could make a bit of these things possible and get a thesis out of them, that is simply due to the endless support and effort of many people.

First of all, I would like to express my deepest gratitude to my supervisors Dr. Jean-Guillaume Dumas and Dr. Dominique Duval for their excellent guidance, tolerance and patience. Many thanks also for making reasonable amount of time whenever I knocked on your doors. This thesis would have never been possible without your constant encouragement. I am also grateful to Dr. Damien Pous who has guided many things related to the amazing proof assistant, Coq. Thanks a lot for always being constructive and answering all my (stupid) questions.

Each member of my thesis committee has shed light on non-apparent issues which helped me understanding thus clarifying things better: Dr. Catherine Dubois, Dr. Olivier Laurent and Dr. Alan Schmitt; thank you so much for spending time on my manuscript, returning corrections and writing reports that enabled me to defend. Dr. Andrej Bauer, Dr. Damien Pous and Dr. Jean-François Monin; many thanks for examining my thesis, making contributions by returning corrections/suggestions. I will always have the honor of having you in the jury. I have to explicitly thank Dr. Andrej Bauer once again for inviting me to lovely Ljubljana and giving me important voice and discussion opportunities. Many thanks are also going Dr. Matija Pretnar and Dr. Alex Simpson for explaining a lot of things related to algebraic handlers thus Eff, during my visit.

It is "usually" pretty painful to live abroad. Luckily enough, I have many friends who truly helped me feel less homesick. There is definitely no order/priority of thanksgiving but I have to admit that some of my friends used the Skype-like tools better: Tolga Aydın, Taşkın Duman, Recep Gündoğdu, Hakan Erdoğan, Bilge & Gürcan Gerçek, Ayşe & Erdem Sarılı, Gizem & Tunay Tuna, Hatice & Mustafa Hacıbekir, Bihter & Bora Yalçın, Başak-Esin & Burçin Güzel, Serap & Caner Canbulat, Oya & Efe Özelçinler, Burcu & Baran Aytaş, Pelin & Hikmet Taştan, Ayşe & Samet Gacaroğlu, Gülçin & Onur Tosun, Zeynep & Mitat Poyraz, Sevilay & Doguş Eyrek, Hayal & İlhan Öztürk, Şeyma & Batuhan Gündoğdu, Kemal Bulat, Çağatay Yücel, Görkem Kılınç, Seçkin Akın, Cevahir Altıntop, Halil Özcan, Ramazan Tuna and Mehmet Emrah Kala; thank you guys so much for always being in accompany.

I have spent three cool years in the city of Grenoble meeting many people with whom I had been in the same boat: Ziad, Jean-Baptiste, Federico Zertuche, Alexandre Aksenov, Pierre-Jean, Thomas, Euriell, Mohammad, Alexis, Patricia, Nhu, Konstantina, Cecilia, Irini, Anastasios, Nelson, Pierre-Olivier, Matthias, Chloé, Meryem, Abdel, Rémi, Alexandre Hoffmann, Charles, Lionel, Federico Pierucci and Dmitry; thanks a lot guys

for all unforgettable memories.

Last but definitely not least, I thank my mother Zeynep Ekici, my father Mehmet Ekici and my brother Çagin Ekici a lot for their true and sincere support even in the worst moments. The entire work is dedicated to each of them individually!

Burak Ekici, Grenoble, December 2015.

"Pure mathematics is, in its way, the poetry of logical ideas."

Albert Einstein.

"Mathematics is the art of giving the same name to different things."

Henri Poincaré.

"Theoretical computer science is closer to mathematics than it is to computer science. There are definitions, theorems and proofs."

Andrej Bauer.

Certification of Programs with Computational Effects
Burak Ekici
LJK, University Joseph Fourier
Grenoble, France

Abstract

In this thesis, we aim to formalize the effects of a computation. Indeed, most used programming languages involve different sorts of effects: state change, exceptions, input/output, non-determinism, etc. They may bring ease and flexibility to the coding process. However, the problem is to take into account the effects when proving the properties of programs. The major difficulty in such kind of reasoning is the mismatch between the syntax of operations with effects and their interpretation.

Typically, a piece of program with arguments in X that returns a value in Y is not interpreted as a function from X to Y, due to the effects. The best-known algebraic approach to the problem interprets programs including effects with the use of *monads*: the interpretation is a function from X to $T(Y)$ where T is a monad. This approach has been extended to *Lawvere theories* and *algebraic handlers*. Another approach called, the *decorated logic*, provides a sort of equational semantics for reasoning about programs with effects.

We specialize the approach of decorated logic to the state and the exceptions effects by defining *the decorated logic for states* ($\mathcal{L}_{st}$) and *the decorated logic for exceptions* ($\mathcal{L}_{exc}$), respectively. This enables us to prove properties of programs involving such effects. Then, we formalize these logics in Coq and certify the related proofs. These logics are built so as to be sound. In addition, we introduce a relative notion of syntactic completeness of a theory in a given logic with respect to a sublogic. We prove that the decorated theory for the global states as well as two decorated theories for exceptions are syntactically complete relatively to their pure sublogics. These proofs are certified in Coq as applications of our generic frameworks.

Keywords: computational effects, states, exceptions, program property proofs, equational semantics, decorated logic, proof certification, Coq.

Certification de programmes avec des effets calculatoires
Burak Ekici
LJK, Université Joseph Fourier
Grenoble, France

Résumé

Dans cette thèse, nous visons à formaliser les effets calculatoires. En effet, les langages de programmation les plus utilisés impliquent différentes sortes d'effets de bord: changement d'état, exceptions, entrées / sorties, non-déterminisme, etc. Ils peuvent apporter facilité et flexibilité dans le processus de codage. Cependant, le problème est de prendre en compte les effets lorsque l'on veut prouver des propriétés de programmes. La principale difficulté dans ce genre de preuve de programmes est le décalage entre la syntaxe des opérations avec effets de bord et leur interprétation.

Typiquement, un fragment de programme avec des arguments de type X qui retourne une valeur de type Y n'est pas interprété comme une fonction de X vers Y, à cause des effets. L'approche algébrique la plus connue pour ce problème permet une interprétation des programmes, y compris ceux comportant des effets, en utilisant des monades : l'interprétation est une fonction de X vers $T(Y)$ où T est une monade. Cette approche a été étendue aux théories de Lawvere et aux "gestionnaires algébriques" (*algebraic handlers*). Une autre approche, appelée logique décorée, fournit une sémantique équationnelle pour ces programmes.

Nous spécialisons l'approche de la logique décorée pour les effets liés à l'état de la mémoire et à la gestion des exceptions en définissant *la logique décorée pour les états* ($\mathcal{L}_{st}$) et *la logique décorée pour les exceptions* ($\mathcal{L}_{exc}$), respectivement. Elles nous permettent de prouver des propriétés de programmes impliquant de tels effets. Ensuite, nous formalisons ces logiques en Coq et certifions les preuves associées. Ces logiques sont construites de manière à être correctes. En outre, nous introduisons une notion de complétude syntaxique relative d'une théorie dans une logique donnée par rapport à une sous-logique. Nous montrons que la théorie décorée pour les états globaux ainsi que deux théories décorées pour les exceptions sont relativement complets relativement à leur sous-logique pure. Non seulement nous pouvons utiliser le système développé pour prouver des programmes comportant des effets, mais également nous utilisons cette formalisation pour certifier les résultats de complétude obtenus.

Mots-clés : effets calculatoires, état, exceptions, preuves de programmes, sémantique équationnelle, logique décorée, certification de programmes, Coq.

Hesaplama yan etkileri içeren programların özellik sertifikasyonu
Burak Ekici
LJK, Joseph Fourier Üniversitesi
Grenoble, France

Özet

Bu tez programlama yan etkileri sertifikasyonunu formalize etmeyi hedefler. Günümüzde yaygın olarak kullanılan programlama dilleri değişik yan etkiler içerirler: bellek durumu (state), istisnai durumlar (exceptions), girdi/çıktı (I/O), deterministik olmayan durumlar (non-determinisn) v.b. Bütün bunlar kodlama evresinde kullanıcıya kolaylıklar sağlasa da program özelliklerinin formal olarak ispatlanmasına ve/veya bu özellikler hakkında çıkarım yapılmasına zorluklar çıkartırlar. Bu bağlamdaki en önemli zorluk program sintaksı ve izahı (interpretation) arasındakı uyumsuzluktur.

Tipik olarak, girdi argumanları X kümesinden gelen ve sonuç olarak Y kümesi elemanlarını döndüren bir program içerdiği yan etkilerden ötürü X'den Y'ye tanımlı bir fonksiyon olarak izah edilmez. Bu konudaki en yaygın kullanılan yaklaşım yan etki içeren programları X'den $T(Y)$'ye tanımlı bir fonskyion olarak izah eder öyle ki T kategori teorik bir *monad*dır. Ayrıca, bu yaklaşım *Lawvere theories* ve *algebraic handlers* gibi farklı yaklaşımlara da genişletilmiştir. Bir diğer yaklaşım ise *decorated logic* olarak adlandırılır ve yan etki içeren programlar arasında bir tür eşitlemeli çıkarım (equational reasoning) imkanı sağlar.

Bu tezde biz *decorated logic* yaklaşımını bellek durumu (state) ve istisnai durumlar (exceptions) yan etklieri için özelleştiriyoruz ve bu formal mantıksal sistemlere de *the decorated logic for states* ($\mathcal{L}_{st}$) ve *the decorated logic for exceptions* ($\mathcal{L}_{exc}$) diyoruz. Bu şekilde sözü geçen programlama yan etkilerini içeren program özelliklerini eşitlemeli olarak ispatlama imkanı sağlıyoruz. Daha sonra sözü geçen formal sistemleri Coq ispat asistanı kullanarak formalize ediyor ve bu ispatları sertifikalandırıyoruz. Bu formal yaklaşımlar geçerli (sound) olarak dizayn edilmişlerdir fakat tamamlılıkları (completeness) hakkında hemen bir kanıya varmak zordur. Bu bağlamda, verilen bir teorinin bir alt mantıksal sisteme (sublogic) göre olan bir çeşit sinatktik tamamlılığının (syntactic completeness) tanımı yapıyoruz. Daha sonra, bu tanımı kullanarak $\mathcal{L}_{st}$ ve $\mathcal{L}_{exc}$'in kendi alt salt (pure) mantıklarına göre olan sintaktik tamamlılıklarını ispatlıyor ve bu ispatı da Coq ispat asistanını kullanarak sertifikalandırıyoruz.

Anahtar kelimeler: hesaplama yan etkileri, bellek durumu, istisnai durumlar, program özellik ispatları, eşitlemeli anlam bilimi, decorated logic, ispat sertifikasyonu, Coq.

Online sources

This thesis comes with some Coq sources that are available online:

- The `STATES-THESIS` library:
  `https://forge.imag.fr/frs/download.php/695/STATES-THESIS.tar.gz`
- The `EXCEPTIONS-THESIS` library:
  `https://forge.imag.fr/frs/download.php/694/EXCEPTIONS-THESIS.tar.gz`
- The `HPC-THESIS` library:
  `https://forge.imag.fr/frs/download.php/696/HPC-THESIS.tar.gz`

Notice that the `EXCEPTIONS-THESIS` library includes the logics both for the core language and the one for the programmers' language as well as the translation of the programmers' language into the core language.

Proof lengths & Benchmarks				
library	source	length in Coq	length in LaTeX	execution time in Coq
`STATES-THESIS`	Proofs.v	12 KB	20 KB	4.806 sec.
`EXCEPTIONS-THESIS`	Proofs.v	8 KB	24 KB	3.256 sec.

The `HPC-THESIS` package includes three different libraries:

(1) `exc_cl-hp`: Hilbert-Post completeness of the base language (core language with no use of categorical coproducts) of exceptions.

(2) `exc_pl-hp`: Hilbert-Post completeness of the programmers' language for exceptions.

(3) `st_hp`: Hilbert-Post completeness of the base language (core language with no use of categorical products) of the state.

Proof lengths & Benchmarks				
library	source	length in Coq	length in LaTeX	execution time in Coq
`exc_cl-hp`	HPCompleteCoq.v	36 KB	28 KB	4.600 sec.
`exc_pl-hp`	HPCompleteCoq.v	8 KB	8 KB	0.988 sec.
`st-hp`	HPCompleteCoq.v	36 KB	32 KB	5.979 sec.

Remark 0.0.1. Above measurements have been performed on a Intel i7-3630QM @2.40GHz machine running the Coq Proof Assistant, v. 8.4pl3.

Contents

List of Figures

1
Introduction

1.1 Motivation of the thesis

Software may involve mistakes that are difficult to detect. One of the current strategies to detect possible mistakes in a given software is to run series of tests and hopefully to figure out the possibly incorrect program behaviors. However, visiting all possible cases is definitely out of testing scope. Hence, testing seems unsatisfactory especially when the software in question is critical. For instance, software systems that are used to exchange secure information, or the ones used in aviation and automotive industries. In order to ensure that a software system is error-free, one needs mathematical formalization and proofs.

The choice of mathematical formalization depends on the notions which are used in a certain software. For instance, if it is implemented in a purely functional manner based on simply typed λ-calculus, then the formalization can be done using cartesian closed categories and properties can be proved within that context. If it involves any sort of outside world interaction (so called computational effect), then it definitely needs a better care. In this case, the choice of formalization has a range: varying from the use of monads to decorated logic, that are briefly presented in Section 2.

1.2 The goal

A computational effect is said to be the apparent mismatch between syntax and semantics of a program. In this dissertation, we separately formalize the global state effect in Chapter 5 and the exceptions effect in Chapter 6 with the decorated logic [DD10]. The latter is mainly presented in Chapter 4. Then, by using these formalizations, we prove primitive program properties including mentioned effects. In addition, we implement these formal treatments in the Coq proof assistant and certify related program property proofs. The inference systems provided by the formal approaches are designed to be sound and their base languages (without categorical structures such as products and coproducts) are here proven to be Hilbert-Post complete.

1.3 Contributions

This thesis comes with the following contributions to state of the art:

(1) the implementations of the decorated logics in the Coq Proof Assistant to verify computational effects arising from a comonad and a monad: the decorated logic for a comonad ($\mathcal{L}_{com}$) and the decorated logic for a monad ($\mathcal{L}_{mon}$),

(2) the formalizations of the state and exception effects through the decorated logic:

(2.1) the decorated logic for the state ($\mathcal{L}_{st}$) as an extension to $\mathcal{L}_{com}$;

(2.2) the decorated logic for exceptions ($\mathcal{L}_{exc}$) and the decorated logic for the programmers' language for exceptions ($\mathcal{L}_{exc-pl}$) as extensions to $\mathcal{L}_{mon}$;

(2.3) the Coq implementations of these logics as applications to item (1),

(3) the Hilbert-Post completeness proofs of the logics $\mathcal{L}_{st}$ without products, $\mathcal{L}_{exc}$ without coproducts and $\mathcal{L}_{exc-pl}$, as well as related proof certifications in Coq as applications to item (2.3).

1.4 Publications

Below, we list the publications and reports that have been produced during this thesis:

Refereed conference papers

[DDE+15] Relative Hilbert-Post completeness exceptions.
Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, Damien Pous and Jean-Claude Reynaud. In Siegfried Rump and Chee Yap, editors, *MACIS 2015, Sixth International Conference on Mathematical Aspects of Computer and Information Sciences*, 2015.

[DDER14] Certified proofs in programs involving exceptions.
Jean-Guillaume Dumas, Dominique Duval, Burak Ekici and Jean-Claude Reynaud.,
CICM 2014 : Eigth Conference on Intelligent Computer Mathematics, Coimbra, Portugal, 7–11 July 2014, CEUR Workshop Proceedings, no 1186, paper 20.

[DDEP14] Formal verification in Coq of program properties involving the global state effect. Jean-Guillaume Dumas, Dominique Duval, Burak Ekici and Damien Pous.,
JFLA 2014 : Journées Francophones des Langages Applicatifs, Fréjus, France, 8–11 January 2014.

Research reports

[Eki15] IMP with exceptions over decorated logic.
Burak Ekici.,
Pre-proceedings of TFP 2015: Trends in Functional Programming, Sophia-Antipolis, France, 3–5 June 2015.

1.5 Content of the thesis

In the following, we describe the content of the thesis by highlighting the main contents of each chapter:

- Chapter 2 is the state of the art chapter where we start in Section 2.1, by introducing some of the existing approaches to formalize computational effects. This is followed by the presentation of different software tools, either to handle computational effects with Haskell and Eff in Section 2.2, or to verify properties of programming languages with effects with Idris, Coq or Isabelle in Section 2.3. Most references to the related work will appear in this section.

- In Chapter 3, first we separately study the Kleisli and coKleisli adjunctions associated to a monad and a comonad in Section 3.1. Then, Section 3.2 starts with the

proof of the comparison theorem for the coKleisli construction [ML71, Ch. VI, §5, dual of Theorem 2] and continues with the composition of Kleisli and coKleisli adjunctions: such a two-level structure is named the *coKleisli-on-Kleisli construction associated to a monad* and studied in detail with an application to the exceptions monad. In Chapter 6 we will make use of this construction to interpret the *decorated logic for the exception effect*. Finally, Section 3.3 dualizes the construction introduced in Section 3.2. There, we first give the comparison theorem for the Kleisli construction [ML71, Ch. VI, §5, Theorem 2] and proceed with the composition of coKleisli and Kleisli adjunctions, yielding the *Kleisli-on-coKleisli construction associated to a comonad*. We apply this composition to the states comonad, so as to interpret the *decorated logic for the state effect* in Chapter 5. The main result of this chapter is that the coKleisli-on-Kleisli category of a monad and the Kleisli-on-coKleisli category of a comonad are proven to be respectively the full image category of the related monad and comonad endofunctors (Theorems 3.2.5 and 3.3.4).

- In Chapter 4, Section 4.1 defines the *monadic equational logic* $\mathcal{L}_{meq}$. This logic is extended into the *decorated logic for a monad* ($\mathcal{L}_{mon}$) in Section 4.2, where the categorical interpretation of $\mathcal{L}_{mon}$ by the coKleisli-on-Kleisli construction associated to a monad is also given. In Section 4.3, the decorated logic for a comonad ($\mathcal{L}_{com}$) is detailed. There, we use the Kleisli-on-coKleisli construction associated to a comonad to interpret the logic $\mathcal{L}_{com}$. The Coq implementation of both logics is given in Section 4.4. These logics have been built so as to be sound with respect to their intended categorical interpretation; but little is known about their completeness. Therefore, in Section 4.5, we conclude with a completeness notion: *relative Hilbert-Post completeness* which is well-suited to a decorated logic. We will show in Sections 5.4 and 6.9 that one *decorated logic for the state effect* and two *decorated logics for the exception effect* are Hilbert-Post complete with respect to their pure sublogics: we adapt the theorem in [Sta10, Th 5] to our logics to give a decorated proof of their completeness.

- In Chapter 5, we start, in Section 5.1, with the syntax of the *decorated logic for the state* ($\mathcal{L}_{st}$) with its interpretation given via the Kleisli-on-coKleisli construction associated to the states comonad. The Coq implementation of the logic $\mathcal{L}_{st}$ is presented in Section 5.2. In Section 5.3, we prove some properties of the state effect as in [PP02, §3], but here in a decorated setting. Lastly, the logic $\mathcal{L}_{st}$ (without products) is proven to be relatively *Hilbert-Post complete* in Section 5.4.

- In Chapter 6, we start, in Section 6.1, with the *decorated logic for the exception* ($\mathcal{L}_{exc}$) with its interpretation given through the coKleisli-on-Kleisli construction associated to the exceptions monad. We present the *decorated logic for the programmers' language for exceptions* ($\mathcal{L}_{exc-pl}$) in Section 6.2, with its interpretation via the Kleisli adjunction associated to the exceptions monad. The translation of the logic $\mathcal{L}_{exc-pl}$ into the logic $\mathcal{L}_{exc}$ is given in Section 6.3. The Coq implementations of the logics $\mathcal{L}_{exc}$ and $\mathcal{L}_{exc-pl}$ and the translation of the logic $\mathcal{L}_{exc-pl}$ into the logic $\mathcal{L}_{exc}$ are respectively presented in Sections 6.4, 6.5 and 6.6. We prove some properties of the exceptions effect in a decorated setting in Section 6.7. The logic $\mathcal{L}_{exc-pl}$, as well as the logic $\mathcal{L}_{exc}$ without coproducts, are proven to be *relatively Hilbert-Post complete* in Sections 6.8 and 6.9.

- The Chapter 7 is the concluding chapter where we give an overview of the results obtained in this thesis and highlight some potential future research directions.

2

About computational effects

In programming languages theory, a program is said to have *computational effects* if, besides a return value, it has observable interactions with the outside world. For instance, using/modifying the program state, raising/recovering exceptions, reading/writing data from/to some file, etc. In order to formally reason about behaviors of a program with *computational effects*, one has to take into account these interactions with the outside world. One difficulty in such a study is the mismatch between the syntax of operations with effects and their interpretation. Typically, an operation in an effectful language with arguments in X that returns a value in Y is not interpreted as a function from X to Y, due to the effects, unless the operation is pure.

In this chapter, we start, in Section 2.1, by introducing some of the existing approaches to formalize computational effects. This is followed by the presentation of different software tools, either to handle computational effects with Haskell and Eff in Section 2.2, or to verify properties of programming languages with effects with Idris, Coq, or Isabelle in Section 2.3. Finally, in Section 2.4, we compare the formal approach behind this thesis with the existing ones.

2.1 Formal approaches

The simply typed λ-calculus is a useful mathematical tool to study behaviors of typed programming languages without computational effects. It can be interpreted in a cartesian closed category with types as objects and terms (program pieces) as arrows. In addition, categorical products and coproducts can be used to cope with n-ary operations and conditionals (or branching), respectively. This result is known as the Curry-Howard-Lambek correspondence which relates intuitionistic logic, simply typed lambda calculus and cartesian closed categories. The *algebraic approach* for formalizing computational effects aims to extend this correspondence in a formal way. This has been considered from several different viewpoints and used to formalize computational effects as detailed in Sections 2.1.2, 2.1.4 and 2.1.5. Some alternative approaches such as effect systems and decorated logic are also briefly presented in Sections 2.1.1 and 2.1.6.

2.1.1 Effect systems

In their 1988 paper [LG88], Lucassen and Gifford presented a new approach to programming languages for parallel computers. The key idea was to use an *effect system* to discover expression scheduling constraints. In this system, every expression comes with three components: *types* to represent the kinds of the return values, *effects* to summarize the observable interactions of expressions and *regions* to highlight the areas of the memory where expressions may have effects. To this extend, one can simply reason that if two expressions do not have overlapping effects, then they can obviously be scheduled

in parallel. The reasoning is done by some inference rules for *types* and *effects* based on the second order typed λ-calculus.

2.1.2 Effects as monads

The best known *algebraic approach* to formalize computational effects was initiated by Moggi in his seminal paper [Mog91]. There, he showed that the effectful operations of an impure language can be interpreted as arrows of a Kleisli category for an appropriate monad (T, η, μ) over a base category $\mathscr{C}$ with finite products (see Definition 3.1.2). For instance, in Moggi's *computational metalanguage*, an operation in an impure language with arguments in X that returns a value in Y is now interpreted as an arrow from $[\![X]\!]$ to $T[\![Y]\!]$ in $\mathscr{C}$ where $[\![X]\!]$ is the object of *values* of type X and $T[\![Y]\!]$ is the object of *computations* that return values of type Y. The use of monads to formalize effects (such as state, exceptions, input/output and non-deterministic choice) was popularized by Wadler in [Wad92] and implemented in the programming languages Haskell (See Section 2.2.1) and F♯.

With this, through monad transformers [Jas09], it is usually possible to "combine" different effects formalized by monads. In [GSR14], Goncharov et al. proposed a framework that combines monad-based computational effects, underdefined or free operations and recursive definitions.

Example 2.1.1. The *exceptions monad* (or *coproduct monad*), on the category of sets, comes with the endofuctor $TX = X + E$ for each set X and the distinguished set of exceptions E. Note that we use the exceptions monad in Section 6 to formalize the exception effect.

Example 2.1.2. The *state monad*, on the category of sets, has the endofunctor $TX = S \to (X \times S)$ for each set X and the distinguished set of states S.

Moggi's *computational metalanguage* was extended into the *basic effect calculus* with a notion of *computation type* as in Filinski's effect PCF [Fil96] and in Levy's call-by-push-value (CBPV) [Lev99]. In their paper [EMS14], Egger at al., defined their effect calculus, named *extended effect calculus* as a canonical calculus incorporating the above ideas of Moggi, Filinski and Levy. Following Moggi, they included a type constructor for computations. Following Filinski and Levy, they classified types as value types and computation types.

2.1.3 Effects as comonads

Being dual to monads, comonads have been used to formalize context-dependent computations. Intuitively, an effect which observes features may arise from a comonad, while an effect which constructs features may arise from a monad [JR11]. Uustalu and Vene have structured stream computations [UV08], Orchard et al. array computations [OBM10] and Tzevelekos game semantics [Tze08] via the use of comonads. In [POM], Petricek et al. proposed a unified calculus for tracking context dependence in functional languages together with a categorical semantics based on indexed comonads. In his report [Orc12], Orchard proposed a method for choosing between monads and comonads when formalizing computational effects.

A computation can be seen as a composition of context-dependence and effectfulness [UV08]. In [BVS93], Brookes and Van Stone showed that such combinations may correspond to distributive laws of a comonad over a monad. This has been applied to clocked causal dataflow computation, combining causal dataflow and exceptions by Uustalu and Vene in [UV05].

Example 2.1.3. The *costate comonad*, on the category of sets, is given with the endofunctor $DX = S \times (S \to X)$ for each set X and the distinguished set of states S.

Example 2.1.4. The *state comonad* (or *product comonad*), on the category of sets, is given with the endofunctor $DX = X \times S$. Note that, we use the state comonad to formalize the global state effect (See Section 5) while Moggi uses the state monad (as in Example 2.1.2) for the same effect [Mog91].

2.1.4 Effects as Lawvere theories

A *Lawvere theory* is a finite product category in which every object is isomorphic to a finite cartesian power

$$A^n = \underbrace{A \times A \times \ldots \times A}_{\text{n times}}$$

of a distinguised object A known as the *generator*. A morphism $f\colon A^n \to A^m$ in a Lawvere theory may be expressed as $f\colon m \to n$.

Lawvere theories first appeared in Lawvere's 1963 PhD dissertation [Law63]. Three years later, in [Lin66], Linton showed that every Lawvere theory induces a monad on the category of sets and on any category which satisfies the local representability condition [Lin69]. Therefore, Moggi's seminal paper [Mog91], formalizing computational effects by monads, made it possible for monadic effects to be formalized through Lawvere theories. To this extend, Plotkin and Power, in [PP02], have shown that effects such as the global and the local state could be formalized by *signatures* of effectful terms and an *equational theory* explaining the interaction among terms.

A *model* of a Lawvere theory $\mathcal{L}$ in a category $\mathscr{C}$, with finite products, is a finite-product preserving functor $M\colon \mathcal{L} \to \mathscr{C}$. A homomorphism between models M_1 and M_2 is a natural transformation $h\colon M_1 \Rightarrow M_2$. All models of the Lawvere theory $\mathcal{L}$, with the model homomorphisms, form a category $Mod_{\mathcal{L}}(\mathscr{C})$ called the *model category of* $\mathcal{L}$. The model category $Mod_{\mathcal{L}}(\mathscr{C})$ is equipped with a forgetful functor $U\colon Mod_{\mathcal{L}}(\mathscr{C}) \to \mathscr{C}$. It can be proven that U has a left adjoint $F\colon \mathscr{C} \to Mod_{\mathcal{L}}(\mathscr{C})$, if $\mathscr{C}$ is a locally finitely presentable category [Bor94]. This adjunction $F \dashv U\colon Mod_{\mathcal{L}}(\mathscr{C}) \to \mathscr{C}$ induces a monad on the base category $\mathscr{C}$. Now, a computational effect is called *algebraic*, if it corresponds to a monad which can be obtained from a Lawvere theory [PP13].

In [HPP06] and [HLPP07], Hyland et al. studied the combination of computational effects in terms of Lawvere theories.

Example 2.1.5. [HP07] The Lawvere theory $\mathcal{L}_E$ for exceptions is generated by $Card(E)$ constant operations (one for each exception e in E) $\mathtt{raise}_e\colon \mathrm{A}^0 \to \mathrm{A}^1$ with no equation [PP03]. The monad $TX = X + E$ on the category of sets is induced by the theory $\mathcal{L}_E$ as follows: Each model M is characterized by the set $B = M(A)$ and the elements $r_e \in B$ such that $M(\mathtt{raise}_e)\colon \{\star\} \to B$ maps $\star$ to r_e, for each e in E. Now, the forgetful functor $U\colon M_{\mathcal{L}_E}(\mathscr{S}et) \to \mathscr{S}et$ maps the model $M = (B, (r_e)_{e\in E})$ to the set B. The left adjoint $F\colon \mathscr{S}et \to M_{\mathcal{L}_E}(\mathscr{S}et)$ maps each set X to the model $(X + E, (e)_{e\in E})$ of $\mathcal{L}_E$ in $\mathscr{S}et$.

Furthermore, it has been shown in [PP03] that the operation $\mathtt{handle}_e$, used for recovering from an exception e, does not satisfy the requirements to be *algebraic* in the sense of [PP01, §2, Definition 2.1] while $\mathtt{raise}_e$ is an algebraic operation in this sense.

Example 2.1.6. [HP07] Let Loc be a finite set of locations and let Val be countable set of values. The countable Lawvere theory $\mathcal{L}_S$ for the state $S = \mathrm{Val}^{\mathrm{Loc}}$ is generated by the operations $\texttt{lookup}\colon \mathrm{Loc} \to \mathrm{Val}$ and $\texttt{update}\colon \mathrm{Loc} \times \mathrm{Val} \to 1$ satisfying seven equalities stated in [PP02, §3]. Along the same lines, Plotkin and Power have shown that the theory $\mathcal{L}_S$ induces the state monad $TX = S \to (X \times S)$ on the category of sets.
Melliès has shown in [Mel10] that some of these seven equalities can be omitted.

2.1.5 Handlers for algebraic effects

Plotkin and Pretnar [PP09, PP13] gave an account for handling algebraic effects: Moggi's classification of terms (*values* and *computations*) is extended with a third level called *handlers* within the framework of Lawvere theories. Here, we focus on exception handlers.

Simple exception handling construct. Let E be the set of exceptions and A be the set of values returned by the computations. Then:

$$H(M) \stackrel{\text{def}}{=} M\,\texttt{handled with}\,\{\texttt{raise}_e \mapsto M_e\}_{e\in E}$$

is the simple handling construct which is itself a computation. The handling construct $H(M)$ is made of a computation $M \in A + E$ and a handler $\{\texttt{raise}_e \mapsto M_e\}_{e\in E}$ where, for each exception $e \in E$, $\texttt{raise}_e$ intercepts M by throwing an exception and $M_e \in A + E$ is the corresponding predefined computation that handling construct proceeds with. There is no equation in the Lawvere theory of exceptions $\mathcal{L}_E$ (Example 2.1.5) but the handling computation is characterized by the following equations:

$$\begin{aligned} H\big(\texttt{return}\,V\big) &= \mathit{inl}(V), \\ H\big(\texttt{raise}_{e'}()\big) &= M_{e'}. \end{aligned}$$

where V is any value in A. Obviously, $M_{e'} = \texttt{raise}_{e'}()$ causes the exception of name e' not to be handled.

Extended exception handling construct. Simple handling construct is generalized to the *extended handling construct*, introduced by Benton and Kennedy [BK01], where returned values are passed to a user-defined continuation map $N\colon A \to B + E$.

$$H(M) \stackrel{\text{def}}{=} M\,\texttt{handled with}\,\{\texttt{raise}()_e \mapsto N_e\}_{e\in E}\,\texttt{to}\,x : A.N(x)$$

Within the handling construct, first the computation $M \in A + E$ is evaluated: if it returns a value $V \in A$, then the return value is bound to the variable x and it remains to evaluate $N(V) \in B + E$. Else if M raises the exception $e' \in E$, then the exception is recovered and the computation $N_{e'} \in B + E$ is evaluated. Obviously, $N_{e'} = \texttt{raise}_{e'}()$ causes the exception e' not to be handled. All these are characterized by the following equalities:

$$\begin{aligned} H\big(\texttt{return}\,V\big) &= N(V), \\ H\big(\texttt{raise}_{e'}()\big) &= N_{e'}. \end{aligned}$$

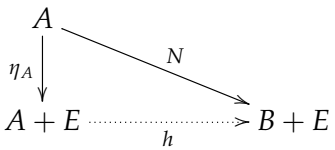


Figure 2.1: Interpreting the extended handling construct

As a further step, a calculus extending Levy's call-by-push-value (CBPV) paradigm [Lev99] (which is based on the distinction between computation and value types) with operations, handler definitions and effect handling constructs has been proposed not only to handle exceptions but also other sort of algebraic effects such as stream redirection, explicit non-determinism, CCS, parameter passing, timeout and rollback [PP13, §3].

Remark 2.1.7. The issue of exception handling has been circumvented in [SM04] in order to get a Hoare logic for exceptions and in [Lev06] by using both algebras and coalgebras. The formalization of exceptions can also be made from a coalgebraic point of view [Jac01].

2.1.6 Decorated Logic

In 2010, Duval and Dominguez [DD10] have proposed yet another paradigm to formalize computational effects by mixing effect systems and algebraic theories, named *the decorated logic*. The key point of this paradigm is that every term comes with a decoration which exposes its features with respect to a single computational effect or to several ones. In addition, an *equational theory* highlights the interactions among terms. There, we have two sorts of equations: *weak* equations relate terms with respect only to their results and *strong* equations relate terms with respect both to their results and effects.

Here, we prefer not to give details of the decorated logic, since in the following chapters, we detail them to cover the state and the exception effects. Instead, we give two examples, in the following, to clarify the use of decorations and equations, mainly the weak equations.

Example 2.1.8. (State effect) [DDFR12a, §1.2] In the object-oriented language `C++`, we write a class `BankAccount` as a toy example to manage simplified bank accounts with the use of types `int` and `void` that are respectively interpreted as the set of integers $\mathbb{Z}$ and a singleton $\{\star\}$. In the `BankAccount` class, there are two methods namely `balance()` and `deposit(x)`. The former returns the balance of the account without modifying it while the latter modifies the amount. Therefore, `balance` is an *accessor* method while `deposit(x)` is a *modifier*. We declare these public methods in `C++` syntax within the class `BankAccount` as follows:

```
class BankAccount {
  public:
        int balance () const;
        void deposit(int);
}
```

This syntax can be translated into a signature $Bank_{app}$, called *apparent signature*, as:

$$Bank_{app}\colon \begin{cases} \texttt{balance}\colon \texttt{void} \to \texttt{int} \\ \texttt{deposit}\colon \texttt{int} \to \texttt{void} \end{cases}$$

with the following interpretation:

$$\begin{cases} [\![\texttt{balance}]\!]\colon \{\star\} \to \mathbb{Z} \\ [\![\texttt{deposit}]\!]\colon \mathbb{Z} \to \{\star\} \end{cases}$$

which is obviously not the intended interpretation.

In order to get the intended interpretation correctly, one needs to consider the state of any account. Let `state` be the type of states of a given bank account. So that we can get the below signature (named *explicit signature*) as:

$$Bank_{expl}: \begin{cases} \texttt{balance}: \texttt{state} \to \texttt{int} \\ \texttt{deposit}: \texttt{int} \times \texttt{state} \to \texttt{state} \end{cases}$$

with the following interpretation:

$$\begin{cases} [\![\texttt{balance}]\!]: \texttt{St} \to \mathbb{Z} \\ [\![\texttt{deposit}]\!]: \mathbb{Z} \times \texttt{St} \to \texttt{St} \end{cases}$$

where `St` is the set of all possible states of an account.

The *apparent signature* is simple and close to the syntax but its interpretation is not the intended one. Contrarily, the *explicit signature* has the intended interpretation but it is far from the syntax itself. At this point, we define a *decorated signature* $Bank_{deco}$ which is close to the syntax and provides the intended interpretation. The decorated signature classifies operations by using superscripts: (0) is used to indicate *pure* operations that have no interaction with the state of the account, we have (1) for state *accessor* and (2) for state *modifier* operations.

$$Bank_{deco}: \begin{cases} \texttt{balance}^{(1)}: \texttt{void} \to \texttt{int} \\ \texttt{deposit}^{(2)}: \texttt{int} \to \texttt{void} \end{cases}$$

Let us consider the following `C++` expressions:

$$\texttt{deposit(10); balance()} \text{ and } \texttt{10 + balance()}$$

which can be seen in decorated terms as:

$$\texttt{balance}^{(1)} \circ \texttt{deposit}^{(2)} \circ \texttt{10}^{(0)} \text{ and } +^{(0)} \circ \langle \texttt{10}^{(0)}, \texttt{balance}^{(1)} \rangle$$

where the operator $+: \texttt{int} \times \texttt{int} \to \texttt{int}$ is pure. Obviously, these two terms have different effects: the former one is a *modifier* while the latter is only an *accessor* with respect to the state of an account. However both return the same integer as a result. Thus, for these cases where operations have the same result but make different manipulations on the state, we introduce a weak relation denoted by the symbol $\sim$:

$$\texttt{balance}^{(1)} \circ \texttt{deposit}^{(2)} \circ \texttt{10}^{(0)} \sim +^{(0)} \circ \langle \texttt{10}^{(0)}, \texttt{balance}^{(1)} \rangle$$

Example 2.1.9. (Exceptions effect) Similarly, it is possible to provide a decorated signature for the exceptions effect: (0) is to indicate *pure* operations, (1) for exception *throwers* and (2) for exception *catchers*. Weak equations relate terms that agree on ordinary arguments but maybe not on exceptional ones.

Therefore, the extended handling in Figure 2.1 can be seen as the interpretation of the following weak equation: $h^{(2)} \sim N^{(1)}$.

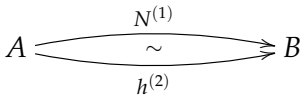


Figure 2.2: Extended handling construct in the decorated logic for exceptions

Indeed, this equation means that the catcher term $h^{(2)}$ agrees with the propagator term $N^{(1)}$ on ordinary arguments. However, on exceptional arguments, they may behave differently: $h^{(2)}$ might recover the computation from the exceptional argument while $N^{(1)}$ must propagate the exceptional argument.

More precisely, the catcher term $h^{(2)}\colon A \to B$ is interpreted as a function $h\colon A + E \to B + E$ while the propagator term $N^{(1)}$ is interpreted as a function $N\colon A \to B + E$. Then, the weak equation in Figure 2.2 is interpreted by the following equality: $N = h \circ \eta_A$, which is the way the extended handling construct in [PP13] is interpreted (Figure 2.1). Actually, the 3-tier system classifying the terms with respect to their exceptional features (pure, thrower, catcher) corresponds to the values, computations and handlers in [PP13].

Notice that in Chapters 5 and 6, we will propose decorated formalizations for the global state and the exceptions effects, respectively.

2.2 Software tools

The formal approaches mentioned in Sections 2.1.2 and 2.1.5 have been implemented in Haskell and Eff as briefly introduced in Sections 2.2.1 and 2.2.2.

2.2.1 Haskell

Haskell is a purely functional and lazy programming language: an expression would return exactly the same result when evaluated twice and an expression is evaluated only when it should return a final result. Haskell proposes a strongly typed system with some sophisticated features like typeclasses and generalized algebraic data types.

Monads are implemented in Haskell to make use of imperative features in a functional setting. In this context, the first attempt was made to perform input/output operations: reading/writing from/to a file have been implemented as monadic operations to impose an order of evaluation. However, the use of monads in Haskell is not limited to input/output. They also support some other imperative features such as state, exceptions, continuations, non-determinism, parallelization etc.

Definition 2.2.1. `Hask` is the category with objects as Haskell types, functions as arrows between these types. The identity arrow for any type A is given as

```
id = \x -> x.
```

And the composition of functions f and g is given as

```
f . g = \x -> f (g x).
```

Now, we can speak of some category theoretic concepts such as functors and monads in Haskell. Indeed, both functors and monads are implemented as typeclasses as given in the following:

```
class Functor F where
      fmap :: (a -> b) -> F a -> F b
```

The `fmap` method, for each `Functor` type `F`, applies a function of the arrow type `a -> b` to an instance of type `F a` so as to return an instance of type `F b`, for each types `a` and `b` in the category `Hask`.

```
class Monad m where
      join   :: m (m a) -> m a
      return :: a -> m a
```

Similarly, for each monadic type `m` and each type `a` in the category `Hask`:

(1) the method `join` takes an instance of type `m (m a)` and returns an instance of type `m a`, corresponding to the multiplication of a monad;

(2) the method `return` takes an instance of type `a` and returns an instance of type `m a`, corresponding to the unit of a monad.

Example 2.2.2. The `Maybe` type of Haskell can be used to represent computations that might fail. Let us illustrate it as functor and monad class instances. We start with its definition:

```
data Maybe a  = Just a | Nothing
```

Indeed, the type `Maybe a` has two constructors: `Just a` and `Nothing` for each type `a` in `Hask`.

```
instance Functor Maybe where
      fmap f Nothing        = Nothing
      fmap f (Just a)       = Just (f a)

instance Monad Maybe where
      return a              = Just a
      join (Just (Nothing)) = Nothing
      join (Just (Just a))  = Just a
```

The method `bind`, denoted `>>=`, can be defined through `fmap` and `join` as follows:

```
bind = join . fmap f (Maybe a)
```

for each function `f :: a -> Maybe a` and type `a` in `Hask`.

For any even integer n, a chain of computations to calculate the n^{th} integral root of a given integer, if it exists, can be implemented with the use of the `Maybe` monad and the `bind` method as follows:

```
sqroot   :: Integer -> Maybe Integer
sqroot x  = sqroot'(0,0) where
      sqroot'(s,r)
            | s > x     = Nothing
            | s == x    = Just r
            | otherwise = sqroot' x (s+2*r+1,r+1)
```

The above function[1] calculates the positive square root of a given integer, if it exists. Now, calculating the positive 4^{th} root is just binding the handled square root value to the same function.

```
4throot   :: Integer -> Maybe Integer
4throot x  = sqroot x >>= sqroot
```

Obviously, to calculate the positive 8^{th} root, a further binding is necessary thus for the positive n^{th} root, one needs $n/2$ bindings. If the computation fails, then it returns `Nothing`.

[1] the source has been taken from the post "Understanding Haskell Monads" by Mr. Ertuğrul Söylemez.

```
8throot    :: Integer -> Maybe Integer
8throot x  = sqroot x >>= sqroot >>= sqroot
```

Therefore, error management is purely handled via the `Maybe` monad.

2.2.2 Eff

Eff, developed by Bauer and Pretnar [BP15, BP14, Pre14], is a programming language implementing the approach of effects as Lawvere theories with handlers. Below, we summarize the constructions specific to `Eff` detailing neither the syntax (expressions and computations), nor the issues related to type checking and denotational semantics.

Instances and operations. We prefer to skip the technicalities of `Eff` types, expressions and computations. However, it is crucial to note that `Eff` has effect types `E`, describing several related *effectful operations*, and handler types `A` $\Rightarrow$ `B` indicating that an instance of such handler acts on computations of standard type `A` and returns computations of standard type `B`. For instance, given an effect instance `e` of type `E` and an operation symbol `op`: `A` $\to$ `B` $\in$ `E` (contained in `E`), there is an operation `e#op`: `A` $\to$ `B` which is known as a generic effect and which is effect free as it is. However, it becomes an effectful operation when applied to an expression *exp*. This is supposed to be handled by a handler of type `A` $\Rightarrow$ `B`. Besides, there is a crucial computation `new` which generates an *effect instance* of effect type `E`.

Handlers. A handler

$$h \;=\; \texttt{val}\ x \mapsto c_v \;\mid\; \texttt{handler}(e_i\#\texttt{op}_i\ x\ k \mapsto c_i)_i \;\mid\; \texttt{finally} \mapsto c_f$$

can be applied to a computation c via the below handling construct:

$$\texttt{with}\ h\ \texttt{handle}\ c.$$

If the computation c evaluates into `val` v, then the handling construct binds v to x and evaluates into c_v. Else if c meets an *effectful* operation $e_i\#\texttt{op}_i\, exp$ during the evaluation, the handling construct binds *exp* to x and the provided continuation to k thus evaluates into a computation c_i which may still be handled by outer handling constructs, since continuation is delimited. The `finally` clause can be seen as an additional transformation which converts the handling construct into:

$$\texttt{let}\ x\ =\ (\texttt{with}\ h\ \texttt{handle}\ c)\ \texttt{in}\ c_f.$$

Obviously, if the c encounters a computation $e_i\#\texttt{op}_i\ e$ which is not considered by the handler h, then the effect gets propagated and might be handled by outer handling constructs.

Now, let us consider a simple example which shows the way to handle exceptions in `Eff`:

Example 2.2.3. An exception is an effect with a single operation named `raise` (no characterizing operation, see Example 2.1.5) which takes a parameter of type `'a` and returns an instance of `empty` type:

```
type 'a exception = effect
  operation raise: 'a -> empty
end
```

The input parameter `'a` can be used by the exception handler while the return instance of the `empty` type indicates that a raised exception does not give the control back to the continuation.

```
let optionalize e = handler
  | e#raise v _ -> print v
  | val x       -> print x ;;
```

The handler `optionalize e` either prints the non-exceptional value `x` or first handles an exception instance `e` with parameter `v` and then prints it out. Notice that in the above implementation, there is no provided continuation: this is ensured by the use of (_).

In order to make use of this handler, we first create an instance of the `exception` effect:

```
let e = new exception;;
```

Now, we handle the computation `raise e (3 * 100)` with `optionalize e` which prints the associated parameter to the screen:

```
with optionalize e handle
    raise e (3 * 100) (* Raise e with argument 300. *)
```

To provide a continuation, say by a user-defined function, one needs to replace the underscore with the function in question.

Lastly, `Eff` enables programmers to implement handlers of several other effects and it also supports effect combination. For detailed examples, see [BP15, §6].

2.3 Proof assistants

It is crucial to note that neither Haskell nor Eff (to our knowledge) include a verification process. Rather, within their formal context, any syntactically well-typed code is supposed to be correct (aka certified) provided that the underlying logic is. Conversely, platforms like Idris, Coq and Isabelle are designed either to be verification oriented or supported. Thus, in the following, we give some pointers about such tools.

2.3.1 Idris

Idris is a purely functional programming language using an eager evaluation strategy and dependent types. It involves a library to manage computational effects named `Effects`. In [Bra13], Edwin C. Brady describes how to use the `Effects` library: how to create new computational effects, how they are implemented as well as how to handle them via an approach based on *algebraic handlers* as in [BP15]. Idris also supports interactive theorem proving with tactics. For all further information, check out the below link:

`http://www.idris-lang.org/`

2.3.2 Coq

Coq is a proof assistant which implements a higher order mathematical language named Gallina. The underlying formal language of Gallina is the *Calculus of Constructions* (CoC) developed by Thierry Coquand and Gérard Huet [CH88] which extends the simply

typed λ-calculus with polymorphism, dependent types and type operators: when considered the of Barendregt's lambda cube [BDS13], it locates on the right-top. In time, CoC has been enriched with the use of inductive, coinductive types and hierarchical Universes so as to evolve in *Calculus of (co)Inductive Constructions* (CIC). For all further information, check out the below link:

`https://coq.inria.fr/`

In this thesis, on the one hand, we use Coq as a platform to formalize Duval's decorated logic for the treatments of computational effects: the state and the exceptions. To do so, we mainly exploit inductive and dependent types. On the other hand, we use Coq as a proof development system by benefiting its interactive proof methods and the tactic language when certifying properties of programs formalized with a decorated logic.

2.3.3 Isabelle

Isabelle [NPW02] is an interactive prover which embeds a formal mathematical language named ISAR. It has mainly been developed at University of Cambridge and Technische Universität München. It involves tools for proving mathematical formulae in a logical calculus. Nowadays Isabelle/HOL is the mostly used and spread instance: apart from proving theorems based on a higher-order logic, it also enables the use of structures such as (co)datatypes, (co)inductive definitions and recursive functions with pattern matching. For all further information, check out the below link:

`https://isabelle.in.tum.de/`

2.4 Concluding remarks: where is this thesis located?

Provided the aforementioned state-of-the-art, in this section, we clarify the point where this thesis is located.

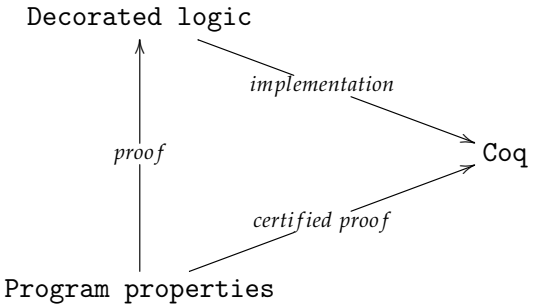


Figure 2.3: Thesis approach

We chose the `decorated logic` to formalize computational effects of a program and prove its properties. As depicted in Figure 2.3, certifying property proofs of programs with effects in a decorated setting is about using `Coq` as a proof development system after implementing the related `decorated logic` in `Coq`.

On the choice of decorated logic. In this thesis, we choose the decorated logic paradigm to formalize computational effects, mainly due to following arguments:

(1) since effects of terms are hidden by the decorations, it is possible to preserve the syntax of term signatures. Thereafter, the provided equational reasoning would be valid for different algebraic models of the same effect.

(2) the equational theory is based on *decorated equivalence relations* proposing different reasoning capabilities: one on effects and returned results and the other one only on returned results.

On the choice of Coq. To our knowledge, apart from `Coq`, `Isabelle` and `Idris`, other mentioned tools, `Haskell` and `Eff`, do not embody platforms which could be used both to program formal logic and develop certified proofs.

We could have used `Idris` as a platform both to implement the decorated logic and to interactively prove theorems. We do not have any apparent argument to prefer `Coq` against `Idris` apart from their focuses: `Idris` supports an interactive theorem proving based on *general-purpose-programming* while `Coq` originally motivates *theorem proving*.

The choice between `Isabelle` and `Coq` does not strongly stand by neither side for our implementation except for the following argument: using inductive predicates of `Coq` might be comparatively harder when implementing but easier when using induction as a reasoning strategy.

Using separate platforms for programming formal logic and developing certified proofs is an option where a verified translator (from the platform to program formal logic to the platform to develop certified proofs) would be necessary. However, using Coq for both issues seems to be more homogenous and trustworthy. Notice also that we will not formalize in Coq the categorical interpretation of the logics we propose. For a Coq formalization of category theoretic structures, see [Ahr15].

3

Categorical background

In this chapter, first we separately study the Kleisli and coKleisli adjunctions associated to a monad and a comonad in Section 3.1. Then, Section 3.2 starts with the proof of the comparison theorem for the coKleisli construction [ML71, Ch. VI, §5, dual of Theorem 2] and continues with the composition of Kleisli and coKleisli adjunctions: such a two-level structure is named the *coKleisli-on-Kleisli construction associated to a monad* and studied in detail with an application to the exceptions monad. In Chapter 6 we will make use of this construction to interpret the *decorated logic for the exception effect*. Finally, Section 3.3 dualizes the construction introduced in Section 3.2. There, we first give the comparison theorem for the Kleisli construction [ML71, Ch. VI, §5, Theorem 2] and proceed with the composition of coKleisli and Kleisli adjunctions, yielding the *Kleisli-on-coKleisli construction associated to a comonad*. We apply this composition to the states comonad, so as to interpret the *decorated logic for the state effect* in Chapter 5.

The main result of this chapter is that the coKleisli-on-Kleisli category of a monad and the Kleisli-on-coKleisli category of a comonad are proven to be respectively the full image category of the related monad and comonad endofunctors (Theorems 3.2.5 and 3.3.4).

3.1 Adjunctions, monads and comonads

This section aims to study the Kleisli adjunction associated to a monad and its dual. We start with some preliminary notions that might be helpful: all related details can be found in [ML71, Ch. IV, Ch. VI].

3.1.1 Preliminaries

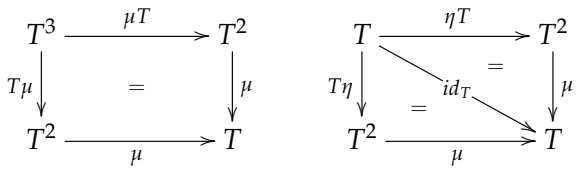
Definition 3.1.1. Let $\mathscr{C}$ and $\mathscr{D}$ be two categories. An *adjunction* $F \dashv G \colon \mathscr{D} \to \mathscr{C}$ is a triple $\langle F, G, \varphi \rangle$ such that $F \colon \mathscr{C} \to \mathscr{D}$, $G \colon \mathscr{D} \to \mathscr{C}$ are functors and $\varphi = (\varphi_{X,A})_{X,A}$ is a family of bijections, natural in X and A, where X is an object of $\mathscr{C}$ and A is an object of $\mathscr{D}$:

$$\varphi_{X,A} \colon Hom_{\mathscr{D}}(FX, A) \xrightarrow{\cong} Hom_{\mathscr{C}}(X, GA) \tag{3.1}$$

Definition 3.1.2. A *monad* $T = (T, \eta, \mu)$ in a category $\mathscr{C}$ consists of an endofunctor $T \colon \mathscr{C} \to \mathscr{C}$ with two natural transformations

$$\eta \colon Id_{\mathscr{C}} \Rightarrow T \qquad \mu \colon T^2 \Rightarrow T \tag{3.2}$$

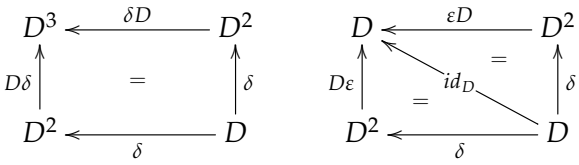
such that the following diagrams commute:



Definition 3.1.3. A *comonad* $D = (T, \varepsilon, \delta)$ in a category $\mathscr{C}$ consists of an endofunctor $D\colon \mathscr{C} \to \mathscr{C}$ with two natural transformations

$$\varepsilon\colon D \Rightarrow Id_{\mathscr{C}} \qquad \delta\colon D \Rightarrow D^2 \tag{3.3}$$

such that the following diagrams commute:



Let us consider an adjunction $F \dashv G\colon \mathscr{D} \to \mathscr{C}$. We set $A = FX$ in (3.1) and get $\eta_X\colon X \to GFX$ in $\mathscr{C}$ which is the image of id_{FX} by $\varphi_{X,FX}$. Symmetrically, by setting $X = GA$, we obtain $\varepsilon_A\colon FGA \to A$ in $\mathscr{D}$ which is the image of id_{GA} by $\varphi^{-1}_{GA,A}$. As shown in [ML71, Ch. IV, §1], $\eta\colon Id_{\mathscr{C}} \Rightarrow GF$ and $\varepsilon\colon FG \Rightarrow Id_{\mathscr{D}}$ are *natural transformations*. Thus, we get the following proposition by [ML71, Ch. VI, §1] and [ML71, Ch. IV, §1, Theorem 1].

Proposition 3.1.4. *An adjunction $F \dashv G\colon \mathscr{D} \to \mathscr{C}$, with associated family of bijections φ as in* Definition 3.1.1*, determines a monad on $\mathscr{C}$ and a comonad on $\mathscr{D}$ as follows:*

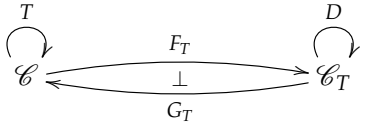
- *The monad (T, η, μ) on $\mathscr{C}$ has endofunctor $T = GF\colon \mathscr{C} \to \mathscr{C}$, unit $\eta\colon Id_{\mathscr{C}} \Rightarrow T$ where $\eta_X = \varphi_{X,FX}(id_{FX})$ and multiplication $\mu\colon T^2 \Rightarrow T$ such that $\mu_X = G(\varepsilon_{FX})$.*
- *The comonad (D, ε, δ) on $\mathscr{D}$ has endofunctor $D = FG\colon \mathscr{D} \to \mathscr{D}$, counit $\varepsilon\colon D \Rightarrow Id_{\mathscr{D}}$ where $\varepsilon_A = \varphi^{-1}_{GA,A}(id_{GA})$ and comultiplication $\delta\colon D \Rightarrow D^2$ such that $\delta_A = F(\eta_{GA})$.*

In addition, we have:

$$\varphi_{X,A} f = Gf \circ \eta_X\colon X \to GA \text{ for each } f\colon FX \to A \tag{3.4}$$
$$\varphi^{-1}_{X,A} g = \varepsilon_A \circ Fg\colon FX \to A \text{ for each } g\colon X \to GA. \tag{3.5}$$

3.1.2 The Kleisli adjunction associated to a monad

Each monad (T, η, μ) on a category $\mathscr{C}$ determines a *Kleisli category* $\mathscr{C}_T$ and an associated adjunction $F_T \dashv G_T\colon \mathscr{C}_T \to \mathscr{C}$ as follows:



$$\eta : Id \Rightarrow T \qquad F_T \dashv G_T \qquad \varepsilon : D \Rightarrow Id$$

Note that all related details can be found in [ML71, Ch. VI, §5].

- The categories $\mathscr{C}$ and $\mathscr{C}_T$ have the same objects and there is a morphism $f^\flat \colon X \to Y$ in $\mathscr{C}_T$ for each morphism $f \colon X \to TY$ in $\mathscr{C}$. So that there is a bijection defined as:

$$(\varphi_T)_{X,Y} \colon Hom_{\mathscr{C}_T}(X,Y) \xrightarrow{\cong} Hom_{\mathscr{C}}(X,TY) \tag{3.6}$$

$$f^\flat \leftarrowtail f \tag{3.7}$$

- For each object X in $\mathscr{C}_T$, the identity arrow is $id_X = h^\flat \colon X \to X$ in $\mathscr{C}_T$ where

$$h = \eta_X \colon X \to TX \text{ in } \mathscr{C}. \tag{3.8}$$

- The composition of a pair of morphisms $f^\flat \colon X \to Y$ and $g^\flat \colon Y \to Z$ in $\mathscr{C}_T$ is given by the Kleisli composition:

$$g^\flat \circ f^\flat = h^\flat \colon X \to Z \text{ where } h = \mu_Z \circ Tg \circ f \colon X \to TZ \text{ in } \mathscr{C}. \tag{3.9}$$

- The functor $F_T \colon \mathscr{C} \to \mathscr{C}_T$ is the identity on objects. On morphisms,

$$F_T f = (\eta_Y \circ f)^\flat, \text{ for each } f \colon X \to Y \text{ in } \mathscr{C}. \tag{3.10}$$

- The functor $G_T \colon \mathscr{C}_T \to \mathscr{C}$ maps each object X in $\mathscr{C}_T$ to TX in $\mathscr{C}$. On morphisms,

$$G_T(g^\flat) = \mu_Y \circ Tg, \text{ for each } g^\flat \colon X \to Y \text{ in } \mathscr{C}_T. \tag{3.11}$$

Then, the monad associated to the adjunction $F_T \dashv G_T$ is actually the monad (T, η, μ), so that $T = G_T F_T$.

Definition 3.1.5. The monad (T, η, μ) is said to satisfy the *mono requirement* if η_X is a monomorphism for each X [Mog89].

Theorem 3.1.6. *Let (T, η, μ) be a monad on a category $\mathscr{C}$. Let $\mathscr{C}_T$ be the Kleisli category and let $F_T \dashv G_T \colon \mathscr{C}_T \to \mathscr{C}$ be the Kleisli adjunction determined by (T, η, μ). Then, η_X is mono for each object X in $\mathscr{C}$ if and only if F_T is faithful.*

Proof. The proof is dual to the proof given in [ML71, Ch. IV, §3, Theorem 1]. □

In addition, the associated comonad (D, ε, δ) is defined by the application of Proposition 3.1.4 as follows:

- On objects, $DX = F_T G_T X = G_T X = TX$ in $\mathscr{C}_T$, for each X in $\mathscr{C}_T$.
- On morphisms,

$$D(f^\flat) = F_T G_T(f^\flat) = F_T(\mu_Y \circ Tf) = (\eta_{TY} \circ \mu_Y \circ Tf)^\flat = h^\flat \tag{3.12}$$

  for each $f^\flat \colon X \to Y$ and some $h^\flat$ in $\mathscr{C}_T$ such that $h = \eta_{TY} \circ \mu_Y \circ Tf \colon TX \to T^2Y$ in $\mathscr{C}$.

- the counit is given by

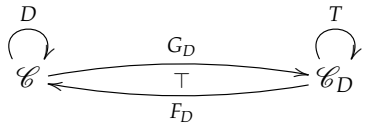
$$\varepsilon_Y = (id_{G_T Y})^\flat, \text{ for each } Y \text{ in } \mathscr{C}_T. \tag{3.13}$$

- the comultiplication is given by

$$\delta_Y = F_T(\eta_{G_T Y}) = (\eta_{G_T F_T G_T Y} \circ \eta_{G_T Y})^\flat \text{ in } \mathscr{C}_T \text{ where } \eta_{G_T F_T G_T Y} \circ \eta_{G_T Y} \colon TY \to T^3Y \text{ in } \mathscr{C}. \tag{3.14}$$

3.1.3 The coKleisli adjunction associated to a comonad

In this section, we dualize of the notions introduced in Section 3.1.2. Now, let (D, ε, δ) be a comonad on a category $\mathscr{C}$. Let $\mathscr{C}_D$ be the *coKleisli category* and let $F_D \dashv G_D \colon \mathscr{C} \to \mathscr{C}_D$ be the adjunction it determines with following settings:



$$\varepsilon : D \Rightarrow Id \qquad G_D \vdash F_D \qquad \eta : Id \Rightarrow T$$

- The categories $\mathscr{C}$ and $\mathscr{C}_D$ have the same objects and there is a morphism $f^\sharp \colon X \to Y$ in $\mathscr{C}_D$ for each morphism $f \colon DX \to Y$ in $\mathscr{C}$. So that there is a bijection defined as:

$$(\psi_D)_{X,Y} \colon Hom_{\mathscr{C}_D}(X, Y) \xrightarrow{\cong} Hom_{\mathscr{C}}(DX, Y) \tag{3.15}$$

$$f^\sharp \leftarrow\!\shortmid f \tag{3.16}$$

- For each object X in $\mathscr{C}_D$, the identity arrow is $id_X = h^\sharp \colon X \to X$ in $\mathscr{C}_D$ where

$$h = \varepsilon_X \colon DX \to X \text{ in } \mathscr{C}. \tag{3.17}$$

- The composition of a pair of morphisms $f^\sharp \colon X \to Y$ and $g^\sharp \colon Y \to Z$ in $\mathscr{C}_D$ is given by the coKleisli composition:

$$g^\sharp \circ f^\sharp = h^\sharp \text{ where } h = g \circ Df \circ \delta_X \colon DX \to Z \text{ in } \mathscr{C}. \tag{3.18}$$

- The functor $G_D \colon \mathscr{C} \to \mathscr{C}_D$ is the identity on objects. On morphisms,

$$G_D f = (f \circ \varepsilon_X)^\sharp, \text{ for each } f \colon X \to Y \text{ in } \mathscr{C}. \tag{3.19}$$

- The functor $F_D \colon \mathscr{C}_D \to \mathscr{C}$ maps each object X in $\mathscr{C}_D$ to DX in $\mathscr{C}$. On morphisms,

$$F_D(g^\sharp) = Dg \circ \delta_X, \text{ for each } g^\sharp \colon X \to Y \text{ in } \mathscr{C}_D. \tag{3.20}$$

Then, the comonad associated to the adjunction $F_D \dashv G_D$ is actually the comonad (D, ε, δ), so that $D = F_D G_D$.

Definition 3.1.7. The comonad (D, ε, δ) is said to satisfy the *epi requirement* if ε_X is epi for each X.

Theorem 3.1.8. *Let (D, ε, δ) be a comonad on a category $\mathscr{C}$. Let $\mathscr{C}_D$ be the coKleisli category and $F_D \dashv G_D \colon \mathscr{C} \to \mathscr{C}_D$ be coKleisli adjunction determined by (D, ε, δ). Then, ε_X is epi for each object X in $\mathscr{C}$, if and only if G_D is faithful.*

Proof. The proof is given in [ML71, Ch. IV, §3, Theorem 1]. □

In addition, the associated monad (T, η, μ) is defined by the application of Proposition 3.1.4 as follows:

- On objects, $TX = G_D F_D X = F_D X = DX$, for each X in $\mathscr{C}_D$.

- On morphisms,

$$Tf^\sharp = G_D F_D(f^\sharp) = G_D(Df \circ \delta_X) = (Df \circ \delta_X \circ \varepsilon_{DX})^\sharp = h^\sharp \tag{3.21}$$

  for each $f^\sharp \colon X \to Y$ and some $h^\sharp$ in $\mathscr{C}_D$ such that $h = Df \circ \delta_X \circ \varepsilon_{DX} \colon D^2X \to DY$ in $\mathscr{C}$.

- the unit is given by

$$\eta_X = (id_{F_D X})^\sharp, \text{ for each } X \text{ in } \mathscr{C}_D. \tag{3.22}$$

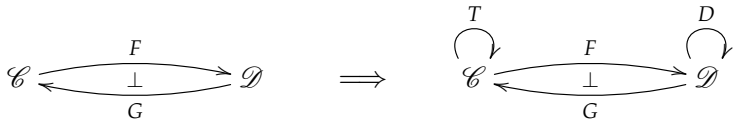
- the multiplication is given by

$$\mu_X = G_D(\varepsilon_{F_D X}) = (\varepsilon_{F_D X} \circ \varepsilon_{F_D G_D F_D X})^\sharp \text{ in } \mathscr{C}_D \tag{3.23}$$

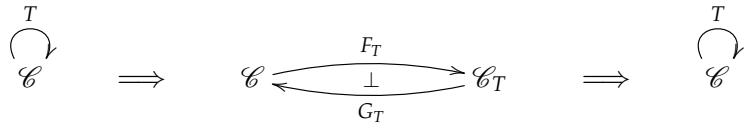
$$\text{where } \varepsilon_{F_D X} \circ \varepsilon_{F_D G_D F_D X} \colon D^3X \to DX \text{ in } \mathscr{C}. \tag{3.24}$$

3.1.4 Summary

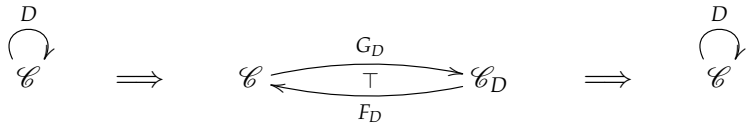
In summary, an adjunction $F \dashv G \colon \mathscr{D} \to \mathscr{C}$ [Kan58] determines a monad (T, η, μ) on the category $\mathscr{C}$ and dually a comonad (D, ε, δ) on the category $\mathscr{D}$.



Conversely, a monad (T, η, μ) on a category $\mathscr{C}$ may determine several adjunctions, including: $F_T \dashv G_T \colon \mathscr{C}_T \to \mathscr{C}$ where $\mathscr{C}_T$ is the *Kleisli category* of (T, η, μ). The associated adjunction $F_T \dashv G_T$ determines back the monad (T, μ, η).

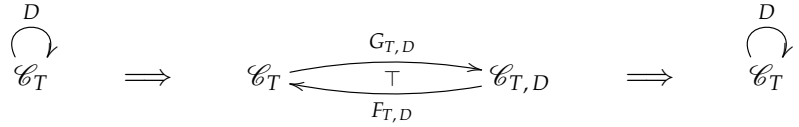


Dually, a comonad (D, ε, δ) on a category $\mathscr{C}$ may determine several adjunctions, including: $F_D \dashv G_D \colon \mathscr{C} \to \mathscr{C}_D$ where $\mathscr{C}_D$ is the *coKleisli category* of (D, ε, δ). The associated adjunction $F_D \dashv G_D$ determines back the comonad (D, ε, δ).



3.2 The coKleisli-on-Kleisli construction associated to a monad

The adjunction, given in Sections 3.1.2, $F_T \dashv G_T \colon \mathscr{C}_T \to \mathscr{C}$ determines a comonad (D, ε, δ) on $\mathscr{C}_T$. This comonad further determines several adjunctions, including: $F_{T,D} \dashv G_{T,D} \colon \mathscr{C}_T \to \mathscr{C}_{T,D}$ where $\mathscr{C}_{T,D}$ is the *coKleisli category* of (D, ε, δ). The associated adjunction $F_{T,D} \dashv G_{T,D}$ determines back the comonad (D, ε, δ).



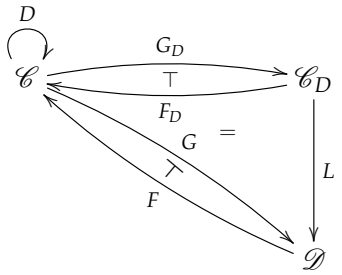
Besides, we show in Theorem 3.2.5 that the category $\mathscr{C}_{T,D}$ is the *full image category* of the endofunctor T.

Therefore, in this section, we study the composition of the Kleisli construction $F_T \dashv G_T \colon \mathscr{C}_T \to \mathscr{C}$ associated to a monad T, as detailed in Section 3.1.2, with the coKleisli adjunction $F_{T,D} \dashv G_{T,D} \colon \mathscr{C}_T \to \mathscr{C}_{T,D}$ associated to a comonad D as given in Section 3.1.3, when the comonad D is determined by $F_T \dashv G_T$. As a result of this composition, we obtain the coKleisli-on-Kleisli construction associated to a monad T. The generic settings provided by such an approach are applied to the *exceptions monad* in Section 3.2.3. The main aim of which will become explicit in Section 6.1 where we interpret the *decorated logic for the exception effect*. This logic proposes a formalism to prove properties of programs with exceptional features.

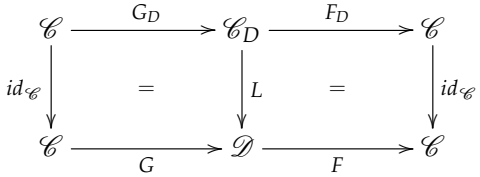
3.2.1 The comparison theorem for the coKleisli construction

Let us start with the proof of the comparison theorem for the coKleisli construction. A specialization of this result will be used in Proposition 3.2.5, which is important for highlighting the relation among the categories defined in Section 3.2.2.

Theorem 3.2.1. (The comparison theorem for the coKleisli construction) *Let $F \dashv G \colon \mathscr{C} \to \mathscr{D}$ be an adjunction and let (D, ε, δ) be the associated comonad on $\mathscr{C}$. Then, there is a unique functor $L \colon \mathscr{C}_D \to \mathscr{D}$ such that $LG_D = G$ and $FL = F_D$, where $\mathscr{C}_D$ is the coKleisli category of (D, ε, δ), with the associated adjunction $F_D \dashv G_D \colon \mathscr{C} \to \mathscr{C}_D$.*



Proof. We give the proof since it is left as an exercice in [ML71, Ch. VI, §5, Theorem 2]. Let us first assume that $L \colon \mathscr{C}_D \to \mathscr{D}$ is a functor satisfying $LG_D = G$ and $FL = F_D$.



Let $\theta_{X,Y} \colon Hom_{\mathscr{C}_D}(X, G_D Y) \xrightarrow{\cong} Hom_{\mathscr{C}}(F_D X, Y)$ be the bijection associated to the adjunction $F_D \dashv G_D$. Similarly, let $\psi_{X,Y} \colon Hom_{\mathscr{D}}(X, GY) \xrightarrow{\cong} Hom_{\mathscr{C}}(FX, Y)$ be the bijection associated to the adjunction $F \dashv G$. Since both the counit of the adjunction $F_D \dashv G_D$ and the counit of the adjunction $F \dashv G$ are the counit ε of the comonad (D, ε, δ), by [ML71,

Ch. IV, §7, Proposition 1], we obtain the commutative diagram below:

$$\begin{array}{ccc} Hom_{\mathscr{C}_D}(X, G_D Y) & \xrightarrow{\theta_{X,Y}} & Hom_{\mathscr{C}}(F_D X, Y) \\ \Big\downarrow{\scriptstyle L_{X,G_D Y}} & = & \Big\downarrow{\scriptstyle id_{F_D X,Y}} \\ Hom_{\mathscr{D}}(LX, LG_D Y) & & Hom_{\mathscr{C}}(F_D X, Y) \\ \| & & \| \\ Hom_{\mathscr{D}}(LX, GY) & \xrightarrow{\psi_{LX,Y}} & Hom_{\mathscr{C}}(FLX, Y) \end{array}$$

Therefore, $L_{X,G_D Y} = \psi^{-1}_{LX,Y} \circ \theta_{X,Y}$. This formula ensures that the functor is L is *unique*. Let us simplify it: by Equation (3.5) in Proposition 3.1.4, we have:

$$\theta_{X,Y} f^\sharp = \varepsilon_Y \circ F_D f^\sharp \colon F_D X \to Y, \text{ for each } f^\sharp \colon X \to G_D Y = Y \text{ in } \mathscr{C}_D.$$

Since, $F_D f^\sharp = Df \circ \delta_X$ in $\mathscr{C}$, for each $f^\sharp \colon X \to Y$ by Equation (3.20), we get

$$\theta_{X,Y} f^\sharp = \varepsilon_Y \circ Df \circ \delta_X \colon F_D G_D X = F_D X \to Y.$$

Thanks to the naturality of ε, we get $\theta_{X,Y} f^\sharp = f \circ \varepsilon_{DX} \circ \delta_X$. The comonadic axiom ensuring $\varepsilon_{DX} \circ \delta_X = id_{DX}$ yields $\theta_{X,Y} f^\sharp = f \colon F_D X \to Y$. Presumed that $F_D = FL$ and since G_D is the identity on objects, we have

$$\theta_{X,Y} f^\sharp = f \colon FLX \to Y \text{ and } LG_D X = LX = GX.$$

Now, by Equation (3.4) in Proposition 3.1.4, we obtain

$$\psi^{-1}_{LX,Y} f = Gf \circ \eta_{LX} = Gf \circ \eta_{GX} = \psi^{-1}_{GX,Y} f \text{ for each } f \colon FGX \to Y \text{ in } \mathscr{C}.$$

Hence,

$$\psi^{-1}_{LX,Y}(\theta_{X,Y} f^\sharp) = \psi^{-1}_{GX,Y} f = Gf \circ \eta_{GX}.$$

In other words: given a functor L satisfying $GL = G_T$ and $LF_T = F$, then it must be such that $LX = GX$ for each object X in $\mathscr{C}_D$ and

$$Lf^\sharp = Gf \circ \eta_{GX} \text{ in } \mathscr{D} \text{ for each } f^\sharp \colon X \to Y \text{ in } \mathscr{C}_D. \tag{3.25}$$

We additionally need to prove that the mapping $L \colon \mathscr{C}_D \to \mathscr{D}$, characterized by $LX = X$ and $Lf^\sharp = Gf \circ \eta_{GX}$, is a functor satisfying $LG_D = G$ and $FL = F_D$:

1. For each object X in $\mathscr{C}_D$, due to the fact that $id_X = (\varepsilon_X)^\sharp$ in $\mathscr{C}_D$, we have $L(id_X) = L((\varepsilon_X)^\sharp) = G\varepsilon_X \circ \eta_{GX}$. By [ML71, Ch. IV, §1, Theorem 1], we have

   $$G\varepsilon_X \circ \eta_{GX} = id_{GX} = id_{LX}.$$

   For each pair of morphisms $f^\sharp \colon X \to Y$ and $g^\sharp \colon Y \to Z$ in $\mathscr{C}_D$, by coKleisli composition, we get

   $$L(g^\sharp \circ f^\sharp) = Gg \circ GFGf \circ GF\eta_{GX} \circ \eta_{GX}.$$

   Since η is natural, we obtain $L(g^\sharp \circ f^\sharp) = Gg \circ \eta_{GY} \circ Gf \circ \eta_{GX}$ which is $L(g^\sharp) \circ L(f^\sharp)$ in $\mathscr{D}$. Hence, $L \colon \mathscr{C}_D \to \mathscr{D}$ is a functor.

2. For each object X in $\mathscr{C}$, $G_D X = X$ in $\mathscr{C}_D$ and $LG_D X = GX$ in $\mathscr{D}$. For each morphism $f\colon X \to Y$ in $\mathscr{C}$, $G_D f = (f \circ \varepsilon_X)^\sharp$ by Equation (3.19). Hence,

$$LG_D f = L((f \circ \varepsilon_X)^\sharp) = Gf \circ G\varepsilon_X \circ \eta_{GX}.$$

Due to ε and η are natural, we have $G\varepsilon_X \circ \eta_{GX} = id_{GX}$ yielding $LG_D f = Gf$. Thus,

$$LG_D = G.$$

3. Now, for each object X in $\mathscr{C}_D$, $LX = GX$ in $\mathscr{D}$. So that $FLX = FGX$ in $\mathscr{C}$. Similarly, $F_D X = FGX$ in $\mathscr{C}$ by definition. Hence, $FLX = F_D X$ on objects. For each morphism $f^\sharp\colon X \to Y$ in $\mathscr{C}_D$, $Lf^\sharp = Gf \circ \eta_{GX}$, by definition. Hence,

$$FLf^\sharp = FGf \circ F\eta_{GX}.$$

Similarly, Equation (3.20) gives:

$$F_D f^\sharp = FGf \circ F\eta_{GX}.$$

We get $FLf = F_D f$ for each mapping f, therefore, $FL = F_D$.

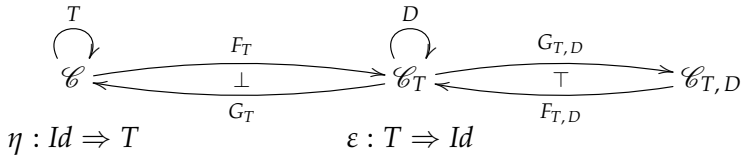
□

3.2.2 The coKleisli-on-Kleisli construction

Let (T, η, μ) be a monad defined on a category $\mathscr{C}$. It determines a Kleisli category $\mathscr{C}_T$ along with the associated adjunction $F_T \dashv G_T : \mathscr{C}_T \to \mathscr{C}$. Let $(D = F_T G_T, \varepsilon, \delta)$ be the comonad that the adjunction $F_T \dashv G_T : \mathscr{C}_T \to \mathscr{C}$ determines on $\mathscr{C}_T$. Refer back to Section 3.1.2, for the related details.

Remark 3.2.2. Note that the details of below items, from 1 to 4, are depicted in Figure 3.1.

Now, let $\mathscr{C}_{T,D}$ be the coKleisli category determined by (D, ε, δ) and let $F_{T,D} \dashv G_{T,D}\colon \mathscr{C}_T \to \mathscr{C}_{T,D}$ be the associated adjunction with the following settings:



1. The categories $\mathscr{C}$ and $\mathscr{C}_T$ have the same objects and there is a morphism $f^\flat\colon X \to Y$ in $\mathscr{C}_T$ for each morphism $f\colon X \to TY$ in $\mathscr{C}$.

2. The categories $\mathscr{C}_T$ and $\mathscr{C}_{T,D}$ have the same objects and there is a morphism $h^\sharp\colon X \to Y$ in $\mathscr{C}_{T,D}$ for each $h\colon TX \to Y$ in $\mathscr{C}_T$.

3. The functor $G_{T,D}\colon \mathscr{C}_T \to \mathscr{C}_{T,D}$ is the identity on objects. On morphisms,

$$G_{T,D}(g^\flat) = (g^\flat \circ \varepsilon_X)^\sharp = h^\sharp \text{ for each } g^\flat\colon X \to Y \text{ in } \mathscr{C}_T. \tag{3.26}$$

Let $h = g^\flat \circ \varepsilon_X\colon TX \to Y$ in $\mathscr{C}_T$. Since $\varepsilon_X = (id_{TX})^\flat$ in $\mathscr{C}_T$, we get $h = g^\flat \circ (id_{TX})^\flat = k^\flat$ for some $k^\flat$ in $\mathscr{C}_T$. By Kleisli composition, we end up with

$$G_{T,D}(g^\flat) = h^\sharp \text{ such that } h = k^\flat \text{ and } k = \mu_Y \circ Tg\colon TX \to TY \text{ in } \mathscr{C}. \tag{3.27}$$

4. The functor $F_{T,D}\colon \mathscr{C}_{T,D} \to \mathscr{C}_T$ maps each X in $\mathscr{C}_{T,D}$ to TX in $\mathscr{C}_T$. On morphisms, $F_{T,D}(h^{\flat\sharp}) = D(h^{\flat}) \circ \delta_X$ for each $h^{\flat\sharp}\colon X \to Y$ in $\mathscr{C}_{T,D}$. Let us introduce mappings $g^{\flat}, a^{\flat}$ and $b^{\flat}$ in $\mathscr{C}_T$ and set

$$g^{\flat} = F_{T,D}(h^{\flat\sharp}), a^{\flat} = \delta_X \text{ and } b^{\flat} = D(h^{\flat})$$

Thus, we have $g^{\flat} = b^{\flat} \circ a^{\flat}$ and by Kleisli composition, we get:

$$g = \mu_{TY} \circ Tb \circ a\colon TX \to T^2Y \text{ in } \mathscr{C}.$$

(a) We have $\delta_X = F_T(\eta_{G_TX})$ by definition. We also have $G_TX = G_TF_TX = TX$ due to F_T being identity on objects. So that $\delta_X = F_T(\eta_{TX})$. Since $a^{\flat} = \delta_X$, we have $a^{\flat} = F_T(\eta_{TX})$. Using the fact that $F_Tf = (\eta_Y \circ f)^{\flat}$ for each $f\colon X \to Y$ in $\mathscr{C}$ (see Equation (3.10)), we derive:

$$a^{\flat} = (\eta_{T^2X} \circ \eta_{TX})^{\flat} \text{ hence } a = (\eta_{T^2X} \circ \eta_{TX}) \text{ in } \mathscr{C}.$$

(b) We also have $b^{\flat} = Dh^{\flat} = F_TG_Th^{\flat}$ such that $h\colon TX \to TY$ in $\mathscr{C}$. Provided by Equation (3.11) that $G_T(h^{\flat}) = \mu_Y \circ Th$. Therefore, we have $F_T(\mu_Y \circ Th) = (\eta_{TY} \circ \mu_Y \circ Th)^{\flat} = b^{\flat}$. So that $b = (\eta_{TY} \circ \mu_Y \circ Th)$ in $\mathscr{C}$. By rewriting a and b in g, we obtain:

$$g = \mu_{TY} \circ T\eta_{TY} \circ T\mu_Y \circ T^2h \circ \eta_{T^2X} \circ \eta_{TX}.$$

Thanks to the monadic axiom stating $\mu_{TY} \circ T\eta_{TY} = id_{T^2Y}$ we have $g = T\mu_Y \circ T^2h \circ \eta_{T^2X} \circ \eta_{TX}$. Since η is natural, used three times, we get $g = T\mu_Y \circ \eta_{T^2Y} \circ Th \circ \eta_{TX} = T\mu_Y \circ \eta_{T^2Y} \circ \eta_{TY} \circ h = \eta_{TY} \circ \mu_Y \circ \eta_{TY} \circ h$. Due to the monadic axiom ensuring $\mu_Y \circ \eta_{TY} = id_{TY}$, we write

$$F_{T,D}(h^{\flat\sharp}) = g^{\flat} \text{ where } g = \eta_{TY} \circ h. \tag{3.28}$$

Then, the associated comonad to the adjunction $F_{T,D} \dashv G_{T,D}$ is actually the comonad (D, ε, δ) where $D = F_{T,D}G_{T,D}$.

5. The composition $G_{T,D} \circ F_T\colon \mathscr{C} \to \mathscr{C}_{T,D}$ is the identity on objects. On morphisms, due to Equation (3.10), we have:

$$G_{T,D}F_T(f) = G_{T,D}((\eta_Y \circ f)^{\flat}) \text{ for each } f\colon X \to Y \text{ in } \mathscr{C}.$$

We further have:

$$G_{T,D}((\eta_Y \circ f)^{\flat}) = ((\eta_Y \circ f)^{\flat} \circ \varepsilon_X)^{\sharp} \text{ in } \mathscr{C}_{T,D} \text{ by Equation (3.26).}$$

Provided that $\varepsilon_X = (id_{TX})^{\flat}$ in $\mathscr{C}_T$, we get $G_{T,D}F_T(f) = ((\eta_Y \circ f)^{\flat} \circ (id_{TX})^{\flat})^{\sharp}$ and $G_{T,D}F_T(f) = (\mu_Y \circ T\eta_Y \circ Tf)^{\flat\sharp}$ by Kleisli composition. Due to the the monadic axiom stating $\mu_Y \circ T\eta_Y = id_{TY}$, it simplifies into

$$G_{T,D}F_T(f) = (Tf)^{\flat\sharp} = h^{\flat\sharp} \text{ such that } h = Tf\colon TX \to TY \text{ in } \mathscr{C}. \tag{3.29}$$

Proposition 3.2.3. *1. The categories $\mathscr{C}$ and $\mathscr{C}_{T,D}$ have the same objects and there is a morphism $k^{\flat\sharp}\colon X \to Y$ in $\mathscr{C}_{T,D}$ for each $k\colon TX \to TY$ in $\mathscr{C}$.*

2. For each object X in $\mathscr{C}_{T,D}$, the identity arrow is $id_X = k^{\flat\sharp}\colon X \to X$ in $\mathscr{C}_{T,D}$ where $k = id_{TX}\colon TX \to TX$ in $\mathscr{C}$.

3. *The composition of a pair of morphisms* $f^{\flat\sharp}\colon X \to Y$ *and* $g^{\flat\sharp}\colon Y \to Z$ *in* $\mathscr{C}_{T,D}$ *is given by* $g^{\flat\sharp} \circ f^{\flat\sharp} = k^{\flat\sharp}$ *where* $k = g \circ f\colon TX \to TZ$ *in* $\mathscr{C}$.

Proof. 1. It is the consequence of items 1 and 2 of Section 3.2.2.

2. For each object X, we have $k^{\flat\sharp} = id_X\colon X \to X$ in $\mathscr{C}_{T,D}$, where $k^{\flat} = \varepsilon_X\colon DX \to X$ in $\mathscr{C}_T$, due to Equation (3.17). Since $\varepsilon_X = (id_{G_TX})^{\flat}$ in $\mathscr{C}_T$, by Equation (3.13), we obtain $k = id_{G_TX}$ in $\mathscr{C}$. Now, $G_TX = TX$ yields $k = id_{TX}\colon TX \to TX$ in $\mathscr{C}$.

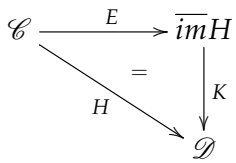
3. For each $g^{\flat\sharp}\colon Y \to Z$ and $f^{\flat\sharp}\colon X \to Y$ in $\mathscr{C}_{T,D}$, due to coKleisli composition, we get $g^{\flat} \circ f^{\flat} = g^{\flat} \circ D(f^{\flat}) \circ \delta_X = k^{\flat}$ in $\mathscr{C}_T$. Since $D(f^{\flat}) = (\eta_{TY} \circ \mu_Y \circ Tf)^{\flat}$ and $\delta_X = (\eta_{T^2X} \circ \eta_{TX})^{\flat}$ respectively given by Equations (3.12) and (3.14), we have $k^{\flat} = g^{\flat} \circ (\eta_{TY} \circ \mu_Y \circ Tf)^{\flat} \circ (\eta_{T^2X} \circ \eta_{TX})^{\flat}$. Thanks to associativity of composition and Kleisli composition we obtain $k^{\flat} = (\mu_Z \circ Tg \circ \eta_{TY} \circ \mu_Y \circ Tf)^{\flat} \circ (\eta_{T^2X} \circ \eta_{TX})^{\flat}$. Using Kleisli composition again, we get $k = \mu_Z \circ T\mu_Z \circ T^2g \circ T\eta_{TY} \circ T\mu_Y \circ T^2f \circ \eta_{T^2X} \circ \eta_{TX}$ in $\mathscr{C}$. Now, we use naturality of η twice and obtain $k = \mu_Z \circ T\mu_Z \circ T\eta_{TZ} \circ Tg \circ T\mu_Y \circ T\eta_{TY} \circ Tf \circ \eta_{TX}$. This simplifies into $k = \mu_Z \circ Tg \circ Tf \circ \eta_{TX}$, thanks to monadic axiom ensuring $\mu_X \circ \eta_{TX} = id_{TX}$ for each object X. Lastly, naturality of η gives $k = \mu_Z \circ \eta_{TZ} \circ g \circ f$, by the same monadic axiom, we end up with $k = g \circ f\colon TX \to TZ$ in $\mathscr{C}$.

□

Definition 3.2.4. Let $H\colon \mathscr{C} \to \mathscr{D}$ be a functor. Then, *the full image of* H is a category $\overline{im}H$ composed of objects X for each object X in $\mathscr{C}$ and morphisms $g^{\star}\colon X \to Y$ for each morphism $g : HX \to HY$ in $\mathscr{D}$. Let $E\colon \mathscr{C} \to \overline{im}H$ and $K\colon \overline{im}H \to \mathscr{D}$ be the functors defined as follows :

$$\begin{cases} E(X) = X \\ E(f) = (Hf)^{\star} \end{cases} \quad \text{and} \quad \begin{cases} K(X) = HX \\ K(g^{\star}) = g \end{cases}$$

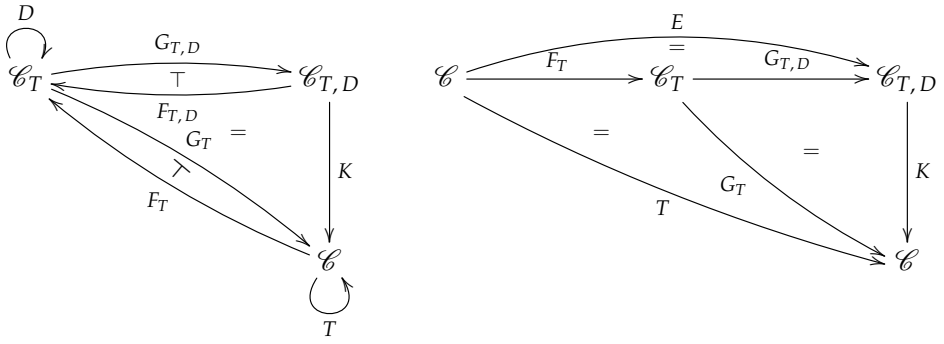
Then, *the full image factorization (or decomposition) of* H is the pair (E, K). Note that $H = K \circ E$.



Theorem 3.2.5. *Let* (T, η, μ) *be a monad on a category* $\mathscr{C}$ *and let* $\mathscr{C}_T$ *be the Kleisli category of* (T, η, μ) *with the associated the adjunction* $F_T \dashv G_T\colon \mathscr{C}_T \to \mathscr{C}$. *Let* (D, ε, δ) *be the comonad on* $\mathscr{C}_T$ *determined by the adjunction* $F_T \dashv G_T$. *And let* $\mathscr{C}_{T,D}$ *be the coKleisli category of* (D, ε, δ) *with the associated adjunction* $F_{T,D} \dashv G_{T,D}\colon \mathscr{C}_T \to \mathscr{C}_{T,D}$. *Then,*

1. *there is a unique functor* $K\colon \mathscr{C}_{T,D} \to \mathscr{C}$ *such that* $F_TK = F_{T,D}$ *and* $KG_{T,D} = G_T$.

2. *the full image factorization of T is given by $T = KE$ where $E = G_{T,D}F_T$.*



3. *for each X in $\mathscr{C}_T$, ε_X is split-epi thus $G_{T,D}$ is faithful.*

Proof. 1. We specialize Theorem 3.2.1 by instantiating $F_D \dashv G_D$ with $F_{T,D} \dashv G_{T,D}$ and $F \dashv G$ with $F_T \dashv G_T$. Thus, we obtain the unique functor $K\colon \mathscr{C}_{T,D} \to \mathscr{C}_T$ such that $KG_{T,D} = G_T$ and $F_TK = F_{T,D}$.

2. The category $\mathscr{C}_{T,D}$ is the full image category of T, since it is made of objects X for each X in $\mathscr{C}$ and arrows $f^{\flat\sharp}\colon X \to Y$ for each $f\colon TX \to TY$ in $\mathscr{C}$. Recall that $T = G_TF_T$ by definition and $G_T = KG_{T,D}$ by point 1, $T = KG_{T,D}F_T = KE$. On the one hand, $E(X) = G_{T,D}F_T(X) = X$ and $E(f) = G_{T,D}F_T(f) = (Tf)^{\flat\sharp}\colon TX \to TY$ thanks to Equation (3.29). On the other hand, $K(X) = TX$ for each object X and $K(g^{\flat\sharp}) = G_T(g^{\flat}) \circ \eta_{TX}$ for each $g^{\flat\sharp}\colon X \to Y$ in $\mathscr{C}_{T,D}$ by Equation (3.25). Thanks to Equation (3.11), we obtain $K(g^{\flat\sharp}) = \mu_Y \circ Tg \circ \eta_{TX}$. Since η is natural, we have $K(g^{\flat\sharp}) = \mu_Y \circ \eta_{TY} \circ g$. The monadic axiom ensuring $\mu_Y \circ \eta_{TY} = id_{TY}$ gives $K(g^{\flat\sharp}) = g\colon TX \to TY$. Obviously, $KE(f) = K(Tf^{\flat\sharp}) = Tf\colon TX \to TY$ for each morphism $f\colon X \to Y$ and $KE(X) = K(X) = TX$ for each object X. Therefore, the full image factorization of T is given by the pair (K, E).

3. It is necessary to show the existence of a mapping $f^{\flat}$ in $\mathscr{C}_T$ such that $\varepsilon_X \circ f^{\flat} = id_X$. Since $\varepsilon_X = (id_{TX})^{\flat}$ and $id_X = (\eta_X)^{\flat}$ in $\mathscr{C}_T$, we get $(id_{TX})^{\flat} \circ f^{\flat} = (\eta_X)^{\flat}$ in $\mathscr{C}_T$, and equivalently, $\eta_X = \mu_X \circ T(id_{TX}) \circ f$ in $\mathscr{C}$ by Kleisli composition. It is trivial to show, by the monadic property $id_{TX} = \mu_X \circ \eta_{TX}$, that this equation is satisfied when f is chosen to be $\eta_{TX} \circ \eta_X$. So that ε_X is split-epi, for each X. Notice also that split-epi implies epi. Now, due to the point (i) of [ML71, Ch. IV, §3, Theorem 1], we conclude that $G_{T,D}$ is faithful.

□

We will use the *coKleisli-on-Kleisli construction* in Section 6.1 where we interpret the *decorated logic for the exception effect*. This logic proposes a formalism to prove properties of programs with features to handle exceptions.

3.2.3 Application to the exceptions monad on sets

In this section, we apply the *coKleisli-on-Kleisli construction associated to a monad* to the *exceptions monad*. This means that we start with the exception monad defined on the category of sets $\mathscr{C}$ and then construct its Kleisli category $\mathscr{C}_T$ with the associated adjunction $F_T \dashv G_T$. This adjunction determines a comonad on $\mathscr{C}_T$ which further determines the

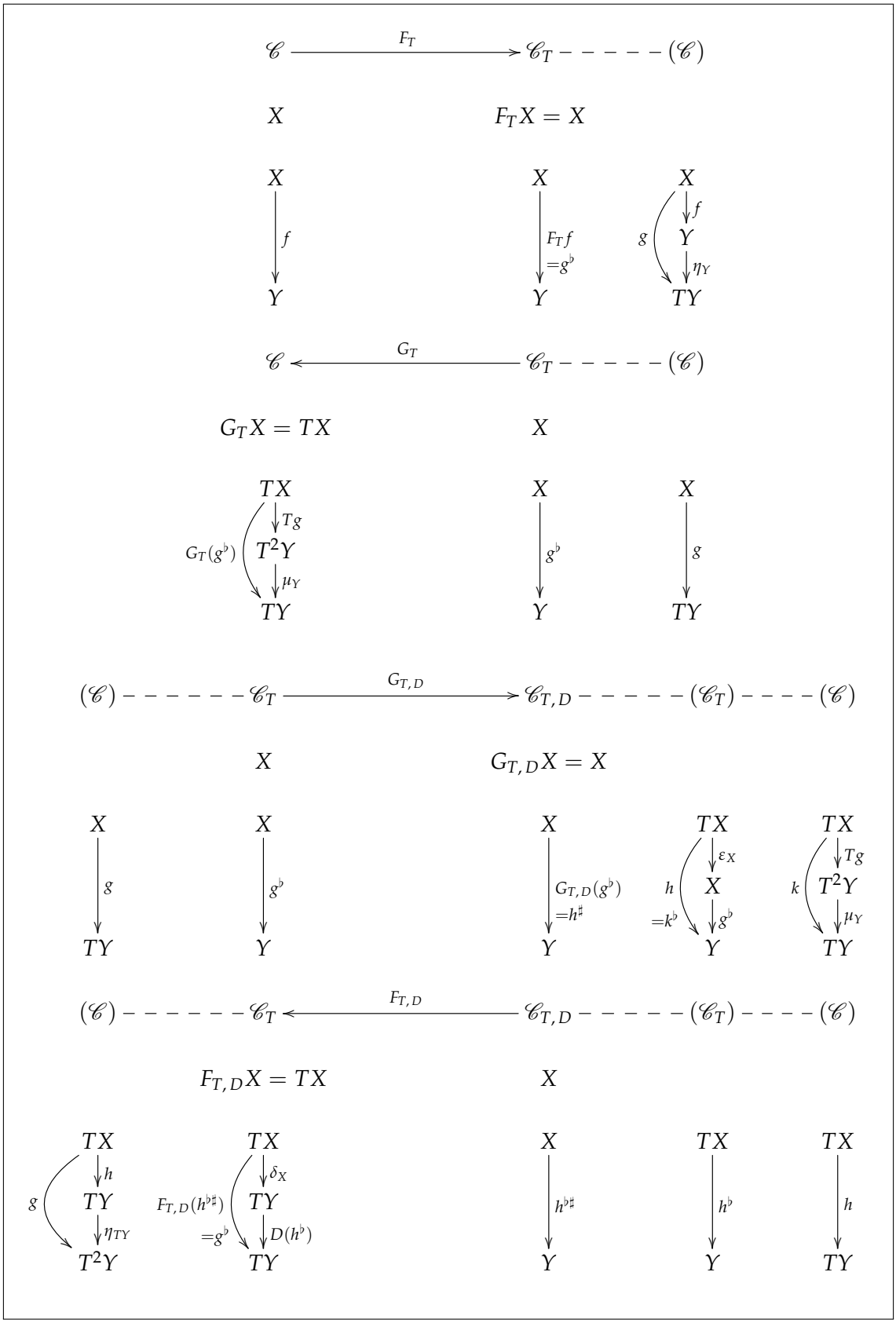


Figure 3.1: Description of the coKleisli-on-Kleisli construction associated to a monad

coKleisli category $\mathscr{C}_{T,D}$ with the associated adjunction $F_{T,D} \dashv G_{T,D}$.

Let $\mathscr{C}$ be the category of sets. It is closed under the *categorical coproduct* or *disjoint union*, denoted '+'. The left and right inclusions associated to '+' are denoted $\mathtt{inl}_{X,Y} : X \to X + Y$ and $\mathtt{inr}_{X,Y} : Y \to X + Y$, for each set X, Y. We consider a distinguished set E called the *set of exceptions*.

In the following, the objects denoted E^i represent the same set (the set of exceptions); superscript i is used to indicate which copy of the set E is considered. Similarly, the notation e^i refers to an element e in E^i.

We now consider the *exceptions monad* (or *coproduct monad*) (T, η, μ) on $\mathscr{C}$ composed of:

- the endofunctor $T : \mathscr{C} \to \mathscr{C}$:
  - on objects, for each X in $\mathscr{C}$, $TX = X + E$ in $\mathscr{C}$.
  - on arrows, $Tf = f + id_E : X + E \to Y + E$ in $\mathscr{C}$ for each $f : X \to Y$ in $\mathscr{C}$.
- the unit $\eta : Id_{\mathscr{C}} \Rightarrow T$:
  - $\eta_X = \mathtt{inl}_{X,E} : X \to X + E$ in $\mathscr{C}$, for each X in $\mathscr{C}$.

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & X + E^1 \\ x & \longmapsto & x \end{array}$$

- the multiplication $\mu : T^2 \Rightarrow T$:
  - $\mu_X = \left[id_{X+E} \mid \mathtt{inr}_{X,E} \right] : X + E + E \to X + E$ in $\mathscr{C}$, for each X in $\mathscr{C}$.

$$\begin{array}{ccc} X + E^1 + E^2 & \xrightarrow{\mu_X} & X + E^3 \\ x & \longmapsto & x \\ e^1 & \longmapsto & e^3 \\ e^2 & \longmapsto & e^3 \end{array}$$

As in Section 3.1.2, the monad (T, η, μ) determines a *Kleisli category* $\mathscr{C}_T$ and an associated adjunction $F_T \dashv G_T : \mathscr{C}_T \to \mathscr{C}$ with the following settings:

$$T \overset{\text{def}}{=} - + E \qquad \mathscr{C} \underset{G_T}{\overset{F_T}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathscr{C}_T$$

$$\eta : Id \Rightarrow T \qquad F_T \dashv G_T \qquad \varepsilon : T \Rightarrow Id$$

- The categories $\mathscr{C}$ and $\mathscr{C}_T$ have the same objects and there is a morphism $f^\flat : X \to Y$ in $\mathscr{C}_T$ for each morphism $f : X \to Y + E$ in $\mathscr{C}$. So that there is a bijection defined as:

$$(\varphi_T)_{X,Y} : Hom_{\mathscr{C}_T}(X, Y) \overset{\cong}{\to} Hom_{\mathscr{C}}(X, Y + E) \tag{3.30}$$

$$f^\flat \leftarrow\!\shortmid f. \tag{3.31}$$

- For each object X in $\mathscr{C}_T$, the identity arrow is $id_X = h^\flat : X \to X$ in $\mathscr{C}_T$ where $h = \eta_X : X \to X + E$ in $\mathscr{C}$.

- The composition of a pair of morphisms $f^\flat \colon X \to Y$ and $g^\flat \colon Y \to Z$ in $\mathscr{C}_T$ is given by the Kleisli composition, $g^\flat \circ f^\flat = h^\flat$ where $h = \mu_Z \circ Tg \circ f \colon X \to Z+E$ in $\mathscr{C}$.

$$\begin{array}{ccccccc} X & \xrightarrow{f} & Y+E^1 & \xrightarrow{Tg} & Z+E^2+E^3 & \xrightarrow{\mu_Z} & Z+E^4 \\ x & \longmapsto & y & \longmapsto & z & \longmapsto & z \\ & & & & e^2 & \longmapsto & e^4 \\ & & e^1 & \longmapsto & e^3 & \longmapsto & e^4 \end{array}$$

- The functor $F_T \colon \mathscr{C} \to \mathscr{C}_T$ is the identity on objects. On morphisms, $F_T f = (\eta_Y \circ f)^\flat = h^\flat$ in $\mathscr{C}_T$ for each $f \colon X \to Y$ in $\mathscr{C}$ and some $h^\flat$ in $\mathscr{C}_T$ such that $h = \eta_Y \circ f$ in $\mathscr{C}$.

$$\begin{array}{ccccc} X & \xrightarrow{f} & Y & \xrightarrow{\eta_Y} & Y+E^1 \\ x & \longmapsto & y & \longmapsto & y \end{array}$$

- The functor $G_T \colon \mathscr{C}_T \to \mathscr{C}$ maps each $X \in \mathscr{C}_T$ to $X + E$ in $\mathscr{C}$. On morphisms, $G_T(g^\flat) = \mu_Y \circ Tg$ for each $g^\flat \colon X \to Y$ in $\mathscr{C}_T$.

$$\begin{array}{ccccccc} X+E^1 & \xrightarrow{Tg} & Y+E^2+E^3 & \xrightarrow{\mu_Y} & Y+E^4 \\ x & \longmapsto & y & \longmapsto & y \\ & & e^2 & \longmapsto & e^4 \\ e^1 & \longmapsto & e^3 & \longmapsto & e^4 \end{array}$$

The adjunction $F_T \dashv G_T : \mathscr{C}_T \to \mathscr{C}$ determines a comonad (D, ε, δ) on $\mathscr{C}_T$ with

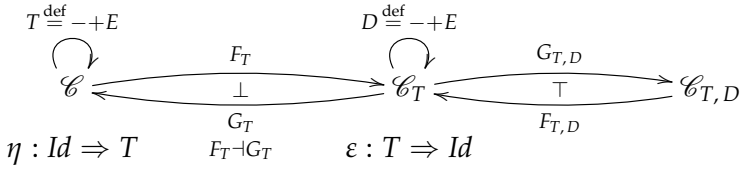
- the endofunctor $D \colon \mathscr{C}_T \to \mathscr{C}_T$:
  - on objects, for each X in $\mathscr{C}_T$, $DX = X + E$ in $\mathscr{C}_T$.
  - on arrows, thanks to Equation (3.12), $D(g^\flat) = (\eta_{Y+E} \circ \mu_Y \circ Tg)^\flat = h^\flat$, for each $g^\flat \colon X \to Y$ and some $h^\flat$ in $\mathscr{C}_D$ such that $h = \eta_{Y+E} \circ \mu_Y \circ Tg \colon X + E \to Y + E + E$ in $\mathscr{C}$.

$$\begin{array}{ccccccccc} X+E^1 & \xrightarrow{Tg} & Y+E^2+E^3 & \xrightarrow{\mu_Y} & Y+E^4 & \xrightarrow{\eta_{Y+E}} & Y+E^5+E^6 \\ x & \longmapsto & y & \longmapsto & y & \longmapsto & y \\ & & e^2 & \longmapsto & e^4 & \longmapsto & e^5 \\ e^1 & \longmapsto & e^3 & \longmapsto & e^4 & \longmapsto & e^5 \end{array}$$

- the counit $\varepsilon \colon D \Rightarrow Id_{\mathscr{C}_T}$:
  - $\varepsilon_X = (id_{X+E})^\flat \colon X + E \to X$ in $\mathscr{C}_T$, for each X in $\mathscr{C}_T$.
- the comultiplication $\delta \colon D \Rightarrow D^2$:
  - $\delta_X = F_T(\eta_{GX}) = F_T(\eta_{X+E}) = (\eta_{X+E+E} \circ \eta_{X+E})^\flat \colon X + E \to X + E + E$ in $\mathscr{C}_T$, for each X in $\mathscr{C}_T$.

$$\begin{array}{ccccccc} X+E^1 & \xrightarrow{\eta_{X+E}} & X+E^2+E^3 & \xrightarrow{\eta_{X+E+E}} & X+E^4+E^5+E^6 \\ x & \longmapsto & x & \longmapsto & x \\ e^1 & \longmapsto & e^2 & \longmapsto & e^4 \\ & & e^3 & \longmapsto & e^5 \end{array}$$

Let $\mathscr{C}_{T,D}$ be the coKleisli category of the comonad (D, ε, δ) and let $F_{T,D} \dashv G_{T,D} \colon \mathscr{C}_T \to \mathscr{C}_{T,D}$ be the associated adjunction, defined as follows:



1. The categories $\mathscr{C}$ and $\mathscr{C}_T$ have the same objects and there is a morphism $f^\flat \colon X \to Y$ in $\mathscr{C}_T$ for each morphism $f \colon X \to Y + E$ in $\mathscr{C}$.

2. The categories $\mathscr{C}_T$ and $\mathscr{C}_{T,D}$ have the same objects and there is a morphism $h^\sharp \colon X \to Y$ in $\mathscr{C}_{T,D}$ for each, $g \colon X + E \to Y$ in $\mathscr{C}_T$.

3. As a trivial consequence of above items 1 and 2, the categories $\mathscr{C}$ and $\mathscr{C}_{T,D}$ also have the same objects and there is a morphism $k^{\flat\sharp} \colon X \to Y$ in $\mathscr{C}_{T,D}$ for each $k \colon X + E \to Y + E$ in $\mathscr{C}$. So that $\mathscr{C}_{T,D}$ is the full image category of the functor $(- + E)$.

4. Due to Point 2 in Proposition 4.5.7, for each object X in $\mathscr{C}_{T,D}$, the arrow is $id_X = k^{\flat\sharp} \colon X \to X$ in $\mathscr{C}_{T,D}$ where $k = id_{X+E} \colon X + E \to X + E$ in $\mathscr{C}$.

5. Due to the Point 3 in Proposition 4.5.7, the composition of a pair of morphisms $f^{\flat\sharp} \colon X \to Y$ and $g^{\flat\sharp} \colon Y \to Z$ in $\mathscr{C}_{T,D}$ is given by $g \circ f \colon X + E \to Z + E$ in $\mathscr{C}$.

6. The functor $G_{T,D} \colon \mathscr{C}_T \to \mathscr{C}_{T,D}$ is the identity on objects. On morphisms, thanks to Equation (3.27), we have $G_{T,D}(g^\flat) = h^\sharp$ such that $h = k^\flat$ and $k = \mu_Y \circ Tg \colon X + E \to Y + E$ in $\mathscr{C}$, for each $g^\flat \colon X \to Y$ in $\mathscr{C}_T$.

$$\begin{array}{ccccc} X+E^1 & \xrightarrow{Tg} & Y+E^2+E^3 & \xrightarrow{\mu_Y} & Y+E^4 \\ x & \longmapsto & y & \longmapsto & y \\ & & e^2 & \longmapsto & e^4 \\ e^1 & \longmapsto & e^3 & \longmapsto & e^4 \end{array}$$

7. The composition $G_{T,D} \circ F_T$ is identity on objects. On morphisms, due to Equation (3.29), we have:

$$G_{T,D}F_T(f) = (Tf)^{\flat\sharp} = h^{\flat\sharp} \text{ such that } h = Tf \colon X + E \to Y + E \text{ in } \mathscr{C}. \tag{3.32}$$

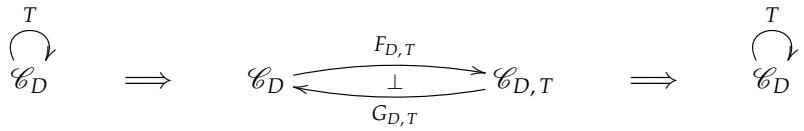
   Thus, the composition $G_{T,D} \circ F_T$ is the functor E in Theorem 3.2.5.

8. The functor $F_{T,D} \colon \mathscr{C}_{T,D} \to \mathscr{C}_T$ maps each X in $\mathscr{C}_{T,D}$ to $X + E$ in $\mathscr{C}_T$. On morphisms, due to Equation (3.28), we obtain $F_{T,D}(h^{\flat\sharp}) = g^\flat$ such that $g = \eta_{Y+E} \circ h$ in $\mathscr{C}$, for each $h^{\flat\sharp} \colon X \to Y$ in $\mathscr{C}_{T,D}$.

$$\begin{array}{ccccc} X+E^1 & \xrightarrow{h} & Y+E^2 & \xrightarrow{\eta_{Y+E}} & Y+E^3+E^4 \\ x & \longmapsto & y & \longmapsto & y \\ & & e^2 & \longmapsto & e^3 \\ e^1 & \longmapsto & y & \longmapsto & y \\ & & e^2 & \longmapsto & e^3 \end{array}$$

3.3 The Kleisli-on-coKleisli construction associated to a comonad

The adjunction, given in Sections 3.1.3, $F_D \dashv G_D : \mathscr{C}_D \to \mathscr{C}$ determines a monad (T, η, μ) on $\mathscr{C}_T$. This monad further determines several adjunctions, including: $F_{D,T} \dashv G_{D,T} : \mathscr{C}_D \to \mathscr{C}_{D,T}$ where $\mathscr{C}_{D,T}$ is the *Kleisli category* of (T, η, μ). The associated adjunction $F_{D,T} \dashv G_{D,T}$ determines back the monad (T, η, μ).



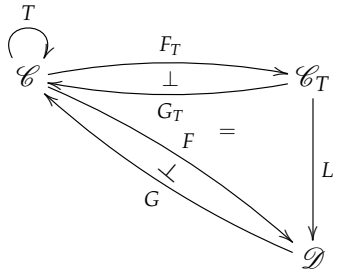
Besides, we show in Theorem 3.3.4 that the category $\mathscr{C}_{D,T}$ is the *full image category* of the endofunctor D.

This section studies the composition of the coKleisli adjunction $F_D \dashv G_D : \mathscr{C} \to \mathscr{C}_D$ associated to a comonad D, as detailed in Section 3.1.3, with the Kleisli adjunction $F_{D,T} \dashv G_{D,T} : \mathscr{C}_{D,T} \to \mathscr{C}_D$ associated to a monad T as given in Section 3.1.2. Note also that T is determined by $F_D \dashv G_D$. As a result of this composition, we obtain the Kleisli-on-coKleisli adjuntion construction to a comonad D. The generic settings provided by such an approach will be applied to the *states monad* in Section 3.3.3. The main aim of which will become explicit in Section 5.1 when we interpret the *decorated logic for the state*. This logic proposes a formalism to prove properties of programs with the state effect.

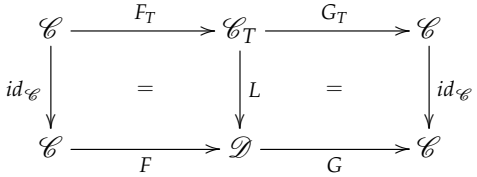
3.3.1 The comparison theorem for the Kleisli construction

Let us start with the proof of comparison theorem for the Kleisli construction. A specialization of this result will be used in Proposition 3.3.4, which is important for highlighting the relation among the categories defined in Section 3.3.2.

Theorem 3.3.1. (The comparison theorem for the Kleisli construction) *Let $F \dashv G : \mathscr{D} \to \mathscr{C}$ be an adjunction and let (T, η, μ) be the associated monad on $\mathscr{C}$. Then, there is a unique functor $L : \mathscr{C}_T \to \mathscr{D}$ such that $GL = G_T$ and $LF_T = F$, where $\mathscr{C}_T$ is the Kleisli category of (T, η, μ), with the associated adjunction $F_T \dashv G_T : \mathscr{C}_T \to \mathscr{C}$.*



Proof. Let us first assume that $L : \mathscr{C}_T \to \mathscr{D}$ is a functor satisfying $GL = G_T$ and $LF_T = F$.



Let $\theta_{X,Y}\colon Hom_{\mathscr{C}_T}(F_TX,Y) \xrightarrow{\cong} Hom_{\mathscr{C}}(X,G_TY)$ be a bijection associated to the adjunction $F_T \dashv G_T$. Similarly, let $\psi_{X,Y}\colon Hom_{\mathscr{D}}(FX,Y) \xrightarrow{\cong} Hom_{\mathscr{C}}(X,GY)$ be a bijection associated to the adjunction $F \dashv G$. Since both the unit of the adjunction $F_T \dashv G_T$ and the unit of the adjunction $F \dashv G$ are the unit η of the monad (T,η,μ) by [ML71, Ch. IV, §7, Proposition 1], we obtain the commutative diagram below:

$$\begin{array}{ccc} Hom_{\mathscr{C}_T}(F_TX,Y) & \xrightarrow{\theta_{X,Y}} & Hom_{\mathscr{C}}(X,G_TY) \\ \Big\downarrow{\scriptstyle L_{F_TX,Y}} & = & \Big\downarrow{\scriptstyle id_{X,G_TY}} \\ Hom_{\mathscr{D}}(LF_TX,LY) & & Hom_{\mathscr{C}}(X,G_TY) \\ \| & & \| \\ Hom_{\mathscr{D}}(FX,LY) & \xrightarrow{\psi_{X,LY}} & Hom_{\mathscr{C}}(X,GLY) \end{array}$$

Therefore, $L_{F_TX,Y} = \psi^{-1}_{X,LY} \circ \theta_{X,Y}$. This formula ensures that the functor L is *unique*. Let us simplify it: by Equation (3.4) in Proposition 3.1.4, we have:

$$\theta_{X,Y} f^\flat = G_T f^\flat \circ \eta_X\colon X \to G_TY, \text{ for each } f^\flat\colon F_TX = X \to Y \text{ in } \mathscr{C}_T.$$

Since $G_T f^\flat = \mu_Y \circ Tf$ in $\mathscr{C}$, for each $f^\flat\colon X \to Y$ in $\mathscr{C}_T$, by Equation (3.11), we have

$$\theta_{X,Y} f^\flat = \mu_Y \circ Tf \circ \eta_X\colon X \to G_TF_TY = G_TY.$$

Thanks to the naturality of η, we get $\theta_{X,Y} f^\flat = \mu_Y \circ \eta_{TY} \circ f$. The monadic axiom ensuring $\mu_Y \circ \eta_{TY} = id_{TY}$ yields $\theta_{X,Y} f^\flat = f\colon X \to G_TY$. Presumed that $G_T = GL$ and since F_T is the identity on objects, we have

$$\theta_{X,Y} f^\flat = f\colon X \to GLY \text{ and } LF_TY = LY = FY.$$

Now, by Equation (3.5) in Proposition 3.1.4, we obtain

$$\psi^{-1}_{X,LY} f = \varepsilon_{LY} \circ Ff = \varepsilon_{FY} \circ Ff = \psi^{-1}_{X,FY} f \text{ for each } f\colon X \to GFY \text{ in } \mathscr{C}.$$

Hence,

$$\psi^{-1}_{X,LY}(\theta_{X,Y} f^\flat) = \psi^{-1}_{X,FY} f = \varepsilon_{FY} \circ Ff.$$

In other words: given a functor L satisfying $GL = G_T$ and $LF_T = F$, then it must be such that $LX = FX$ for each object X in $\mathscr{C}_T$ and

$$Lf^\flat = \varepsilon_{FY} \circ Ff \text{ in } \mathscr{D} \text{ for each } f^\flat\colon X \to Y \text{ in } \mathscr{C}_T. \tag{3.33}$$

We additionally need to prove that the mapping $L\colon C_T \to \mathscr{D}$, characterized by $LX = X$ and $Lf^\flat = \varepsilon_Y \circ Ff$, is a functor satisfying $GL = G_T$ and $LF_T = F$:

1. For each X in $\mathscr{C}_T$, due to the fact that $id_X = (\eta_X)^\flat$ in $\mathscr{C}_T$, we have $L(id_X) = L((\eta_X)^\flat) = \varepsilon_{FX} \circ F\eta_X$. By [ML71, Ch. IV, §1, Theorem 1], we have

   $$\varepsilon_{FX} \circ F\eta_X = id_{FX} = id_{LX}.$$

   For each pair of morphisms $f^\flat\colon X \to Y$ and $g^\flat\colon Y \to Z$ in $\mathscr{C}_T$, by Kleisli composition, we get

   $$L(g^\flat \circ f^\flat) = \varepsilon_{FZ} \circ FG\varepsilon_{FZ} \circ FGFg \circ Ff.$$

   Since ε is natural, we obtain $\varepsilon_{FZ} \circ Fg \circ \varepsilon_{FY} \circ Ff$ which is $L(g^\flat) \circ L(f^\flat)$ in $\mathscr{D}$. Hence, $L\colon \mathscr{C}_T \to \mathscr{D}$ is a functor.

2. For each object X in $\mathscr{C}_T$, $LX = FX$ in $\mathscr{D}$ and $GLX = GFX = TX = G_T X$ in $\mathscr{C}$. For each morphism $f^\flat \colon X \to Y$ in $\mathscr{C}_T$, $Lf^\flat = \varepsilon_{FY} \circ Ff$ in D by definition. Hence,

$$GLf^\flat = G\varepsilon_{FY} \circ GFf.$$

Similarly, Equation (3.11) gives:

$$G_T f^\flat = G\varepsilon_{FY} \circ GFf.$$

We get $GLf^\flat = G_T f^\flat$ for each mapping $f^\flat$, therefore,

$$GL = G_T.$$

3. F_T is the identity on objects, thus $LF_T X = LX = FX$. For each morphism $f \colon X \to Y$ in $\mathscr{C}$, we have $F_T f = (\eta_Y \circ f)^\flat$ in $\mathscr{C}_T$, by definition. So that

$$LF_T f = L(\eta_Y \circ f)^\flat = \varepsilon_{FY} \circ F\eta_Y \circ Ff.$$

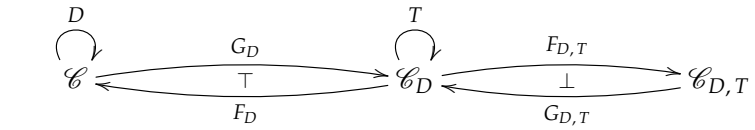
Due to ε and η being natural, we have $\varepsilon_{FY} \circ F\eta_Y = id_{FY}$ yielding $LF_T f = Ff$ for each mapping f, thus, $LF_T = F$. □

3.3.2 The Kleisli-on-coKleisli construction

Let (D, ε, δ) be a comonad defined on a category $\mathscr{C}$. It determines a coKleisli category $\mathscr{C}_D$ along with the associated adjunction $F_D \dashv G_D : \mathscr{C} \to \mathscr{C}_D$. Let $(T = G_D F_D, \eta, \mu)$ be the monad that the adjunction $F_D \dashv G_D : \mathscr{C} \to \mathscr{C}_D$ determines on $\mathscr{C}_D$. Refer back to Section 3.1.3, for the related details.

Remark 3.3.2. Note that the details of below items, from 1 to 4, are depicted in Figure 3.2.

Now, let $\mathscr{C}_{D,T}$ be the Kleisli category determined by (T, η, μ) and let $F_{D,T} \dashv G_{D,T} \colon \mathscr{C}_{D,T} \to \mathscr{C}_D$ be the associated adjunction with the following settings:



$\varepsilon : D \Rightarrow Id$ $\quad F_D \dashv G_D \quad$ $\eta : Id \Rightarrow T$

1. The categories $\mathscr{C}$ and $\mathscr{C}_D$ have the same objects and there is a morphism $f^\sharp \colon X \to Y$ in $\mathscr{C}_D$, for each $f \colon DX \to Y$ in $\mathscr{C}$.

2. The categories $\mathscr{C}_D$ and $\mathscr{C}_{D,T}$ have the same objects and there is a morphism $h^\flat \colon X \to Y$ in $\mathscr{C}_{D,T}$, for each $h \colon X \to DY$ in $\mathscr{C}_D$.

3. The functor $F_{D,T} \colon \mathscr{C}_D \to \mathscr{C}_{D,T}$ is the identity on objects. On morphisms,

$$F_{D,T}(g^\sharp) = (\eta_Y \circ g^\sharp)^\flat = h^\flat \text{ for each } g^\sharp \colon X \to Y \text{ in } \mathscr{C}_D. \tag{3.34}$$

Let $h = (\eta_Y \circ g^\sharp) \colon X \to DY$ in $\mathscr{C}_D$. Since $\eta_Y = (id_{DY})^\sharp$ in $\mathscr{C}_D$, we get $h = (id_{DY})^\sharp \circ g^\sharp = k^\sharp$ for some $k^\sharp$ in $\mathscr{C}_D$. By coKleisli composition, we end up with

$$F_{D,T}(g^\sharp) = h^\flat \text{ such that } h = k^\sharp \text{ and } k = Dg \circ \delta_X \colon DX \to DY \text{ in } \mathscr{C}. \tag{3.35}$$

4. The functor $G_{D,T}\colon \mathscr{C}_{D,T} \to \mathscr{C}_D$ maps each X in $\mathscr{C}_{D,T}$ to DX in $\mathscr{C}_D$. On morphisms, $G_{D,T}(h^{\sharp\flat}) = \mu_Y \circ Th^{\sharp}$ for each $h^{\sharp\flat}\colon X \to Y$ in $\mathscr{C}_{D,T}$. Let us introduce mappings $g^{\sharp}, a^{\sharp}$ and $b^{\sharp}$ in $\mathscr{C}_D$ and set

$$g^{\sharp} = G_{D,T}(h^{\sharp\flat}), a^{\sharp} = \mu_Y \text{ and } b^{\sharp} = Th^{\sharp}.$$

Thus, we have $g^{\sharp} = a^{\sharp} \circ b^{\sharp}$ and by coKleisli composition, we get:

$$g = a \circ Db \circ \delta_{DX}\colon D^2X \to DY \text{ in } \mathscr{C}.$$

(a) We have $\mu_Y = G_D(\varepsilon_{F_DY})$ by definition. We also have $F_DY = F_DG_DY = DY$ due to G_D being identity on objects. So that $\mu_Y = G_D(\varepsilon_{DY})$. Since $a^{\sharp} = \mu_Y$, we have $a^{\sharp} = G_D(\varepsilon_{DY})$. Using the fact that $G_Df = (f \circ \varepsilon_X)^{\sharp}$ for each $f\colon X \to Y$ in $\mathscr{C}$ (see Equation (3.19)), we derive:

$$a^{\sharp} = (\varepsilon_{DY} \circ \varepsilon_{D^2Y})^{\sharp} \text{ hence } a = (\varepsilon_{DY} \circ \varepsilon_{D^2Y}) \text{ in } \mathscr{C}.$$

(b) We also have $b^{\sharp} = Th^{\sharp} = G_DF_Dh^{\sharp}$ such that $h\colon DX \to DY$ in $\mathscr{C}$. Provided by Equation (3.20) that $F_D(h^{\sharp}) = Dh \circ \delta_X$. Therefore, we have $G_D(Dh \circ \delta_X) = (Dh \circ \delta_X \circ \varepsilon_{DX})^{\sharp} = b^{\sharp}$. So that $b = (Dh \circ \delta_X \circ \varepsilon_{DX})$ in $\mathscr{C}$. By rewriting a and b in g, we obtain

$$g = \varepsilon_{DY} \circ \varepsilon_{D^2Y} \circ D^2h \circ D\delta_X \circ D\varepsilon_{DX} \circ \delta_{DX}.$$

Thanks to the comonadic axiom stating $D\varepsilon_{DX} \circ \delta_{DX} = id_{D^2X}$ we have $g = \varepsilon_{DY} \circ \varepsilon_{D^2Y} \circ D^2h \circ D\delta_X$. Since ε is natural, used three times, we get $g = \varepsilon_{DY} \circ Dh \circ \varepsilon_{D^2X} \circ D\delta_X = h \circ \varepsilon_{DX} \circ \varepsilon_{D^2X} \circ D\delta_X = h \circ \varepsilon_{DX} \circ \delta_X \circ \varepsilon_{DX}$. Due to the comonadic axiom ensuring $\varepsilon_{DX} \circ \delta_X = id_{Dx}$, we write

$$G_{D,T}(h^{\sharp\flat}) = g^{\sharp} \text{ where } g = h \circ \varepsilon_{DX} \tag{3.36}$$

Then, the associated monad to the adjunction $F_{D,T} \dashv G_{D,T}$ is actually the monad (T, μ, η) where $T = G_{D,T}F_{D,T}$.

5. The composition $F_{D,T} \circ G_D$ is identity on objects. On morphisms, due to Equation (3.19), we have:

$$F_{D,T}G_D(f) = F_{D,T}((f \circ \varepsilon_X)^{\sharp}) \text{ for each } f\colon X \to Y \text{ in } \mathscr{C}.$$

We further have:

$$F_{D,T}((f \circ \varepsilon_X)^{\sharp}) = (\eta_Y \circ (f \circ \varepsilon_X)^{\sharp})^{\flat} \text{ in } \mathscr{C}_{D,T} \text{ by Equation (3.35).}$$

Provided that $\eta_X = (id_{DX})^{\sharp}$ in $\mathscr{C}_D$, we get $F_{D,T}G_D(f) = ((id_{DX})^{\sharp} \circ (f \circ \varepsilon_X)^{\sharp})^{\flat}$ and $F_{D,T}G_D(f) = (Df \circ D\varepsilon_X \circ \delta_X)^{\sharp\flat}$ by coKleisli composition. Due to the comonadic axiom stating $D\varepsilon_X \circ \delta_X = id_{DX}$, it simplifies into

$$F_{D,T}G_D(f) = (Df)^{\sharp\flat} = h^{\sharp\flat} \text{ such that } h = Df\colon DX \to DY \text{ in } \mathscr{C}. \tag{3.37}$$

Proposition 3.3.3. *1. The categories $\mathscr{C}$ and $\mathscr{C}_{D,T}$ have the same objects and there is a morphism $k^{\sharp\flat}\colon X \to Y$ in $\mathscr{C}_{D,T}$ for each $k\colon DX \to DY$ in $\mathscr{C}$.*

2. For each object X in $\mathscr{C}_{D,T}$, the identity arrow is $id_X = k^{\sharp\flat}\colon X \to X$ in $\mathscr{C}_{D,T}$ where $k = id_{DX}\colon DX \to DX$ in $\mathscr{C}$.

3. *The composition of a pair of morphisms* $f^{\sharp\flat}\colon X \to Y$ *and* $g^{\sharp\flat}\colon Y \to Z$ *in* $\mathscr{C}_{D,T}$ *is given by* $g^{\sharp\flat} \circ f^{\sharp\flat} = k^{\sharp\flat}$ *where* $k = g \circ f\colon DX \to DZ$ *in* $\mathscr{C}$.

Proof. 1. It is the trivial consequence of items 1 and 2 of Section 3.3.2.

2. For each object X, we have $k^{\sharp\flat} = id_X\colon X \to X$ in $\mathscr{C}_{D,T}$, where $k^{\sharp} = \eta_X\colon X \to TX$ in $\mathscr{C}_D$, due to Equation (3.8). Since $\eta_X = (id_{F_DX})^{\sharp}$ in $\mathscr{C}_D$, by Equation (3.22), we obtain $k = id_{F_DX}$ in $\mathscr{C}$. Now, $F_DX = DX$ yields $k = id_{DX}\colon DX \to DX$ in $\mathscr{C}$.

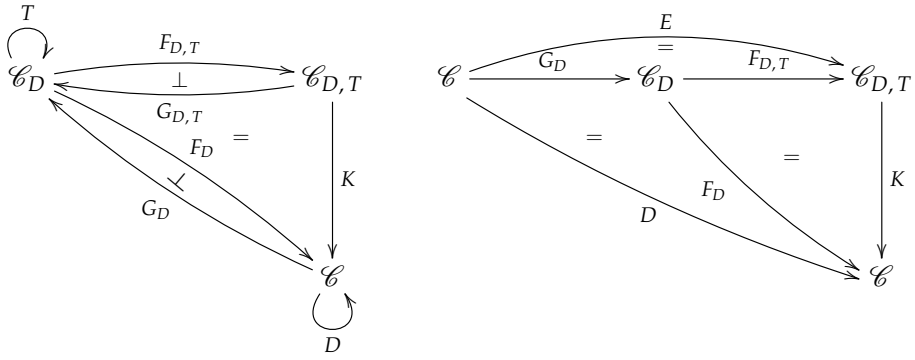
3. For each $g^{\sharp\flat}\colon Y \to Z$ and $f^{\sharp\flat}\colon X \to Y$ in $\mathscr{C}_{D,T}$, due to the Kleisli composition, we get $g^{\sharp} \circ f^{\sharp} = \mu_Z \circ T(g^{\sharp}) \circ f^{\sharp} = k^{\sharp}$ in $\mathscr{C}_D$. Since $T(g^{\sharp}) = (Dg \circ \delta_Y \circ \varepsilon_{DY})^{\sharp}$ and $\mu_Z = (\varepsilon_{DZ} \circ \varepsilon_{D^2Z})^{\sharp}$ respectively given by Equations (3.21) and (3.23), we have $k^{\sharp} = (\varepsilon_{DZ} \circ \varepsilon_{D^2Z})^{\sharp} \circ (Dg \circ \delta_Y \circ \varepsilon_{DY})^{\sharp} \circ f^{\sharp}$. Thanks to associativity of composition and coKleisli composition, we obtain $k^{\sharp} = (\varepsilon_{DZ} \circ \varepsilon_{D^2Z})^{\sharp} \circ (Dg \circ \delta_Y \circ \varepsilon_{DY} \circ Df \circ \delta_X)^{\sharp}$. Using coKleisli composition again, we get $k = \varepsilon_{DZ} \circ \varepsilon_{D^2Z} \circ D^2g \circ D\delta_Y \circ D\varepsilon_{DY} \circ D^2f \circ D\delta_X \circ \delta_X$ in $\mathscr{C}$. Now, we use naturality of ε twice and obtain $k = \varepsilon_{DZ} \circ Dg \circ D\varepsilon_{DY} \circ D\delta_Y \circ Df \circ D\varepsilon_{DX} \circ D\delta_X \circ \delta_X$. This simplifies into $k = \varepsilon_{DZ} \circ Dg \circ Df \circ \delta_X$, thanks to comonadic axiom ensuring $\varepsilon_{DX} \circ \delta_X = id_{DX}$ for each object X. Lastly, naturality of ε gives $k = g \circ f \circ \varepsilon_{DX} \circ \delta_X$, by the same comonadic axiom, we end up with $k = g \circ f\colon DX \to DZ$ in $\mathscr{C}$.

□

Theorem 3.3.4. *Let* (D, δ, ε) *be a comonad on a category* $\mathscr{C}$ *and let* $\mathscr{C}_D$ *be the coKleisli category of* (D, δ, ε) *with the associated adjunction* $F_D \dashv G_D\colon \mathscr{C} \to \mathscr{C}_D$. *Let* (T, μ, η) *be the monad on* $\mathscr{C}_D$ *determined by the adjunction* $F_D \dashv G_D$. *And let* $\mathscr{C}_{D,T}$ *be the Kleisli category of* (T, μ, η) *with the associated adjunction* $F_{D,T} \dashv G_{D,T}\colon \mathscr{C}_{D,T} \to \mathscr{C}_D$. *Then;*

1. *there is a unique functor* $K\colon \mathscr{C}_{D,T} \to \mathscr{C}$ *such that* $KF_{D,T} = F_D$ *and* $G_DK = G_{D,T}$.

2. *the full image factorization of* D *is given by* $D = KE$ *where* $E = F_{D,T}G_D$.



3. *for each* X *in* $\mathscr{C}_D$, η_X *is split-mono thus* $F_{D,T}$ *is faithful.*

Proof. 1. We specialize the Theorem 3.3.1 by instantiating $F_T \dashv G_T$ with $F_{T,D} \dashv G_{T,D}$ and $F \dashv G$ with $F_T \dashv G_T$. Thus, we obtain the unique functor $K\colon \mathscr{C}_{D,T} \to \mathscr{C}_D$ such that $KF_{D,T} = F_D$ and $G_DK = G_{D,T}$.

2. The category $\mathscr{C}_{D,T}$ is the full image category of D, since it is made of objects X for each X in $\mathscr{C}$ and arrows $f^{\sharp\flat}\colon X \to Y$ for each $f\colon DX \to DY$ in $\mathscr{C}$. Recall that $D = F_DG_D$ by definition and $F_D = KF_{D,T}$ by point 1, so that $D = KF_{D,T}G_D = KE$. On the one hand, $E(X) = F_{D,T}G_D(X) = X$ and $E(f) = F_{D,T}G_D(f) = (Df)^{\sharp\flat}\colon DX \to$

DY thanks to Equation (3.37). On the other hand, $K(X) = DX$ for each object X and $K(g^{\sharp\flat}) = \varepsilon_{DY} \circ F_D(g^{\sharp})$ for each $g^{\sharp\flat}\colon X \to Y$ in $\mathscr{C}_{D,T}$ by Equation (3.33). Thanks to Equation (3.20), we obtain $K(g^{\sharp\flat}) = \varepsilon_{DY} \circ Dg \circ \delta_X$. Since ε is natural, we have $K(g^{\sharp\flat}) = \varepsilon_{DY} \circ \delta_Y \circ g$. The comonadic axiom ensuring $\varepsilon_{DY} \circ \delta_Y = id_{DY}$ gives $K(g^{\sharp\flat}) = g\colon DX \to DY$. Obviously, $KE(f) = K(Df^{\sharp\flat}) = Df\colon DX \to DY$ for each morphism $f\colon X \to Y$ and $KE(X) = K(X) = DX$ for each object X. Therefore, the *full image factorization of* D is given by the pair (K, E).

3. It is necessary to show the existence of a mapping $f^{\sharp}$ in $\mathscr{C}_D$ such that $f^{\sharp} \circ \eta_X = id_X$. Since $\eta_X = (id_{DX})^{\sharp}$ and $id_X = (\varepsilon_X)^{\sharp}$, we get $f^{\sharp} \circ (id_{DX})^{\sharp} = (\varepsilon_X)^{\sharp}$ in $\mathscr{C}_T$ and equivalently $\varepsilon_X = f \circ D(id_{FX=DX}) \circ \delta_X$ in $\mathscr{C}$ by coKleisli composition. It is trivial to show, by the comonadic property $id_{DX} = \varepsilon_{DX} \circ \delta_X$, that this equation is satisfied when f is chosen to be $\varepsilon_X \circ \varepsilon_{DX}$. So that η_X is split-mono, for each X in $\mathscr{C}_1$. Notice also that split-mono implies mono. Now, due to the point (i) of the dual of [ML71, Ch. IV, §3, Theorem 1], we conclude that $F_{D,T}$ is faithful.

□

We will use *Kleisli-on-coKleisli construction* in Section 5.1 where we interpret the *decorated logic for the state*. This logic proposes a formalism to prove properties of programs with the *state effect*.

3.3.3 Application to the state comonad on sets

In this section, we apply the *Kleisli-on-coKleisli construction associated to a comonad* to the *state comonad*. This means that we start with the state comonad defined on the category of sets $\mathscr{C}$ and then construct its coKleisli category $\mathscr{C}_D$ with the associated adjunction $F_D \dashv G_D$. This adjunction determines a monad on $\mathscr{C}_D$ which further determines the Kleisli category $\mathscr{C}_{D,T}$ with the associated adjunction $F_{D,T} \dashv G_{D,T}$.

Let $\mathscr{C}$ be the category of sets. It is closed under the *categorical product* (or *cartesian product*) denoted '$\times$'. The left and right projections associated to '$\times$' are denoted $\mathtt{fst}_{X,Y}\colon X \times Y \to X$ and $\mathtt{snd}_{X,Y}\colon X \times Y \to Y$, for each sets X, Y. In $\mathscr{C}$, we consider a distinguished *set of states* denoted S.

Now, let (D, ε, δ) be the *states comonad* (or *product comonad*) defined on $\mathscr{C}$ as:

- the endofunctor $D\colon \mathscr{C} \to \mathscr{C}$:
  - on objects, for each X in $\mathscr{C}$, $DX = X \times S$ in $\mathscr{C}$.
  - on arrows, $Df = f \times id_S\colon X \times S \to Y \times S$ in $\mathscr{C}$, for each $f\colon X \to Y$ in $\mathscr{C}$.
- the counit $\varepsilon\colon D \Rightarrow Id_{\mathscr{C}}$:
  - $\varepsilon_X = \mathtt{fst}_{X,S}\colon X \times S \to X$ in $\mathscr{C}$ for each X in $\mathscr{C}$.

$$\begin{array}{ccc} X \times S & \xrightarrow{\varepsilon_X} & X \\ (x, s) & \longmapsto & x \end{array}$$

- the comultiplication $\delta\colon D \Rightarrow D^2$:

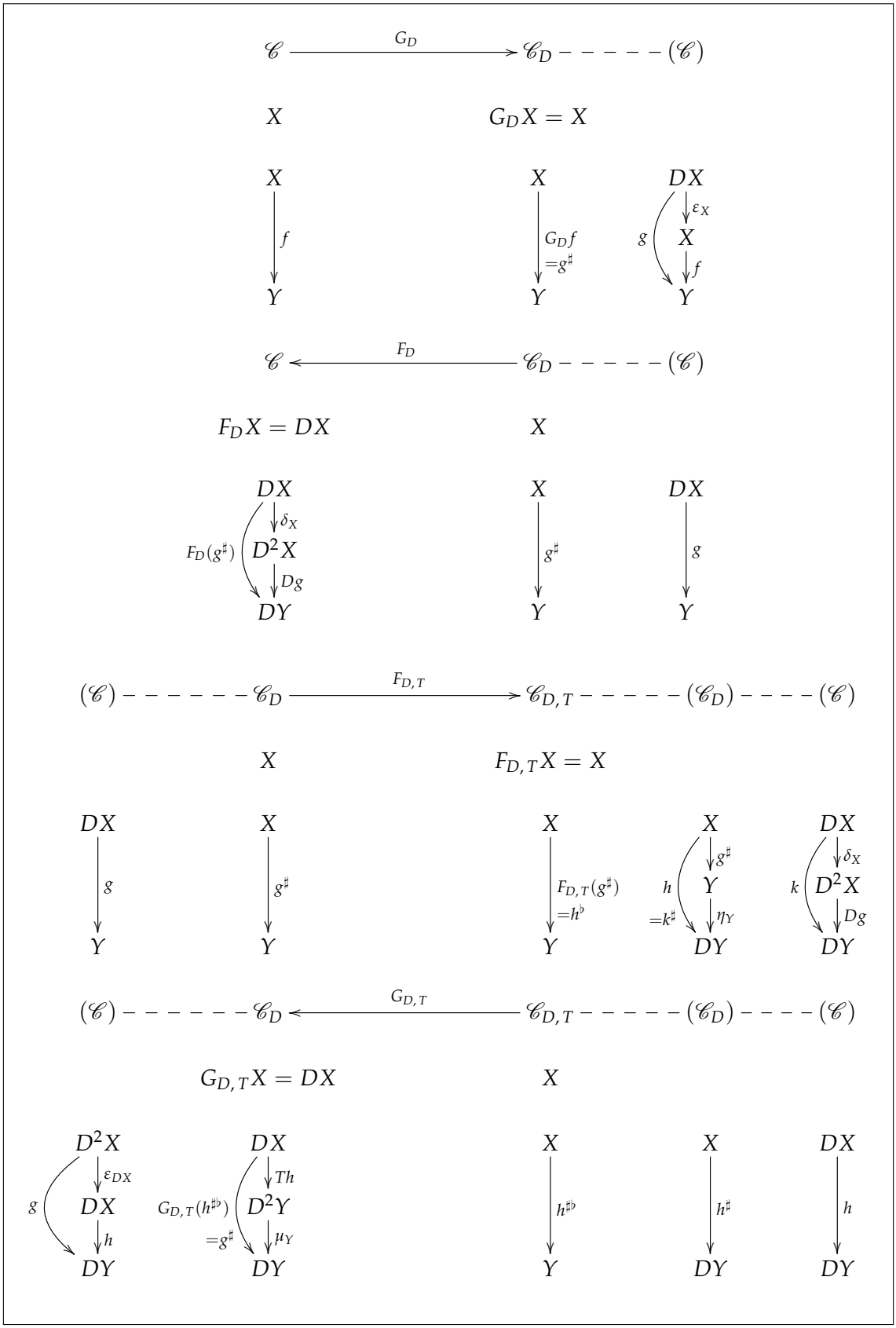
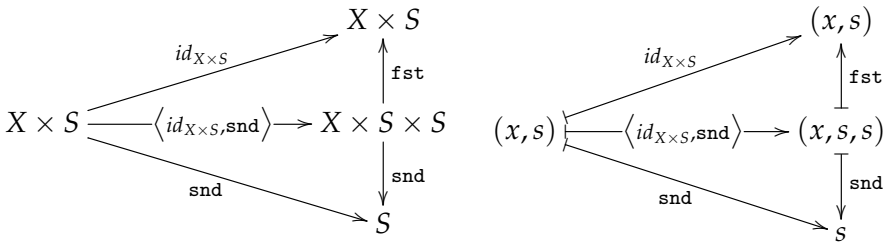
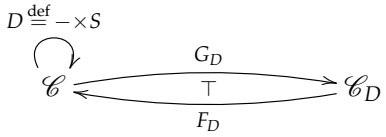


Figure 3.2: Description of the Kelisli-on-coKleisli construction associated to a comonad

– $\delta_X = \langle id_{X\times S}, \mathtt{snd}_{X,S}\rangle : X \times S \to X \times S \times S$ for each X in $\mathscr{C}$.



The comonad (D, ε, η) determines a *coKleisli category* $\mathscr{C}_D$ and an adjunction $F_D \dashv G_D : \mathscr{C} \to \mathscr{C}_D$ with the following settings:



$\varepsilon : D \Rightarrow Id \qquad G_D \vdash F_D \qquad \eta : Id \Rightarrow D$

- The categories $\mathscr{C}$ and $\mathscr{C}_D$ have the same objects and there is a morphism $f^\sharp : X \to Y$ in $\mathscr{C}_D$ for each morphism $f : X \times S \to Y$ in $\mathscr{C}$.
- For each object X in $\mathscr{C}_D$, the identity arrow is $id_X = h^\sharp : X \to X$ in $\mathscr{C}_D$ where $h = \varepsilon_X : X \times S \to X$ in $\mathscr{C}$.
- The composition of a pair of morphisms $f^\sharp : X \to Y$ and $g^\sharp : Y \to Z$ in $\mathscr{C}_D$ is given by the coKleisli composition, $g^\sharp \circ f^\sharp = h^\sharp$ where $h = g \circ Df \circ \delta_X : X \times S \to Z$ in $\mathscr{C}$.

$$\begin{array}{ccccccc} X \times S & \xrightarrow{\delta_X} & X \times S \times S & \xrightarrow{f \times id_S} & Y \times S & \xrightarrow{g} & Z \\ (x,s) & \longmapsto & (x,s,s) & \longmapsto & (y,s) & \longmapsto & z \end{array}$$

  where $y = f(x,s)$ and $z = g(y,s)$.

- The functor $G_D : \mathscr{C} \to \mathscr{C}_D$ is the identity on objects. On morphisms, $G_D f = (f \circ \varepsilon_X)^\sharp = h^\sharp$, for each $f : X \to Y$ in $\mathscr{C}$ and some $h^\sharp$ in $\mathscr{C}_D$ such that $h = f \circ \varepsilon_X$ in $\mathscr{C}$.

$$\begin{array}{ccccc} X \times S & \xrightarrow{\varepsilon_X} & X & \xrightarrow{f} & Y \\ (x,s) & \longmapsto & x & \longmapsto & y \end{array}$$

  where $y = f(x)$

- The functor $F_D : \mathscr{C}_D \to \mathscr{C}$ maps each X in $\mathscr{C}_D$ to $X \times S$ in $\mathscr{C}$. On morphisms, $F_D(g^\sharp) = Dg \circ \delta_X$ for each $g^\sharp : X \to Y$ in $\mathscr{C}_D$.

$$\begin{array}{ccccc} X \times S & \xrightarrow{\delta_X} & X \times S \times S & \xrightarrow{Dg} & Y \times S \\ (x,s) & \longmapsto & (x,s,s) & \longmapsto & (y,s) \end{array}$$

  where $y = g(x,s)$

The adjunction $F_D \dashv G_D : \mathscr{C} \to \mathscr{C}_D$ determines a monad (T, η, μ) on $\mathscr{C}_D$ defined as follows:

- the endofunctor $T\colon \mathscr{C}_D \to \mathscr{C}_D$:
  - on objects, for each X in $\mathscr{C}_D$, $TX = X \times S$ in $\mathscr{C}_D$.
  - on arrows, thanks to Equation (3.21), $T(g^\sharp) = (Dg \circ \delta_X \circ \varepsilon_{X\times S})^\sharp = h^\sharp$, for each $g^\sharp\colon X \to Y$ and some $h^\sharp$ in $\mathscr{C}_D$ such that $h = Dg \circ \delta_X \circ \varepsilon_{X\times S}\colon X \times S \times S \to Y \times S$ in $\mathscr{C}$.

$$\begin{array}{ccccccc} X\times S\times S & \xrightarrow{\varepsilon_{X\times S}} & X\times S & \xrightarrow{\delta_X} & X\times S\times S & \xrightarrow{Dg} & Y\times S \\ (x,s_1,s_2) & \longmapsto & (x,s_1) & \longmapsto & (x,s_1,s_1) & \longmapsto & (y,s_1) \end{array}$$

    where $y = g(x, s_1)$.

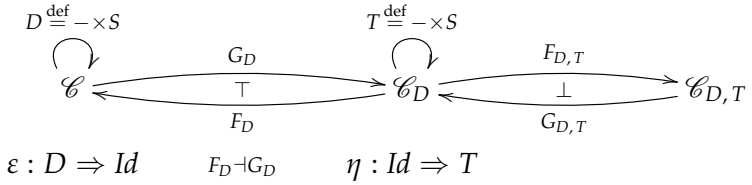
- the unit $\eta\colon Id_{\mathscr{C}_D} \Rightarrow T$:
  - $\eta_X = (id_{X\times S})^\flat\colon X \to X \times S$ in $\mathscr{C}_D$ for each X in $\mathscr{C}_D$.

$$\begin{array}{ccc} X\times S & \xrightarrow{id_{X\times S}} & X\times S \\ (x,s) & \longmapsto & (x,s) \end{array}$$

- the multiplication $\mu\colon T^2 \Rightarrow T$:
  - $\mu_X = G_D(\varepsilon_{F_D X}) = G_D(\varepsilon_{X\times S}) = (\varepsilon_{X\times S} \circ \varepsilon_{X\times S\times S})^\sharp\colon X \times S \times S \to X \times S$ for each X in $\mathscr{C}_D$.

$$\begin{array}{ccccc} X\times S\times S\times S & \xrightarrow{\varepsilon_{X\times S\times S}} & X\times S\times S & \xrightarrow{\varepsilon_{X\times S}} & X\times S \\ (x,s_1,s_2,s_3) & \longmapsto & (x,s_1,s_2) & \longmapsto & (x,s_1) \end{array}$$

Let $\mathscr{C}_{D,T}$ be the Kleisli category of the monad (T, μ, ε) and let $F_{D,T} \dashv G_{D,T}\colon \mathscr{C}_{D,T} \to \mathscr{C}_D$ be the associated adjunction, we have them defined as follows:



1. The categories $\mathscr{C}$ and $\mathscr{C}_D$ have the same objects and there is a morphism $h^\sharp\colon X \to Y$ in $\mathscr{C}_D$, for each $h\colon X \times S \to Y$ in $\mathscr{C}_D$.
2. The categories $\mathscr{C}_D$ and $\mathscr{C}_{D,T}$ have the same objects and there is a morphism $h^\flat\colon X \to Y$ in $\mathscr{C}_{D,T}$, for each $h\colon X \to Y \times S$ in $\mathscr{C}_D$.
3. As a trivial consequence of above points 1 and 2, the categories $\mathscr{C}_{D,T}$ and $\mathscr{C}$ have the same objects and there is a morphism $h^{\sharp\flat}\colon X \to Y \in \mathscr{C}_{D,T}$, for each $h\colon X \times S \to Y \times S$ in $\mathscr{C}$. So that $\mathscr{C}_{D,T}$ is the full image category of the functor $(- \times S)$.
4. Due to Point 2 in Proposition 3.3.3, for each object $X \in \mathscr{C}_{D,T}$, the identity arrow is $id_X = h^{\sharp\flat}\colon X \to X \in \mathscr{C}_{D,T}$ where $h = id_{X\times S}\colon X \times S \to X \times S$ in $\mathscr{C}_D$.
5. Due to Point 3 in Proposition 3.3.3, the composition of a pair of morphisms $f^{\sharp\flat}\colon X \to Y$ and $g^{\sharp\flat}\colon Y \to Z \in \mathscr{C}_{D,T}$ is given by $g \circ f\colon X \times S \to Z \times S$ in $\mathscr{C}$.

6. The functor $F_{D,T}\colon \mathscr{C}_D \to \mathscr{C}_{D,T}$ is the identity on objects. On morphisms, thanks to Equation (3.34), we have $F_{D,T}(g^\sharp) = h^\flat$ such that $h = k^\sharp$ and $k = Dg \circ \delta_X\colon X \times S \to Y \times S$ in $\mathscr{C}$.

$$\begin{aligned} X \times S &\xrightarrow{\delta_X} X \times S \times S \xrightarrow{Dg} Y \times S \\ (x,s) &\longmapsto (x,s,s) \longmapsto (y,s) \end{aligned}$$

7. The composition $F_{D,T} \circ G_D$ is the identity on objects. On morphisms, due to Equation (3.37), we have:

$$F_{D,T}G_D(f) = (Df)^{\sharp\flat} = h^{\sharp\flat} \text{ such that } h = Df\colon X \times S \to Y \times S \text{ in } \mathscr{C}.$$

   Thus, the composition $F_{D,T} \circ G_D$ is the functor E in Theorem 3.3.4.

8. The functor $G_{D,T}\colon \mathscr{C}_{D,T} \to \mathscr{C}_D$ maps each X in $\mathscr{C}_{D,T}$ to $X \times S$ in $\mathscr{C}_D$. On morphisms, due to Equation (3.36), we have $G_{D,T}(h^{\sharp\flat}) = g^\sharp$ such that $g = h \circ \varepsilon_{X\times S}$ in $\mathscr{C}$ for each $h^{\sharp\flat}\colon X \to Y$ in $\mathscr{C}_{D,T}$.

$$\begin{aligned} X \times S \times S &\xrightarrow{\varepsilon_{X\times S}} X \times S \xrightarrow{h} Y \times S \\ (x,s_1,s_2) &\longmapsto (x,s_1) \longmapsto (y,s_3) \end{aligned}$$

such that $(y,s_3) = h(x,s_1)$.

4

Decorated logics

We present two equational-based logics with categorical interpretations, in order to prove properties of programming languages with effects. We start with the monadic equational logic as in [Mog91], which can be interpreted in any category. Then, we extend it by adding decorations to its terms and equations. In fact, we propose two dual inference systems that can be instantiated using monads or comonads, respectively, so as to cope with different computational effects. The first inference system is interpreted in the Kleisli category of a monad and the coKleisli category of the associated comonad as in Section 3.2.2. Dually, the second one is interpreted in the coKleisli category of a comonad and the Kleisli category of the associated monad as in Section 3.3.2. Both logics combine a 3-tier effect system for terms, with a 2-tier system for equations made of "up-to-effects" and "strong" equations.

Section 4.1 defines the *monadic equational logic* $\mathcal{L}_{meq}$. This logic is extended into the *decorated logic for a monad* ($\mathcal{L}_{mon}$) in Section 4.2, where the categorical interpretation of $\mathcal{L}_{mon}$ by the coKleisli-on-Kleisli construction associated to a monad is also given. In Section 4.3, the decorated logic for a comonad ($\mathcal{L}_{com}$) is detailed. There, we use the Kleisli-on-coKleisli construction associated to a comonad to interpret the logic $\mathcal{L}_{com}$. The Coq implementation of both logics is given in Section 4.4. These logics have been built so as to be sound with respect to their intended categorical interpretation; but little is known about their completeness. Therefore, in Section 4.5, we conclude with a completeness notion: *relative Hilbert-Post completeness* which is well-suited to a decorated logic. We will show in Sections 5.4 and 6.9 that one *decorated logic for the state effect* and two *decorated logics for the exception effect* are Hilbert-Post complete with respect to their pure sublogics: we adapt the theorem in [Sta10, Th 5] to our logics to give a decorated proof of their completeness.

4.1 The monadic equational logic

The *monadic equational logic* ($\mathcal{L}_{meq}$) is interpreted in a category with objects as types, arrows as terms and equalities as equations. The reason we choose to start with such a logic is that it can be extended into a *decorated logic* with the use of decorations on terms and equations. Notice that the keyword "*monadic*" has little to do with monads. It indicates that the operations of the logic are *unary* (or mono-adic). We remind the monadic equational logic [Mog91] with its grammar and inference rules given in Figures 4.1 and 4.2:

Grammar for the monadic equational logic:

Types:	$\mathtt{t}$	$::=$	$\mathtt{X} \mid \mathtt{Y} \mid \dots$
Terms:	$\mathtt{f}, \mathtt{g}$	$::=$	$\mathtt{id_t} \mid \mathtt{a} \mid \mathtt{b} \mid \cdots \mid \mathtt{g} \circ \mathtt{f}$
Equations:	$\mathtt{e}$	$::=$	$\mathtt{f} \cong \mathtt{g}$

Figure 4.1: Syntax for $\mathcal{L}_{meq}$

Every term has a source and a target type, e.g., $f\colon X \to Y$. Every equation is formed by terms with the same source and target types, e.g., $e : f \cong g$ where $f,\ g\colon\ X \to\ Y$.

The logic $\mathcal{L}_{meq}$ can be interpreted in a category: each type as an object, each term as an arrow and each equation as an equality between arrows with the same source and target.

categorical rules

(id) $\dfrac{X}{id_X\colon X \to X}$ (comp) $\dfrac{f\colon X \to Y \quad g\colon Y \to Z}{(g \circ f)\colon X \to Z}$

(id-source) $\dfrac{f\colon X \to Y}{f \circ id_X \cong f}$ (id-target) $\dfrac{f\colon X \to Y}{id_Y \circ f \cong f}$

(assoc) $\dfrac{f\colon X \to Y \ \ g\colon Y \to Z \ \ h\colon Z \to U}{h \circ (g \circ f) \cong (h \circ g) \circ f}$

congruence rules

(refl) $\dfrac{f}{f \cong f}$ (sym) $\dfrac{f \cong g}{g \cong f}$ (trans) $\dfrac{f \cong g \ \ g \cong h}{f \cong h}$

(replsubs) $\dfrac{f_1 \cong f_2\colon X \to Y \ \ g_1 \cong g_2\colon Y \to Z}{g_1 \circ f_1 \cong g_2 \circ f_2}$

Figure 4.2: Inference rules for $\mathcal{L}_{meq}$

The *congruence rules* indicate that the relation '$\cong$' is a congruence which means, an equivalence relation (reflexive, symmetric and transitive) which obeys *replacement* and *substitution* of compatible terms with respect to the composition. The basic categorical rules indicate that there is an identity morphism $id_X\colon X \to X$ for each type X, that composition is an associative operation and that composing any term f with id is f, up to $\cong$, no matter the composition order.

4.2 The decorated logic for a monad

The decorated logic for a monad ($\mathcal{L}_{mon}$) [DDR14] extends the monadic equational logic ($\mathcal{L}_{meq}$) with the use of decorations on terms and equations. We give the syntax and the inference rules of $\mathcal{L}_{mon}$ in Figures 4.3 and 4.5, respectively.

Grammar for the decorated logic for a monad:

Types:	$\mathtt{t}$	::=	$\mathtt{X} \mid \mathtt{Y} \mid \ldots$
Terms:	$\mathtt{f}, \mathtt{g}$	::=	$\mathtt{id_t} \mid \mathtt{a} \mid \mathtt{b} \mid \cdots \mid \mathtt{g} \circ \mathtt{f}$
Decoration for terms:	$(\mathtt{d})$	::=	$(0) \mid (1) \mid (2)$
Equations:	$\mathtt{e}$	::=	$\mathtt{f} \equiv \mathtt{g} \mid \mathtt{f} \sim \mathtt{g}$

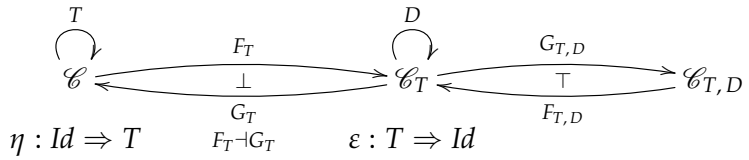
Figure 4.3: Syntax for $\mathcal{L}_{mon}$

Each term has a source and a target type as well as a decoration which is denoted as a superscript (0), (1) or (2): a *pure* term has the decoration (0), a *constructor* has (1) and a *modifier* term comes with the decoration (2). Similarly, each equation is formed by two terms with the same source and target as well as a decoration, denoted by "$\sim$" if it is *weak* or by "$\equiv$" if it is *strong*.

Remark 4.2.1. Note that within the scope of any decorated logic in this thesis, when stating the rules, the decorations are not explicitly given, if the rule in question is valid for all decorations. However, the decorations appear in the related interpretations.

Let (T, η, μ) be a monad satisfying the mono requirement which means that its unit η is a monomorphism. See Definition 3.1.5. In order to express the meaning (interpretation) of the logic $\mathcal{L}_{mon}$, we use the coKleisli-on-Kleisli construction associated to the monad (T, η, μ) as detailed in Section 3.2. There, we have introduced the adjunctions $F_T \dashv G_T$ and $F_{T,D} \dashv G_{T,D}$ with faithful functors $F_T \colon \mathscr{C} \to \mathscr{C}_T$ and $G_{T,D} \colon \mathscr{C}_T \to \mathscr{C}_{T,D}$. This gives raise to a hierarchy among morphisms in $\mathscr{C}_{T,D}$. This hierarchy is useful for interpreting decorations: *pure* terms are in $\mathscr{C}$, *constructors* are in $\mathscr{C}_T$ and *modifiers* are in $\mathscr{C}_{T,D}$.

Definition 4.2.2. Let $\mathbf{C_T}$ be the interpretation of the syntax for the logic $\mathcal{L}_{mon}$ with the following details:



(1) The types are interpreted as the objects of $\mathscr{C}$.

(2) The terms are interpreted as morphisms as follows:

(2.1) a *pure* term $f^{(0)} \colon X \to Y$ in $\mathscr{C}$ as $f \colon X \to Y$ in $\mathscr{C}$

(2.2) a *constructor* term $f^{(1)} \colon X \to Y$ in $\mathscr{C}_T$ as $f \colon X \to TY$ in $\mathscr{C}$

(2.3) a *modifier* term $f^{(2)} \colon X \to Y$ in $\mathscr{C}_{T,D}$ as $f \colon TX \to TY$ in $\mathscr{C}$

(3) A strong equation between modifiers $f^{(2)} \equiv g^{(2)} \colon X \to Y$ in $\mathscr{C}_{T,D}$ is interpreted by an equality $f = g \colon TX \to TY$ in $\mathscr{C}$. Similarly, a strong equation between constructors $f^{(1)} \equiv g^{(1)} \colon X \to Y$ in $\mathscr{C}_T$ is interpreted by an equality $f = g \colon X \to TY$ in $\mathscr{C}$. And a strong equation between pure terms $f^{(0)} \equiv g^{(0)} \colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g \colon X \to Y$ in $\mathscr{C}$.

(4) A weak equation between terms $f^{(2)} \sim g^{(2)} \colon X \to Y$ is interpreted by an equality $f \circ \eta_X = g \circ \eta_X \colon X \to TY$ in $\mathscr{C}$. Similarly, a weak equation between constructors

$f^{(1)} \sim g^{(1)} \colon X \to Y$ in $\mathscr{C}_T$ is interpreted by an equality $f = g \colon X \to TY$ in $\mathscr{C}$. And a weak equation between pure terms $f^{(0)} \sim g^{(0)} \colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g \colon X \to Y$ in $\mathscr{C}$.

$\mathcal{L}_{mon}$		Interpretation of $\mathcal{L}_{mon}$
modifier	$f^{(2)} : X \to Y$	$f : TX \to TY$
constructor	$f^{(1)} : X \to Y$	$f : X \to TY$
pure term	$f^{(0)} : X \to Y$	$f : X \to Y$
strong equation	$f^{(2)} \equiv g^{(2)} : X \to Y$	$f = g : TX \to TY$
weak equation	$f^{(2)} \sim g^{(2)} : X \to Y$	$f \circ \eta_X = g \circ \eta_X : X \to TY$

Figure 4.4: Summary of Definition 4.2.2

Example 4.2.3. Let $T = - + E$ be the *monad of exceptions* defined over the category of sets as in Section 3.2.3. We will use this specialization in Section 6.1 to interpret *the decorated logic for the exception ($\mathcal{L}_{exc}$)* which is an extension to $\mathcal{L}_{mon}$ used to formalize the exception effect.

In Figure 4.5, we propose an inference system associated to the syntax in Figure 4.3. The rules in question are obtained by decorating the rules in Figure 4.2. In addition, we introduce the hierarchies (or conversions) among decorations.

hierarchy rules

$$\frac{f^{(0)}}{f^{(1)}} \qquad \frac{f^{(1)}}{f^{(2)}}$$

$$\text{(stow)}\frac{f \equiv g}{f \sim g} \qquad \text{(wtos)}\frac{f^{(d)} \sim g^{(d')}}{f \equiv g} \text{ for all } d, d' \leq 1$$

congruence rules

$$\text{(refl)}\ \frac{f}{f \equiv f} \qquad \text{(sym)}\ \frac{f \equiv g}{g \equiv f}$$

$$\text{(trans)}\ \frac{f \equiv g \quad g \equiv h}{f \equiv h}$$

$$\text{(replsubs)}\ \frac{f_1 \equiv f_2 \colon X \to Y \quad g_1 \equiv g_2 \colon Y \to Z}{g_1 \circ f_1 \equiv g_2 \circ f_2}$$

$$\text{(wsym)}\ \frac{f \sim g}{g \sim f} \qquad \text{(wtrans)}\ \frac{f \sim g \quad g \sim h}{f \sim h}$$

$$\text{(wrepl)}\frac{f_1 \sim f_2 \colon X \to Y \quad g \colon Y \to Z}{g \circ f_1 \sim g \circ f_2}$$

$$\text{(pwsubs)}\frac{f^{(0)} \colon X \to Y \quad g_1 \sim g_2 \colon Y \to Z}{g_1 \circ f \sim g_2 \circ f}$$

categorical rules

$$\text{(id)}\ \frac{X}{id_X^{(0)} \colon X \to X} \qquad \text{(comp)}\ \frac{f^{(d)} \colon X \to Y \quad g^{(d')} \colon Y \to Z}{(g \circ f)^{(max(d,d'))} \colon X \to Z} \text{ for all } d, d'$$

$$\text{(ids)}\ \frac{f \colon X \to Y}{f \circ id_X \equiv f} \qquad \text{(idt)}\ \frac{f \colon X \to Y}{id_Y \circ f \equiv f}$$

$$\text{(assoc)}\ \frac{f \colon X \to Y \quad g \colon Y \to Z \quad h \colon Z \to U}{h \circ (g \circ f) \equiv (h \circ g) \circ f}$$

Figure 4.5: Inference rules for the logic $\mathcal{L}_{mon}$.

Proposition 4.2.4. *The logic $\mathcal{L}_{mon}$ is sound with respect to the interpretation $\mathbf{C_T}$ given in Definition 4.2.2. Moreover, the hierarchy rules are interpreted by faithful functors (informally conversions are "safe").*

Proof. (1) The conversion from *pure* terms to *constructors* is interpreted by the functor F_T. For each $f\colon X \to Y$ in $\mathscr{C}$, $F_T(f) = h^\flat$ in $\mathscr{C}_T$ where $h = \eta_Y \circ f\colon X \to TY$ in $\mathscr{C}$ (See Equation 3.10). We assume that η is a monomorphism. This implies, thanks to the point (i) of the dual of [ML71, Ch. IV, §3, Theorem 1], that F_T is faithful. Therefore, this conversion is safe.

(2) The conversion from *constructors* to *modifiers* is interpreted by the functor $G_{T,D}$. For each $f^\flat\colon X \to Y$ in $\mathscr{C}_T$, $G_{T,D}(f^\flat) = k^{\flat\sharp}$ in $\mathscr{C}_{T,D}$ where $k = \mu_Y \circ Tf\colon TX \to TY$ in $\mathscr{C}$ (See Equation 3.27). Due to Proposition 3.2.5, the functor $G_{T,D}$ is faithful. So that this conversion is safe.

(3) Now, the conversion from *pure* terms to *modifiers* is interpreted by the composition $G_{T,D} \circ F_T$. Thanks to Equation 3.29, we have $G_{T,D} \circ F_T(f) = (Tf)^{\flat\sharp} = k^{\flat\sharp}$ in $\mathscr{C}_{D,T}$, for each $f\colon X \to Y$ in $\mathscr{C}$ and some $k^{\flat\sharp}$ in $\mathscr{C}_{D,T}$ such that $k = Tf\colon TX \to TY$ in $\mathscr{C}$. The functors, $G_{T,D}$ and F_T are faithful so is $G_{T,D} \circ F_T$. Therefore, this conversion is safe.

(4) When a term has several decorations (due to being *pure* or *constructor*), it has different interpretations. I.e., $f^{(0)}\colon X \to Y$ can be interpreted either as $f\colon X \to Y$, $f\colon X \to TY$ or $f\colon TX \to TY$. Similarly, $f^{(1)}\colon X \to Y$ can be interpreted either as $f\colon X \to TY$ or as $f\colon TX \to TY$: the choice should be clear from the context. In any case, they will end up with the same result up to conversions. Therefore, the rules are given in such a way that terms are decorated with the largest possible decorations. Note also that when a term appears with no decoration, this means that it has the decoration (2).

(5) (stow) For each pair of mappings $f, g\colon TX \to TY$ in $\mathscr{C}$, if $f = g$ holds, then obviously $f \circ \eta_X = g \circ \eta_X\colon X \to TY$. Recalling the items (3) and (4), we say that the interpretation of a strong equation $f^{(2)} \equiv g^{(2)}$ implies the interpretation of weak equation $f^{(2)} \sim g^{(2)}$. So that the conversion from strong to weak is freely allowed.

(6) (wtos) Moreover, it is possible to infer from items (3) and (4) that the interpretation of a weak equation $f \sim g$ coincides with the one for $f \equiv g$ in case f and g are not modifier terms. So that $f^{(1)} \sim g^{(1)}$ can be converted into $f^{(1)} \equiv g^{(1)}$.

(7) (wrepl) Given $f_1^{(2)} \sim f_2^{(2)}\colon X \to Y$ with interpretation $f_1 \circ \eta_X = f_2 \circ \eta_X\colon X \to TY$ in $\mathscr{C}$ and $g^{(2)}\colon Y \to Z$ with interpretation $g\colon TY \to TZ$, we get $g^{(2)} \circ f_1^{(2)} \sim g^{(2)} \circ f_2^{(2)}$ in $\mathscr{C}_{T,D}$ with the following interpretation: $g \circ f_1 \circ \eta_X = g \circ f_2 \circ \eta_X\colon X \to TZ$ in $\mathscr{C}$. That informally means that weak equations obey the *replacement* rule with no precondition.

(8) (pwsubs) Given $g_1^{(2)} \sim g_2^{(2)}\colon Y \to Z$ with interpretation $g_1 \circ \eta_Y = g_2 \circ \eta_Y\colon Y \to TZ$ in $\mathscr{C}$ and $f^{(0)}\colon X \to Y$ which can be seen as $f^{(2)}\colon X \to Y$ and interpreted as $Tf\colon TX \to TY$ in $\mathscr{C}$, thanks to above point (3). We get $g_1^{(2)} \circ f^{(0)} \sim g_2^{(2)} \circ f^{(0)}$ with the following interpretation: $g_1 \circ Tf \circ \eta_X = g_2 \circ Tf \circ \eta_X\colon X \to TZ$ in $\mathscr{C}$. It is simple to check that this equality holds: due to the naturality of η, we get $g_1 \circ \eta_Y \circ f = g_1 \circ \eta_Y \circ f$ which is true considering the given interpretation $g_1 \circ \eta_Y = g_2 \circ \eta_Y$.

This informally means that weak equations obey the *substitution* rule only when the substituted term is pure.

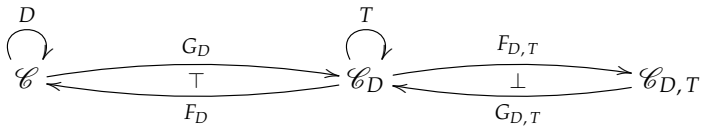
(9) The *identity* term $id_X^{(0)} : X \to X$ is interpreted as $id_X : X \to X$ in $\mathscr{C}$. The composition of two modifiers $f^{(2)} : X \to Y$ and $g^{(2)} : Y \to Z$ has the interpretation $g \circ f = TX \to TZ$ in $\mathscr{C}$. Given these, the interpretations for the rules (ids), (idt) and (assoc) can trivially be deduced. □

4.3 The decorated logic for a comonad

The decorated logic for a comonad ($\mathcal{L}_{com}$) [DDR14] extends the monadic equational logic $\mathcal{L}_{meq}$ with the use of decorations on terms and equations. The syntax of $\mathcal{L}_{com}$ is similar to that of $\mathcal{L}_{mon}$ and given in Figure 4.3. Each term has a source and a target type as well as a decoration which is denoted as a superscript (0), (1) or (2): a *pure* term has the decoration (0), an *observer* has (1) and a *modifier* term comes with the decoration (2). Similarly, each equation is formed by two terms with the same source and target as well as a decoration, denoted by "$\sim$" if it is *weak* or by "$\equiv$" if it is *strong*.

The logic $\mathcal{L}_{com}$ is dually interpreted with $\mathcal{L}_{mon}$. Let (D, ε, δ) be a comonad satisfying the epi requirement which means that its counit ε is an epimorphism. See Definition 3.1.7. In order to interpret $\mathcal{L}_{com}$, we this time, use the Kleisli-on-coKleisli construction associated to the comonad (D, ε, δ) as detailed in Section 3.3. There, we have introduced the adjunctions $F_D \dashv G_D$ and $F_{D,T} \dashv G_{D,T}$ with the faithful functors $G_D : \mathscr{C} \to \mathscr{C}_D$ and $F_{D,T} : \mathscr{C}_D \to \mathscr{C}_{D,T}$. This gives raise to a hierarchy among morphisms in $\mathscr{C}_{D,T}$. We use this hierarchy to interpret the decorations: *pure* terms are in $\mathscr{C}$, *observers* are in $\mathscr{C}_D$ and *modifiers* are in $\mathscr{C}_{D,T}$.

Definition 4.3.1. Let $\mathbf{C_D}$ be interpretation of the syntax for the logic $\mathcal{L}_{com}$ with following details:



$\varepsilon : D \Rightarrow Id \qquad G_D \vdash F_D \qquad \eta : Id \Rightarrow T$

(1) The types are interpreted as the objects of $\mathscr{C}$.

(2) The terms are interpreted as the morphisms as follows:

(2.1) a *pure* term $f^{(0)} : X \to Y$ in $\mathscr{C}$ as $f : X \to Y$ in $\mathscr{C}$

(2.2) an *observer* term $f^{(1)} : X \to Y$ in $\mathscr{C}_D$ as $f : DX \to Y$ in $\mathscr{C}$

(2.3) a *modifier* term $f^{(2)} : X \to Y$ in $\mathscr{C}_{D,T}$ as $f : DX \to DY$ in $\mathscr{C}$

(3) A strong equation between modifiers $f^{(2)} \equiv g^{(2)} : X \to Y$ in $\mathscr{C}_{D,T}$ is interpreted by an equality $f = g : DX \to DY$ in $\mathscr{C}$. Similarly, a strong equation between accessors $f^{(1)} \equiv g^{(1)} : X \to Y$ in $\mathscr{C}_D$ is interpreted by an equality $f = g : DX \to Y$ in $\mathscr{C}$. And a strong equation between pure terms $f^{(0)} \equiv g^{(0)} : X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g : X \to Y$ in $\mathscr{C}$.

(4) A weak equation between modifiers $f^{(2)} \sim g^{(2)} \colon X \to Y$ in $\mathscr{C}_{D,T}$ is interpreted by an equality $\varepsilon_Y \circ f = \varepsilon_Y \circ g \colon DX \to Y$ in $\mathscr{C}$. Similarly, a weak equation between accessors $f^{(1)} \sim g^{(1)} \colon X \to Y$ in $\mathscr{C}_D$ is interpreted by an equality $f = g \colon DX \to Y$ in $\mathscr{C}$. And a weak equation between pure terms $f^{(0)} \sim g^{(0)} \colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g \colon X \to Y$ in $\mathscr{C}$.

$\mathcal{L}_{mon}$		Interpretation of $\mathcal{L}_{mon}$
modifier	$f^{(2)} : X \to Y$	$f : DX \to DY$
observer	$f^{(1)} : X \to Y$	$f : X \to DY$
pure term	$f^{(0)} : X \to Y$	$f : X \to Y$
strong equation	$f^{(2)} \equiv g^{(2)} : X \to Y$	$f = g : DX \to DY$
weak equation	$f^{(2)} \sim g^{(2)} : X \to Y$	$\varepsilon_Y \circ f = \varepsilon_Y \circ g : DX \to Y$

Figure 4.6: Summary of Definition 4.3.1

Example 4.3.2. Let $D = - \times S$ be the *comonad of states* defined over the category of set where S is the distinguished set of states and '$\times$' is the cartesian product operator. We will use this specialization in Section 5.1 to interpret *the decorated logic for the state* ($\mathcal{L}_{st}$) which is an extension to $\mathcal{L}_{com}$ used to formalize the state effect.

In Figure 4.7, we propose an inference system associated to the syntax given in Figure 4.3.

hierarchy rules : See Figure 4.5

congruence rules: the single difference only - see Figure 4.5 for the rest

$$\text{(pwrepl)}\frac{f^{(0)} \colon Y \to Z \quad g_1 \sim g_2 \colon X \to Y}{f \circ g_1 \sim f \circ g_2} \qquad \text{(wsubs)}\frac{f_1 \sim f_2 \colon Y \to Z \quad g \colon X \to Y}{f_1 \circ g \sim f_2 \circ g}$$

categorical rules : See Figure 4.5

Figure 4.7: Inference rules for the logic $\mathcal{L}_{com}$.

Proposition 4.3.3. *The logic $\mathcal{L}_{com}$ is sound with respect to the interpretation $\mathbf{C_D}$ given in Definition 4.3.1. Moreover, the hierarchy rules are interpreted by faithful functors (informally conversions are "safe").*

Proof. (1) The conversion from *pure* terms to *observers* is interpreted by the functor G_D. For each $f \colon X \to Y$, $G_D(f) = h^\sharp$ in $\mathscr{C}_D$ where $h = f \circ \varepsilon_X \colon DX \to Y$ in $\mathscr{C}$ (See Equation 3.19). We assume that ε is an epimorphism. This implies, thanks to the point (i) of [ML71, Ch. IV, §3, Theorem 1], that G_D is faithful. Therefore, this conversion is safe.

(2) The conversion from *observers* to *modifiers* is interpreted by a the functor $F_{D,T}$. For each $f^\sharp \colon X \to Y$, $F_{D,T}(f^\sharp) = k^{\sharp\flat}$ in $\mathscr{C}_{D,T}$ where $k = Df \circ \delta_X \colon DX \to DY$ in $\mathscr{C}$ (See Equation 3.35). Due to Proposition 3.3.4, $F_{T,D}$ is faithful. So that this conversion is safe.

(3) Now, the conversion from *pure* terms to *modifiers* is interpreted by the composition $F_{D,T} \circ G_D$. Thanks to Equation 3.37, we have $F_{D,T} \circ G_D(f) = (Df)^{\sharp\flat} =$

$k^{\sharp\flat}$ in $\mathscr{C}_{D,T}$, for each $f\colon X \to Y$ in $\mathscr{C}$ and some $k^{\sharp\flat}$ in $\mathscr{C}_{D,T}$ such that $k = Df\colon DX \to DY$ in $\mathscr{C}$. The functors $F_{D,T}$ and G_D are faithful, so is $F_{D,T} \circ G_D$. Therefore, this conversion is safe.

(4) When a term has several decorations (due to being *pure* or *observer*) then it has different interpretations. I.e., $f^{(0)}\colon X \to Y$ can be interpreted either as $f\colon X \to Y$, $f\colon DX \to Y$ or $f\colon DX \to DY$. Similarly, $f^{(1)}\colon X \to Y$ can be interpreted either as $f\colon DX \to Y$ or as $f\colon DX \to DY$: the choice should be clear from the context. In any case, they will end up with the same result up to conversions. Therefore, the rules are given in such a way that terms are decorated with the largest possible decorations. Similar to the decorated logic for a monad, when a term appears with no decoration, this means that it has the decoration (2).

(5) For each pair of mappings $f, g\colon DX \to DY$ in $\mathscr{C}$, if $f = g$ holds, then obviously $\varepsilon_Y \circ f = \varepsilon_Y \circ g\colon DX \to Y$. Recalling the items (3) and (4), we say that the interpretation of a strong equation $f^{(2)} \equiv g^{(2)}$ implies the interpretation of weak equation $f^{(2)} \sim g^{(2)}$. So that the conversion from strong to weak is freely allowed.

(6) Moreover, it is possible to infer from items (3) and (4) that the interpretation of a weak equation $f \sim g$ coincides with the one for $f \equiv g$ in case f and g are not modifier terms. So that $f^{(1)} \sim g^{(1)}$ can be converted into $f^{(1)} \equiv g^{(1)}$.

(7) (wsubs) Given $f_1^{(2)} \sim f_2^{(2)}\colon Y \to Z$ with interpretation $\varepsilon_Z \circ f_1 = \varepsilon_Z \circ f_2\colon DY \to Z$ in $\mathscr{C}$ and $g^{(2)}\colon X \to Y$ with interpretation $g\colon DX \to DY$ in $\mathscr{C}$, we get $f_1^{(2)} \circ g^{(2)} \sim f_2^{(2)} \circ g^{(2)}$ in $\mathscr{C}_{T,D}$ with the following interpretation: $\varepsilon_Z \circ f_1 \circ g = \varepsilon_Z \circ f_2 \circ g\colon DX \to Z$ in $\mathscr{C}$. That informally means that weak equations obey the *substitution* rule with no precondition.

(8) (pwrepl) Given $g_1^{(2)} \sim g_2^{(2)}\colon X \to Y$ with interpretation $\varepsilon_Y \circ g_1 = \varepsilon_Y \circ g_2\colon DX \to Y$ in $\mathscr{C}$ and $f^{(0)}\colon Y \to Z$ which can be seen as $f^{(2)}\colon Y \to Z$ and interpreted as $Df\colon DY \to DZ$ in $\mathscr{C}$, thanks to above point (3). We get $f^{(0)} \circ g_1^{(2)} \sim f^{(0)} \circ g_2^{(2)}$ with the following interpretation: $\varepsilon_Z \circ Df \circ g_1 = \varepsilon_Z \circ Df \circ g_2\colon DX \to Z$ in $\mathscr{C}$. It is simple to check that this equality holds: due to the naturality of η, we get $f \circ \varepsilon_Y \circ g_1 = f \circ \varepsilon_Y \circ g_2$ which is true considering the given interpretation $\varepsilon_Y \circ g_1 = \varepsilon_Y \circ g_2$. This informally means that weak equations obey the *replacement* rule only when the replaced term is pure.

(9) The *identity* term $id_X^{(0)}\colon X \to X$ is interpreted as $id_X\colon X \to X$ in $\mathscr{C}$. The composition of modifiers $f^{(2)}\colon X \to Y$ and $g^{(2)}\colon Y \to Z$ has the interpretation $g \circ f = DX \to DZ$ in $\mathscr{C}$. Given these, the interpretations of the rules (ids), (idt) and (assoc) can trivially be deduced. □

4.4 Decorated logic in Coq

The decorated logics for a monad and comonad are implemented as separate frameworks in the Coq Proof Assistant. In order to construct these frameworks, we need to define data structures, terms, decorations and basic rules as axioms. This organization is reflected with corresponding Coq modules as follows:

BASES:	Terms ⟶ Decorations ⟶ Axioms

Remark 4.4.1. This organization will be extended into Coq libraries, when we formalize the decorated logics for the state in Section 5.2 and for the exception in Section 6.4.

First, we give the definitions of *non-decorated terms*: they constitute the main part of the design with the introduction of the basic operations. The next step is to decorate these functions. For instance, the `id` function is defined as *pure* and this status is represented by a *pure* label in the library. All the rules related to decorated functions are stated in the module called *Axioms*. Considering the entire design, we benefit from an important aspect provided by Coq environment, namely *dependent types*. They provide a unified formalism in the creation of new data types and allow us to deal in a simple manner with most of the typing issues. More precisely, the new Coq `Type` `term` defined in Section 4.4.1 is not a `Type`, but rather a `Type` → `Type` → `Type`. The domain/codomain information of `term` is embedded into the Coq type system, so that we do not need to talk about ill-typed `terms` and their compositions. For instance, given `X Y: Type`, we have `term Y X` which is also a `Type` instance representing the type of terms with domain X and codomain Y, in $\mathcal{L}_{dec}$: the reason for this exchange is just to get an ease in term compositions. Now, let us go through the Coq implementation details of the logic $\mathcal{L}_{com}$. Along the way, the differences with the logic $\mathcal{L}_{mon}$ will be pinpointed.

4.4.1 Terms

We define the terms of the decorated logic for a monad $\mathcal{L}_{com}$ by using an *inductive predicate* called `term`. It mainly establishes a new Coq `Type` out of two input `Types`.

```
Inductive term: Type → Type → Type :=
 | tpure:  forall {X Y: Type}, (X → Y) → term Y X
 | comp:   forall {X Y Z: Type}, term X Y → term Y Z → term X Z.
Infix "o" := comp (at level 70).
```

The type `term Y X` is dependent. It depends on the `Type` instances `X` and `Y` and represents the arrow type: `X` → `Y` in the decorated framework. The constructor `tpure` takes a Coq side (pure) function and translates it into the decorated environment. So that pure terms, as `id`, such are built by applying the `tpure` constructor to Coq functions. The `comp` constructor deals with the composition of two compatible terms. I.e., given a pair of terms `f` : `term X Y` and `g` : `term Y Z`, then the composition `f` ∘ `g` would be an instance of the type `term X Z`. For the sake of conciseness, infix 'o' is used to denote the term composition. Notice that together with the associativity of composition (see Section 4.4.3), this defines a category with objects as Coq `Types` and morphisms as `tpure f: term Y X`, for each pure Coq function `f` : `X` → `Y`.

```
Definition id {X: Type} : term X X := tpure id.
```

Since the *identity function* is natively embedded in Coq, we use `tpure` constructor to have it within the decorated scope. We have an abuse of notation here: the `id` applied to the `tpure` constructor is the one already involved in the Coq system (aka `Datatypes.id`) while the `id` we define is an instance of type `term X X`, representing the type of mappings from `X` to `X`. In such a setting, we also have *constant terms*:

```
Definition constant {X: Type} (v: X): term X unit := tpure (fun tt ⇒ v).
```

Any Coq side pure function of type $\mathbb{1} \to$ `X` for each `Type` instance `X`, is translated into the decorated settings via the `tpure` constructor: `fun tt ⇒ v` corresponds to the lambda term `~tt : unit.v` where `unit` is the singleton type and `tt` is the unique instance of it. Therefore, for any constant value `v: X`, the pure term `constant v` is of type `term X unit`.

4.4.2 Decorations

The decorations are first enumerated and then assigned to the terms by using an *inductive predicate* named `is`. It forms a proposition (a `Prop` instance in Coq) out of a `term` and a `kind` which is the name of the enumerated `Type` for decorations. Within the context of a decorated proof, `is` is used to check whether the given term is properly decorated or not. We respectively use keywords `pure`, `ro` and `rw` instead of (0), (1) and (2): `pure` for *pure*, `ro` for *observer* and `rw` for *modifier* terms. Below, is the association of decorations on terms:

```
Inductive kind := pure | ro  | rw.
Inductive is : kind → forall X Y, term X Y → Prop :=
 | is_tpure  : forall X Y (f: X → Y), is pure (@tpure X Y f)
 | is_comp   : forall k X Y Z (f: term X Y) (g: term Y Z), is k f → is k g → is k (f ∘ g)
 | is_pure_ro : forall X Y (f: term X Y), is pure f → is ro f
 | is_ro_rw  : forall X Y (f: term X Y), is ro f → is rw f.
```

Any term constructed through the `tpure` constructor is pure. The decoration of composed terms depends on the decorations of the components: indeed, it takes the larger decoration. The hierarchy rules among terms are also given here: the constructor `is_pure_ro` ensures that a pure term can be seen as an observer and similarly `is_ro_rw` is to indicate that an observer term can be taken as a modifier, on demand. See the non-equational hierarchy and categorical rules in Figure 4.5. It is trivial to infer now that `id` is a pure term:

```
Lemma is_id X: is pure (@id X).
Proof. unfold id. apply is_tpure. Qed.
```

After unfolding `id`, one needs to show that `is pure (tpure Datatypes.id)` holds. Now, it suffices to apply the constructor `is_tpure` to close the goal. It would also be sufficient to skip the unfolding: directly applying the constructor `is_tpure`. The tactic `unfold` provides information on how the term in question has been defined. To stay pedagogical (for these example), we prefer using unfold.

1 subgoals X : Type _____(1/1) is pure id	unfold id.
1 subgoals X : Type _____(1/1) is pure (tpure id)	apply is_tpure.

4.4.3 Axioms: decorated logic for a comonad

We can now detail the Coq implementation of the axioms. The idea here is to decorate also the equations. On the one hand, the weak equation between parallel morphisms models the fact that they have the same result but may perform differently with respect to an associated effect. On the other hand, if both have result and effect equivalence, then the equation becomes strong. We hereby state the rules with respect to weak and strong equations by defining them in a mutually inductive way: mutuality is used here

to enable the rules including strong and weak equations at the same time. In Coq, both strong and weak equations are defined to be the instances of the `relation` class. We respectively use the symbols '==' and '∼' to denote `strong` and `weak` equations within Coq. See the equational rules in Figure 4.7.

```
Reserved Notation "x == y" (at level 80). Reserved Notation "x ∼ y" (at level 80).
Definition idem X Y (x y: term X Y) := x = y.
Inductive strong: forall X Y, relation (term X Y) :=
    (*congruence rules*)
 | refl X Y: Reflexive (@strong X Y)
 | sym: forall X Y, Symmetric (@strong X Y)
 | trans: forall X Y, Transitive (@strong X Y)
 | replsubs: forall X Y Z, Proper (@strong X Y ==> @strong Y Z ==> @strong X Z) comp
    (*categorical rules*)
 | ids: forall X Y (f: term X Y), f o id == f
 | idt: forall X Y (f: term X Y), id o f == f
 | assoc: forall X Y Z T (f: term X Y) (g: term Y Z) (h: term Z T), f o (g o h) == (f o g) o h
    (*the hierarchy rule*)
 | wtos: forall X Y (f g: term X Y), is ro f → is ro g → f ∼ g → f == g
   (*tpure preserves the pure composition*)
 | tcomp: forall X Y Z (f: Z → Y) (g: Y → X), tpure (compose g f) == tpure g o tpure f
with weak: forall X Y, relation (term X Y) :=
    (*congruence rules*)
 | wsym: forall X Y, Symmetric (@weak X Y)
 | wtrans: forall X Y, Transitive (@weak X Y)
 | pwrepl: forall X Y Z (g: term X Y), is pure g → Proper (@weak Y Z ==> @weak X Z) (comp g)
 | wsubs: forall X Y Z, Proper (@weak X Y ==> @idem Y Z ==> @weak X Z) comp
    (*the hierarchy rule*)
 | stow: forall X Y (f g: term X Y), f == g → f ∼ g
where "x == y" := (strong x y) and "x ∼ y" := (weak x y).
```

(1) More precisely, strong equation is an equivalence relation so that *reflexivity*, *symmetry* and *transitivity* properties are assumed: it is defined to be an instance of the `Reflexive`, `Symmetric` and `Transitive` relation classes of Coq.

(2) The *replacement* and *substitution* properties with respect to strong equation are assumed by stating the composition as a proper element of the relation (`@strong X Y ==> @strong Y Z ==> @strong X Z`) for each `X Y Z: Type`. Due to Sozeau [Soz10], '==>' is the right-associative notation used to indicate the `respectfulness` property among relations. I.e., `respectful (R: (@strong Y Z)) (R': (@strong X Z))` returns an instance R'' of type `relation (term Y Z -> term X Z)`. So that composition is now a proper instance of type `relation (term X Y -> term Y Z -> term X Z)` with respect to the strong equation.

(2.1) Let us now suppose that `f: (term X Y)` and $g == g'$`: (term Y Z)` are given and we intend to show that $f \circ g == f \circ g'$ holds, for each `X Y Z: Type`. It is trivial to reduce it to $f \circ g == f \circ g$ through the use of the tactic `setoid_rewrite`, developed by Coen [Coe04], since *replacement* with respect to strong equation has already been enabled with no precondition.

1 subgoals X : Type Y : Type Z : Type f : term X Y g : term Y Z g′ : term Y Z H0 : g == g′ ----------(1/1) f o g == f o g′	setoid_rewrite ←H0.
1 subgoals X : Type Y : Type Z : Type f : term X Y g : term Y Z g′ : term Y Z H0 : g == g′ ----------(1/1) f o g == f o g	apply refl.

2.2 Similarly, let us suppose that `f` $\equiv$ `f'`: `(term X Y)` and `g`: `(term Y Z)` are given and we intend to show that `f` $\circ$ `g` $\equiv$ `f'` $\circ$ `g` holds. One can trivially reduce it to `f` $\circ$ `g` $\equiv$ `f` $\circ$ `g`, since strong *substitution* comes with no precondition.

1 subgoals X : Type Y : Type Z : Type f : term X Y f' : term X Y g : term Y Z H0 : f == f' ----------(1/1) f' o g == f o g	setoid_rewrite ←H0.
1 subgoals X : Type Y : Type Z : Type f : term X Y f' : term X Y g : term Y Z H0 : f == f' ----------(1/1) f o g == f o g	apply refl.

(3) By `ids` and `idt`, we get that the composition of `id` with any term `f` is strongly equal to `f` no matter the composition order.

(4) The property *associativity* is attached to the composition.

(5) Converting any instance of weak equation into strong is not pricelessly ensured by `wtos`: one has to make sure that both hand sides are at most *observers* or `ro` in Coq implementation.

(6) The rule `tcomp` states that the `tpure` constructor preserves the composition of pure terms up to the strong equation.

(7) Weak equality is also an equivalence relation: it is assumed to be an instance of `Symmetric` and `Transitive` relation classes. It is trivial to prove that weak equation is an instance of `Reflexive` class:

```
Instance wrefl X Y: Reflexive (@weak X Y).
Proof. intros X Y f. apply stow. apply refl. Qed.
```

First, the goal-side weak equation `f` $\sim$ `f` is reduced to `f` $==$ `f` via the rule (stow). Then, it suffices to apply the reflexivity property of strong equation (refl) to close the goal.

1 subgoals X : Type Y : Type f : term Y X ----------(1/1) f ∼ f	apply stow
1 subgoals X : Type Y : Type f : term Y X ----------(1/1) f == f	apply refl.

(8) The *replacement* with respect to weak equation is enabled only when the replaced term is pure. Thus, `(comp g)` is an instance of the type `(@weak Y Z ==> @weak X Z)` for each `X Y Z: Type` and pure term `g: term X Y`. Suppose that we are given `f` $\sim$ `f'`: `(term Y Z)` and $g^{(0)}$: `(term X Y)` so to show that `g` $\circ$ `f` $\sim$ `g` $\circ$ `f'` holds. Since `g` is pure, one simply handles `g` $\circ$ `f` $\sim$ `g` $\circ$ `f`.

1 subgoals X : Type Y : Type Z : Type f : term Y Z f′ : term Y Z g : term X Y H0: is pure g H1 : f ∼ f′ ----------(1/1) g ∘ f′ ∼ g ∘ f	setoid_rewrite ←H1.
1 subgoals X : Type Y : Type Z : Type f : term Y Z f′ : term Y Z g : term X Y H0: is pure g H1 : f ∼ f′ ----------(1/1) g ∘ f ∼ g ∘ f	apply wrefl.

(9) The *substitution* with respect to weak equation is given by assuming that composition is a proper instance of the relation (`@weak X Y ==> @idem Y Z ==> @weak X Z`) for each `X Y Z: Type` where `idem` takes two instances say `x` and `y` of the type `term X Y` and checks whether `x` equals to `y`. Let us suppose that `f ∼ f′: (term X Y)` and `g: (term Y Z)` are given and we intend to show that `f ∘ g ∼ f′ ∘ g` holds, for each `X Y Z: Type`. It is trivial to obtain `f ∘ g ∼ f ∘ g`, since *weak substitution* can be done with no precondition.

1 subgoals X : Type Y : Type Z : Type f : term X Y f′ : term X Y g : term Y Z H0 : f ∼ f′ ----------(1/1) f′ ∘ g ∼ f ∘ g	setoid_rewrite ←H0.
1 subgoals X : Type Y : Type Z : Type f : term X Y f′ : term X Y g : term Y Z H0 : f ∼ f′ ----------(1/1) f ∘ g ∼ f ∘ g	apply wrefl.

(10) Lastly, strong equation freely converts into weak equation via `stow`.

4.4.4 Axioms: decorated logic for a monad

In this section, we consider the implementation of decorated logic for a monad $\mathcal{L}_{mon}$. This follows the same approach for terms and decorations with the implementation of the logic $\mathcal{L}_{com}$. The single difference appears within the context of rules: dual to the implementation of the logic $\mathcal{L}_{com}$, weak *substitution* is enabled only when the substituted term is pure while weak *replacement* comes with no precondition. See congruence rules in Figure 4.7.

```
Definition pure_id X Y (x y: term X Y) := (is pure x) ∧ x = y.
Definition idem X Y (x y: term X Y) := x = y.

Inductive strong: forall X Y, relation (term X Y) :=
      ...
with weak: forall X Y, relation (term X Y) :=
      ...
  | wrepl : forall X Y Z, Proper (@idem Z Y ==> @weak Y X ==> @weak Z X) comp
  | pwsubs : forall X Y Z, Proper (@weak Z Y ==> @pure_id Y X ==> @weak Z X) comp
where "x == y" := (strong x y) and "x ∼ y" := (weak x y).
```

(1) The *replacement* with respect to weak equation is given by assuming that composition is a proper instance of the relation (`@idem Z Y ==> @weak Y X ==> @weak Z X`), for each `X Y Z: Type` where `idem` takes two instances say `x` and `y` of the type `term X Y` and checks whether `x` equals to `y`. Let us suppose that `f ∼ f′: (term`

Y X) and g: (term Z Y) are given and we intend to show that g ∘ f ∼ g ∘ f′ holds, for each X Y Z: Type. It is trivial to obtain g ∘ f ∼ g ∘ f, since *weak replacement* can be done with no precondition.

1 subgoals X : Type Y : Type Z : Type f : term Y X f′ : term Y X g : term Z Y H0 : f ∼ f′ _____(1/1) g ∘ f ∼ g ∘ f′	setoid_rewrite ←H0.
1 subgoals X : Type Y : Type Z : Type f : term Y X f′ : term Y X g : term Y Z H0 : f ∼ f′ _____(1/1) g ∘ f ∼ g ∘ f	apply wrefl.

(2) The *substitution* with respect to weak equation is enabled only when the substituted term is pure. Thus, (comp) is an instance of the relation (@weak Z Y ==> @pure_id Y X ==> @weak Z X), for each X Y Z: Type where pure_id takes two instances say x and y of the type term X Y and checks whether x is pure and equals to y. Suppose that we are given f ∼ f′: (term Z Y) and $g^{(0)}$: (term Y X) so as to show that f ∘ g ∼ f′ ∘ g holds. Since g is pure, one simply handles that f ∘ g ∼ f ∘ g as:

1 subgoals X : Type Y : Type Z : Type f : term Z Y f′ : term Z Y g : term Y X H0 : f ∼ f′ H1 : is pure g _____(1/1) f ∘ g ∼ f′ ∘ g	apply pwsubs.	
2 subgoal X : Type Y : Type Z : Type f : term Z Y f′ : term Z Y g : term Y X H0 : f ∼ f′ H1 : is pure g _____(1/2) f ∼ f′ _____(2/2) pure_id g g	exact H0. split.	
2 subgoal X : Type Y : Type Z : Type f : term Z Y f′ : term Z Y g : term Y X H0 : f ∼ f′ H1 : is pure g _____(1/2) is pure g _____(2/2) g = g		exact H1. reflexivity.

Applying pwsubs (See Figure 4.5) results in two subgoals: f ∼ f' and pure_id g g. The former is already an assumption so that we remain with the latter which can be split into two further subgoals when unfolded: is pure g and g = g. Now, the former is an assumption and the latter is trivial given that the relation '=' is reflexive.

Let us conclude with the notion called *Hilbert-Post Completeness* which is well-suited with a decorated theory. We will make use of this notion to show in Section 5.4 and Section 6.9 that the base languages (with no use of categorical pairs and copairs) of the *decorated logic for the state* (which is an extension to $\mathcal{L}_{com}$) and the *decorated logic for the exception* (extending $\mathcal{L}_{mon}$) with the *programmers' language for exceptions* are Hilbert-Post complete.

4.5 Hilbert-Post completeness

Each logic in this thesis comes with a *language*, which is a set of *formulae* (equations), and with *deduction rules*. Deduction rules are used for deriving (or generating) *theorems*, which are some formulae, from some chosen formulae called *axioms*. A *theory* $\mathcal{T}$ is a set of theorems which is *deductively closed*, in the sense that every theorem which can be

derived from $\mathcal{T}$ using the rules of the logic is already in $\mathcal{T}$. We describe a categorical *intended model* for each logic we introduce; the rules of the logic are designed so as to be *sound* with respect to this intended model. Given a logic $\mathcal{L}$, the theories of $\mathcal{L}$ are partially ordered by inclusion. There is a maximal theory $\mathcal{T}_{max}$, where all formulae are theorems. There is a minimal theory $\mathcal{T}_{min}$, which is generated by the empty set of axioms. For all theories $\mathcal{T}$ and $\mathcal{T}'$, we denote by $\mathcal{T}+\mathcal{T}'$ the theory generated from $\mathcal{T}$ and $\mathcal{T}'$.

Example 4.5.1. With this point of view there are many different *equational logics*, with the same deduction rules but with different languages, depending on the definition of *terms*. In an equational logic, formulae are *pairs of parallel terms* $(f,g) : X \to Y$ and theorems are *equations* $f \equiv g : X \to Y$. Typically, the language of an equational logic may be defined from a *signature* (made of sorts and operations). The deduction rules are such that the equations in a theory form a *congruence*, i.e., an equivalence relation compatible with the structure of the terms. For instance, we may consider the logic "of naturals" $\mathcal{L}_{nat}$, with its language generated from the signature made of a sort N, a constant $0 : \mathbb{1} \to N$ and an operation $s : N \to N$. For this logic, the minimal theory is the theory "of naturals" $\mathcal{T}_{nat}$, the maximal theory is such that $s^k \equiv s^\ell$ and $s^k \circ 0 \equiv s^\ell \circ 0$ for all natural numbers k and ℓ, and (for instance) the theory "of naturals modulo 6" $\mathcal{T}_{mod6}$ can be generated from the equation $s^6 \equiv id_N$. We consider models of equational logics in sets: each type X is interpreted as a set (still denoted X), which is a singleton when X is $\mathbb{1}$, each term $f : X \to Y$ as a function from X to Y (still denoted $f : X \to Y$), and each equation as an equality of functions.

Definition 4.5.2. Given a logic $\mathcal{L}$ and its maximal theory $\mathcal{T}_{max}$, a theory $\mathcal{T}$ is *consistent* if $\mathcal{T} \neq \mathcal{T}_{max}$, and it is *Hilbert-Post complete* if it is consistent and if any theory which contains $\mathcal{T}$ coincides with $\mathcal{T}_{max}$ or with $\mathcal{T}$.

Example 4.5.3. In Example 4.5.1 we considered two theories for the logic $\mathcal{L}_{nat}$: the theory "of naturals" $\mathcal{T}_{nat}$ and the theory "of naturals modulo 6" $\mathcal{T}_{mod6}$. Since both are consistent and $\mathcal{T}_{mod6}$ contains $\mathcal{T}_{nat}$, the theory $\mathcal{T}_{nat}$ is not Hilbert-Post complete. The unique Hilbert-Post complete theory for $\mathcal{L}_{nat}$ is made of all equations but $s \equiv id_N$, it can be generated from the axioms $s \circ 0 \equiv 0$ and $s \circ s \equiv s$.

If a logic $\mathcal{L}$ is an extension of a sublogic $\mathcal{L}_0$, each theory $\mathcal{T}_0$ of $\mathcal{L}_0$ generates a theory $F(\mathcal{T}_0)$ of $\mathcal{L}$. Conversely, each theory $\mathcal{T}$ of $\mathcal{L}$ determines a theory $G(\mathcal{T})$ of $\mathcal{L}_0$, made of the theorems of $\mathcal{T}$ which are formulae of $\mathcal{L}_0$, so that $G(\mathcal{T}_{max}) = \mathcal{T}_{max,0}$. The functions F and G are monotone and they form a *Galois connection* [Smi10, Definition 2.1.1], denoted $F \dashv G$: for each theory $\mathcal{T}$ of $\mathcal{L}$ and each theory $\mathcal{T}_0$ of $\mathcal{L}_0$ we have $F(\mathcal{T}_0) \subseteq \mathcal{T}$ if and only if $\mathcal{T}_0 \subseteq G(\mathcal{T})$. It follows that $\mathcal{T}_0 \subseteq G(F(\mathcal{T}_0))$ and $F(G(\mathcal{T})) \subseteq \mathcal{T}$.

Definition 4.5.4. (Duval et al., [DDE+15]) Given a logic $\mathcal{L}_0$, an extension $\mathcal{L}$ of $\mathcal{L}_0$ and the associated Galois connection $F \dashv G$, a theory $\mathcal{T}'$ of $\mathcal{L}$ is $\mathcal{L}_0$-*derivable* from a theory $\mathcal{T}$ of $\mathcal{L}$ if $\mathcal{T}' = \mathcal{T} + F(\mathcal{T}'_0)$ for some theory $\mathcal{T}'_0$ of $\mathcal{L}_0$. And a theory $\mathcal{T}$ is *relatively Hilbert-Post complete with respect to* $\mathcal{L}_0$ if it is consistent and if any theory of $\mathcal{L}$ which contains $\mathcal{T}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$.

Each theory $\mathcal{T}$ is $\mathcal{L}_0$-derivable from itself, because $\mathcal{T} = \mathcal{T} + F(\mathcal{T}_{min,0})$, where $\mathcal{T}_{min,0}$ is the minimal theory of $\mathcal{L}_0$. In addition, Theorem 4.5.6 shows that relative completeness lifts the usual "absolute" completeness from $\mathcal{L}_0$ to $\mathcal{L}$.

Lemma 4.5.5. *Let us consider a logic $\mathcal{L}_0$, an extension $\mathcal{L}$ of $\mathcal{L}_0$ and the associated Galois connection $F \dashv G$. For each theory $\mathcal{T}$ of $\mathcal{L}$, a theory $\mathcal{T}'$ of $\mathcal{L}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$ if and only if $\mathcal{T}' = \mathcal{T} + F(G(\mathcal{T}'))$. As a special case, $\mathcal{T}_{max}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$ if and only if $\mathcal{T}_{max} = \mathcal{T} + F(\mathcal{T}_{max,0})$. A theory $\mathcal{T}$ of $\mathcal{L}$ is relatively Hilbert-Post complete with respect*

to $\mathcal{L}_0$ if and only if it is consistent and every theory $\mathcal{T}'$ of $\mathcal{L}$ which contains $\mathcal{T}$ is such that $\mathcal{T}' = \mathcal{T} + F(G(\mathcal{T}'))$.

Proof. Clearly, if $\mathcal{T}' = \mathcal{T} + F(G(\mathcal{T}'))$ then $\mathcal{T}'$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$. Conversely, let $\mathcal{T}'_0$ be a theory of $\mathcal{L}_0$ such that $\mathcal{T}' = \mathcal{T} + F(\mathcal{T}'_0)$, and let us prove that $\mathcal{T}' = \mathcal{T} + F(G(\mathcal{T}'))$. For each theory $\mathcal{T}'$ we know that $F(G(\mathcal{T}')) \subseteq \mathcal{T}'$; since here $\mathcal{T} \subseteq \mathcal{T}'$ we get $\mathcal{T} + F(G(\mathcal{T}')) \subseteq \mathcal{T}'$. For each theory $\mathcal{T}'_0$ we know that $\mathcal{T}'_0 \subseteq G(F(\mathcal{T}'_0))$ and that $G(F(\mathcal{T}'_0)) \subseteq G(\mathcal{T}) + G(F(\mathcal{T}'_0)) \subseteq G(\mathcal{T} + F(\mathcal{T}'_0))$, so that $\mathcal{T}'_0 \subseteq G(\mathcal{T} + F(\mathcal{T}'_0))$; since here $\mathcal{T}' = \mathcal{T} + F(\mathcal{T}'_0)$ we get first $\mathcal{T}'_0 \subseteq G(\mathcal{T}')$ and then $\mathcal{T}' \subseteq \mathcal{T} + F(G(\mathcal{T}'))$. Then, the result for $\mathcal{T}_{max}$ comes from the fact that $G(\mathcal{T}_{max}) = \mathcal{T}_{max,0}$. The last point follows immediately. □

Theorem 4.5.6. *Let us consider a logic $\mathcal{L}_0$, an extension $\mathcal{L}$ of $\mathcal{L}_0$ and the associated Galois connection $F \dashv G$. Let $\mathcal{T}_0$ be a theory of $\mathcal{L}_0$ and $\mathcal{T} = F(\mathcal{T}_0)$. If $\mathcal{T}_0$ is Hilbert-Post complete (in $\mathcal{L}_0$) and $\mathcal{T}$ is relatively Hilbert-Post complete with respect to $\mathcal{L}_0$, then $\mathcal{T}$ is Hilbert-Post complete (in $\mathcal{L}$).*

Proof. Since $\mathcal{T}$ is relatively complete with respect to $\mathcal{L}_0$, it is consistent. Since $\mathcal{T} = F(\mathcal{T}_0)$ we have $\mathcal{T}_0 \subseteq G(\mathcal{T})$. Let $\mathcal{T}'$ be a theory such that $\mathcal{T} \subseteq \mathcal{T}'$. Since $\mathcal{T}$ is relatively complete with respect to $\mathcal{L}_0$, by Lemma 4.5.5 we have $\mathcal{T}' = \mathcal{T} + F(\mathcal{T}'_0)$ where $\mathcal{T}'_0 = G(\mathcal{T}')$. Since $\mathcal{T} \subseteq \mathcal{T}'$, $\mathcal{T}_0 \subseteq G(\mathcal{T})$ and $\mathcal{T}'_0 = G(\mathcal{T}')$, we get $\mathcal{T}_0 \subseteq \mathcal{T}'_0$. Thus, since $\mathcal{T}_0$ is complete, either $\mathcal{T}'_0 = \mathcal{T}_0$ or $\mathcal{T}'_0 = \mathcal{T}_{max,0}$; let us check that then either $\mathcal{T}' = \mathcal{T}$ or $\mathcal{T}' = \mathcal{T}_{max}$. If $\mathcal{T}'_0 = \mathcal{T}_0$ then $F(\mathcal{T}'_0) = F(\mathcal{T}_0) = \mathcal{T}$, so that $\mathcal{T}' = \mathcal{T} + F(\mathcal{T}'_0) = \mathcal{T}$. If $\mathcal{T}'_0 = \mathcal{T}_{max,0}$ then $F(\mathcal{T}'_0) = F(\mathcal{T}_{max,0})$; since $\mathcal{T}$ is relatively complete with respect to $\mathcal{L}_0$, the theory $\mathcal{T}_{max}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$, which implies (by Lemma 4.5.5) that $\mathcal{T}_{max} = \mathcal{T} + F(\mathcal{T}_{max,0}) = \mathcal{T}'$. □

Proposition 4.5.7. *Let $\mathcal{L}_1$ be an intermediate logic between $\mathcal{L}_0$ and $\mathcal{L}$, let $F_1 \dashv G_1$ and $F_2 \dashv G_2$ be the Galois connections associated to the extensions $\mathcal{L}_1$ of $\mathcal{L}_0$ and $\mathcal{L}$ of $\mathcal{L}_1$, respectively. Let $\mathcal{T}_1 = F_1(\mathcal{T}_0)$ and let $\mathcal{T} = F_2(\mathcal{T}_1)$. If $\mathcal{T}_1$ is relatively Hilbert-Post complete with respect to $\mathcal{L}_0$ and $\mathcal{T}$ is relatively Hilbert-Post complete with respect to $\mathcal{L}_1$, then $\mathcal{T}$ is relatively Hilbert-Post complete with respect to $\mathcal{L}_0$.*

Proof. This is an easy consequence of the fact that $F = F_2 \circ F_1$. □

Corollary 4.5.10 provides a characterization of relative Hilbert-Post completeness which is used in Sections 5.4 and 6.9 and in the Coq implementation.

Definition 4.5.8. For each set E of formulae, let $Th(E)$ be the theory generated by E; and when $E = \{e\}$, let $Th(e) = Th(\{e\})$. Then, two sets E_1, E_2 of formulae are *$\mathcal{T}$-equivalent* if $\mathcal{T} + Th(E_1) = \mathcal{T} + Th(E_2)$; and a formula e of $\mathcal{L}$ is *$\mathcal{L}_0$-derivable* from a theory $\mathcal{T}$ of $\mathcal{L}$ if $\{e\}$ is $\mathcal{T}$-equivalent to E_0 for some set E_0 of formulae of $\mathcal{L}_0$.

Proposition 4.5.9 provides a characterization of relative Hilbert-Post completeness which will be used in the next Sections.

Proposition 4.5.9. *Let $\mathcal{T}$ be a theory of $\mathcal{L}$. Each theory $\mathcal{T}'$ of $\mathcal{L}$ containing $\mathcal{T}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$ if and only if each formula e in $\mathcal{L}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$.*

Proof. Let us assume that each theory $\mathcal{T}'$ of $\mathcal{L}$ containing $\mathcal{T}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$. Let e be a formula in $\mathcal{L}$, let $\mathcal{T}' = \mathcal{T} + Th(e)$, and let $\mathcal{T}'_0$ be a theory of $\mathcal{L}_0$ such that $\mathcal{T}' = \mathcal{T} + F(\mathcal{T}'_0)$. The definition of $Th(-)$ is such that $Th(\mathcal{T}'_0) = F(\mathcal{T}'_0)$, so that we get $\mathcal{T} + Th(e) = \mathcal{T} + Th(E_0)$ where $E_0 = \mathcal{T}'_0$. Conversely, let us assume that each formula e in $\mathcal{L}$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$. Let $\mathcal{T}'$ be a theory containing $\mathcal{T}$. Let

$\mathcal{T}'' = \mathcal{T} + F(G(\mathcal{T}'))$, so that $\mathcal{T} \subseteq \mathcal{T}'' \subseteq \mathcal{T}'$ (because $F(G(\mathcal{T}')) \subseteq \mathcal{T}'$ for any $\mathcal{T}'$). Let us consider an arbitrary formula e in $\mathcal{T}'$, by Definition 4.5.8 there is a set E_0 of formulae of $\mathcal{L}_0$ such that $\mathcal{T} + Th(e) = \mathcal{T} + Th(E_0)$. Since e is in $\mathcal{T}'$ and $\mathcal{T} \subseteq \mathcal{T}'$, we have $\mathcal{T} + Th(e) \subseteq \mathcal{T}'$, so that $\mathcal{T} + Th(E_0) \subseteq \mathcal{T}'$. It follows that E_0 is a set of theorems $\mathcal{T}'$ which are formulae of $\mathcal{L}_0$ which means that $E_0 \subseteq G(\mathcal{T}')$, and consequently $Th(E_0) \subseteq F(G(\mathcal{T}'))$, so that $\mathcal{T} + Th(E_0) \subseteq \mathcal{T}''$. Since $\mathcal{T} + Th(e) = \mathcal{T} + Th(E_0)$, we get $e \in \mathcal{T}''$. We have proved that $\mathcal{T}' = \mathcal{T}''$, so that $\mathcal{T}'$ is $\mathcal{L}_0$-derivable from $\mathcal{T}$. □

Corollary 4.5.10. *A theory $\mathcal{T}$ of $\mathcal{L}$ is Hilbert-Post complete with respect to $\mathcal{L}_0$ if and only if it is consistent and for each formula e of $\mathcal{L}$, there is a set E_0 of formulae of $\mathcal{L}_0$ such that $\{e\}$ is $\mathcal{T}$-equivalent to E_0.*

Two interesting aspects of a relatively Hilbert-Post complete theory are given in Theorem 4.5.6 and Proposition 4.5.7. The former shows that relative the Hilbert-Post completeness lifts the (absolute) Hilbert-Post completeness from the sub-logic to the extended logic. The latter reveals the fact that the relative Hilbert-Post completeness is compatible with the composition of logics.

Note again that the base languages of the decorated theory for states and exceptions (in the absence of categorical products and coproducts, respectively) will be proved in Sections 5.4 and 6.9 as *Hilbert-Post complete* with respect to their pure sublogics.

5

The state effect

The use and modification of the memory state is the fundamental feature of imperative languages. For instance, a `C` function may observe the value of a variable or modify it. In order to prove correctness of programs with such features, one has to take into account the use and manipulation of the state. In this chapter, any access to the state is treated as a computational effect: a syntactic term $f : X \to Y$ is not interpreted as $f : X \to Y$ unless it is *pure*. Indeed, a term which reads the memory state has the interpretation $f : X \times S \to Y$, while a term which updates the state is interpreted as $f : X \times S \to Y \times S$, where "$\times$" is the product operator and S is the set of states. In this chapter, we introduce the *decorated logic for the state* ($\mathcal{L}_{st}$), as an extension to the *decorated logic for a comonad* ($\mathcal{L}_{com}$) which has been introduced in Section 4.3. This logic is used to prove equivalence of programs involving the state effect, while keeping the memory accesses and manipulations implicit, as described in [DDEP14]. We obtain the decorations of the logic $\mathcal{L}_{st}$, for terms and equations, from the logic $\mathcal{L}_{com}$. In addition, we introduce the interface functions `lookup` and `update` for state access and manipulation, respectively. Since, in the presence of the state effect, the result of evaluating the arguments may depend on the order in which they are evaluated, we use a decorated version of categorical products as in [DDR11] to impose an order in evaluating the arguments of multivariate operations. In Figure 5.1 we instantiate the comonad D in Figure 4.6 with the comonad of states:

$\mathcal{L}_{st}$		$\longrightarrow$ *Interpretation of* $\mathcal{L}_{st}$
modifier	$f^{(2)} : X \to Y$	$f : X \times S \to Y \times S$
accessor	$f^{(1)} : X \to Y$	$f : X \times S \to Y$
pure term	$f^{(0)} : X \to Y$	$f : X \to Y$
strong equation	$f^{(2)} \equiv g^{(2)} : X \to Y$	$f = g : X \times S \to Y \times S$
weak equation	$f^{(2)} \sim g^{(2)} : X \to Y$	$\pi_1 \circ f = \pi_1 \circ g : X \times S \to Y$
		where $\pi_1 \colon X \times S \to X$ is the first projection

Figure 5.1: The decorated logic $\mathcal{L}_{st}$ and its interpretation: an overview.

We start, in Section 5.1, with the syntax of the *decorated logic for the state* ($\mathcal{L}_{st}$) with its interpretation given via the Kleisli-on-coKleisli construction associated to the states comonad. The Coq implementation of the logic $\mathcal{L}_{st}$ is presented in Section 5.2. In Section 5.3, we prove some properties of the state effect as in [PP02, §3], but here in a decorated setting. Lastly, the logic $\mathcal{L}_{st}$ (without products) is proven to be relatively *Hilbert-Post complete* in Section 5.4.

5.1 The decorated logic for the state

The decorated logic for the state ($\mathcal{L}_{st}$) extends the decorated logic for a comonad ($\mathcal{L}_{com}$) with the product types (sorts), the singleton type $\mathbb{1}$ and the type V_i of values that can be stored in any location $i \in Loc$ where Loc is a finite set. Similar to the terms (operations) of the logic $\mathcal{L}_{com}$, each term has a source and a target type. Additionally, there is a (left) pair term $\langle f, g \rangle_l \colon X \to Y_1 \times Y_2$ for each couple of terms $f \colon X \to Y_1$ and $g \colon X \to Y_2$. For each product type $X \times Y$, there are canonical projections $\pi_1 \colon X \times Y \to X$ and $\pi_2 \colon X \times Y \to Y$. The symbol $\langle\ \rangle_X$ denotes the unique term from X to the singleton type $\mathbb{1}$ for each type X. The term $\mathtt{lookup}_i \colon \mathbb{1} \to V_i$ stands to observe the content of a given location i while the term $\mathtt{update}_i \colon V_i \to \mathbb{1}$ is used to modify it. We give the syntax of $\mathcal{L}_{st}$ in Figure 5.2 and its inference rules in Figures 5.3, 5.4 and 5.5 in addition to the ones stated in Figure 4.7.

Grammar of the decorated logic for the state: $(i \in Loc)$

Types:	$\mathtt{t}, \mathtt{s}$	::=	$\mathtt{X} \mid \mathtt{Y} \mid \cdots \mid \mathtt{t} \times \mathtt{s} \mid \mathbb{1} \mid \mathtt{V_i}$
Terms:	$\mathtt{f}, \mathtt{g}$	::=	$\mathtt{id_t} \mid \mathtt{a} \mid \mathtt{b} \mid \cdots \mid \mathtt{g} \circ \mathtt{f} \mid \langle \mathtt{f}, \mathtt{g} \rangle_1 \mid \pi_1 \mid \pi_2 \mid \langle\ \rangle_\mathtt{t} \mid \mathtt{lookup_i} \mid \mathtt{update_i}$
Decoration for terms:	$(\mathtt{d})$	::=	$(0) \mid (1) \mid (2)$
Equations:	$\mathtt{e}$	::=	$\mathtt{f} \equiv \mathtt{g} \mid \mathtt{f} \sim \mathtt{g}$

Figure 5.2: $\mathcal{L}_{st}$: syntax

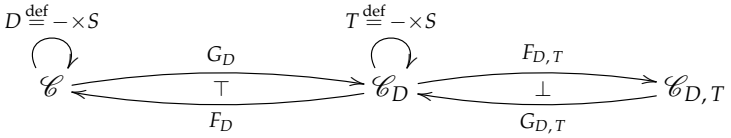
Each term has a decoration which is denoted as a superscript (0), (1) or (2): a *pure* term has the decoration (0), an *observer* (or *accessor*) has (1) and a *modifier* term comes with the decoration (2). Similarly, each equation is formed by two terms with the same source and target as well as a decoration, denoted by "$\sim$" if it is *weak* or by "$\equiv$" if it is *strong*.

Let $\mathscr{C}$ be a category with finite products and a distinguished object of states S. Let $(D = - \times S, \varepsilon, \delta)$ be the states comonad defined over $\mathscr{C}$. Let us assume that S is given such that the the epi-requirement is satisfied (Definition 3.1.7). For instance, when $\mathscr{C}$ is the category of sets, then S cannot be the empty set.

The interpretation of $\mathcal{L}_{st}$ is given via the Kleisli-on-coKleisli construction associated to a comonad, detailed in Section 3.3, applied to the states comonad. Recall that in Section 3.3.2, we have introduced the adjunctions $F_D \dashv G_D$ and $F_{D,T} \dashv G_{D,T}$ with the faithful functors $G_D \colon \mathscr{C} \to \mathscr{C}_D$ and $F_{D,T} \colon \mathscr{C}_D \to \mathscr{C}_{D,T}$. This gives raise to a hierarchy among terms in $\mathscr{C}_{D,T}$. We use this hierarchy to interpret the decorations: *pure* terms are in $\mathscr{C}$, *accessors* are in $\mathscr{C}_D$ and *modifiers* are in $\mathscr{C}_{D,T}$.

Definition 5.1.1. Let $\mathbf{C_{ST}}$ the interpretation of the syntax for the logic $\mathcal{L}_{st}$ with the

following details:



$\varepsilon : D \Rightarrow Id \qquad F_D \dashv G_D \qquad \eta : Id \Rightarrow T$

(1) The types are interpreted as the objects of $\mathscr{C}$.

(1.1) the unit type $\mathbb{1}$ is interpreted by the *final object* of the category $\mathscr{C}$.

(1.2) for each i in *Loc*, the type V_i is interpreted as an object Val_i.

(1.3) for each pair of types X and Y, the product types $X \times Y$ are interpreted as the binary products in $\mathscr{C}$.

Now, we can define the object of states as $S = \Pi_{i \in Loc} Val_i$. The projections are denoted $\pi_i \colon S \to Val_i$, for each location i. The object S in $\mathscr{C}$ is not the interpretation of a "type of states". Indeed, the use of decorations in the logic $\mathcal{L}_{st}$ provides a signature without any occurrence of such a "type of states". So that signature is kept close to the syntax. Besides, for each object X in $\mathscr{C}$, the first projection $\pi_{1,X,S} \colon X \times S \to X$ is ε_X and the second projection $\pi_{2,X,S} \colon X \times S \to S$, up to the isomorphism between of S and $\mathbb{1} \times S$, is $D(\langle\,\rangle_X)$.

(2) The terms are interpreted as the morphisms as follows:

(2.1) a *pure* term $f^{(0)} \colon X \to Y$ in $\mathscr{C}$ as $f \colon X \to Y$ in $\mathscr{C}$

(2.2) an *accessor* term $f^{(1)} \colon X \to Y$ in $\mathscr{C}_D$ as $f \colon X \times S \to Y$ in $\mathscr{C}$

(2.3) a *modifier* term $f^{(2)} \colon X \to Y$ in $\mathscr{C}_{D,T}$ as $f \colon X \times S \to Y \times S$ in $\mathscr{C}$

(3) The terms $f_1^{(1)} \colon X \to Y_1$ and $f_2^{(2)} \colon X \to Y_2$ are interpreted as $f_1 \colon X \times S \to Y_1$ and $f_2 \colon X \times S \to Y_2 \times S$ in $\mathscr{C}$. Thus, $\langle f_1, f_2 \rangle_l^{(2)} \colon X \to Y_1 \times Y_2$ is interpreted as the categorical pair $\langle f_1, f_2 \rangle \colon X \times S \to Y_1 \times Y_2 \times S$ in $\mathscr{C}$. It is called the *left pair* of f and g.

(4) The pure projections $\pi_1^{(0)} \colon X \times Y \to X$ and $\pi_2^{(0)} \colon X \times Y \to Y$ are interpreted as the canonical projections $\pi_1 \colon X \times Y \to X$ and $\pi_2 \colon X \times Y \to Y$ associated to pairs.

(5) The pure term $\langle\,\rangle_X^{(0)} \colon X \to \mathbb{1}$ is interpreted as the unique mapping from X to the final object $\mathbb{1}$ in $\mathscr{C}$.

(6) For each i in *Loc*, the term $\texttt{lookup}_i^{(1)} \colon \mathbb{1} \to V_i$ is an *accessor* in $\mathscr{C}_D$ and interpreted as $\texttt{lookup}_i = \pi_i \colon S \to Val_i$ in $\mathscr{C}$ (up to the isomorphism between $\mathbb{1} \times S$ and S).

(7) For each i, the term $\texttt{update}_i^{(2)} \colon V_i \to \mathbb{1}$ is a *modifier* in $\mathscr{C}_{D,T}$ and its interpretation is characterized by the following equalities: for each j in *Loc* such that $i \neq j$, $\pi_j \circ \texttt{update}_i = \pi_j \circ \pi_{2,Val_i,S} \colon Val_i \times S \to Val_j$ and $\pi_i \circ \texttt{update}_i = \pi_{1,Val_i,S} \colon Val_i \times S \to Val_i$.

(8) A strong equation between modifiers $f^{(2)} \equiv g^{(2)} \colon X \to Y$ in $\mathscr{C}_{D,T}$ is interpreted by an equality $f = g \colon X \times S \to Y \times S$ in $\mathscr{C}$. Similarly, a strong equation between accessors $f^{(1)} \equiv g^{(1)} \colon X \to Y$ in $\mathscr{C}_D$ is interpreted by an equality $f = g \colon X \times S \to Y$ in $\mathscr{C}$. And a strong equation between pure terms $f^{(0)} \equiv g^{(0)} \colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g \colon X \to Y$ in $\mathscr{C}$. Intuitively, two terms are strongly equal if they have equivalences on returned results and effects on the state.

(9) A weak equation between modifiers $f^{(2)} \sim g^{(2)} \colon X \to Y$ in $\mathscr{C}_{D,T}$ is interpreted by an equality $\varepsilon_Y \circ f = \varepsilon_Y \circ g \colon X \times S \to Y$ in $\mathscr{C}$. Similarly, a weak equation between accessors $f^{(1)} \sim g^{(1)} \colon X \to Y$ in $\mathscr{C}_D$ is interpreted by an equality $f = g \colon X \times S \to Y$ in $\mathscr{C}$. And a weak equation between pure terms $f^{(0)} \sim g^{(0)} \colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g \colon X \to Y$ in $\mathscr{C}$. Intuitively, two terms are weakly equal if they return the same result with maybe different effects on the state.

The rules of the logic $\mathcal{L}_{com}$, as stated in Figure 4.7, are rules of the logic $\mathcal{L}_{st}$. Now, we introduce the additional rules of the logic $\mathcal{L}_{st}$ in several steps, with some comments.

5.1.1 The effect rule

the effect rule

$$\text{(effect)}\ \frac{f_1, f_2 \colon X \to Y \quad f_1 \sim f_2 \quad \langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2}{f_1 \equiv f_2}$$

Figure 5.3: $\mathcal{L}_{st}$: the effect rule

(effect) This rule states that weak and strong equations are related with the property that $f_1 \equiv f_2$ if and only if $f_1 \sim f_2$ and $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2$. In other words, two terms f_1 and f_2 are strongly equal if and only if they have the "same result" ($f_1 \sim f_2$) and "the same effect" ($\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2$).

5.1.2 The pair rules

rules for the left pairs

$$\text{(unit)}\ \frac{X}{\langle\ \rangle_X^{(0)} \colon X \to \mathbb{1}} \quad \text{(w-unit)}\ \frac{f \colon X \to \mathbb{1}}{f \sim \langle\ \rangle_X}$$

$$\text{(lpair)}\ \frac{f_1^{(d)} \colon X \to Y_1 \quad f_2 \colon X \to Y_2}{\langle f_1, f_2 \rangle_l \colon X \to Y_1 \times Y_2}\ (\text{for all } d \leq 1) \quad \text{(proj)}\ \frac{i \in \{1,2\} \quad Y_1 \quad Y_2}{\pi_i^{(0)} \colon Y_1 \times Y_2 \to Y_i}$$

$$\text{(s-lpair-eq)}\ \frac{f_1^{(d)} \colon X \to Y_1 \quad f_2 \colon X \to Y_2}{\pi_2 \circ \langle f_1, f_2 \rangle_l \equiv f_2}\ (\text{for all } d \leq 1)$$

$$\text{(w-lpair-eq)}\ \frac{f_1^{(d)} \colon X \to Y_1 \quad f_2 \colon X \to Y_2}{\pi_1 \circ \langle f_1, f_2 \rangle_l \sim f_1}\ (\text{for all } d \leq 1)$$

$$\text{(lpair-ueq)}\ \frac{f_1, f_2 \colon X \to Y_1 \times Y_2 \quad \pi_1 \circ f_1 \sim \pi_1 \circ f_2 \quad \pi_2 \circ f_1 \sim \pi_2 \circ f_2}{f_1 \sim f_2}$$

Figure 5.4: $\mathcal{L}_{st}$: rules for left pairs

(w-unit) This rule intuitively means that for each modifier term $f\colon X \to \mathbb{1}$, there is an obvious result equivalence between f and the unique mapping $\langle\ \rangle_X$, since both return *void*. The unique instance of type $\mathbb{1}$ can be used to interpret the result *void*.

(lpair) This rule states that the left pair $\langle f_1, f_2\rangle_l$ is defined only when f_1 is pure or accessor. Indeed, when both f_1 and f_2 are modifiers, such a construction would lead to a conflict on the returned result. When f_1 is an accessor, with (w-lpair-eq), we ensure that $\langle f_1, f_2\rangle_l$ returns "the same" result with f_1 and with (s-lpair-eq) that $\langle f_1, f_2\rangle_l$ returns "the same" result with f_2 along with "the same" manipulation on the state.

(lpair-ueq) This rule ensures that a left pair is unique up to the weak equation.

5.1.3 Some properties of pairs

In this section, we start with a property of the "empty pair" and then prove the unicity of left pairs up to the strong equation. Afterwards, we build the symmetric (or right) pairs by using the left pairs and prove some of their properties. Lastly, we construct the left and right products, by respectively using left and right pairs, and similarly prove some related properties.

Proposition 5.1.2. (s-unit) *For all d, $d' \leq 1$, given two terms of the form $f_1^{(d)}$, $f_2^{(d')}\colon X \to \mathbb{1}$ for each X, then $f_1 \equiv f_2$.*

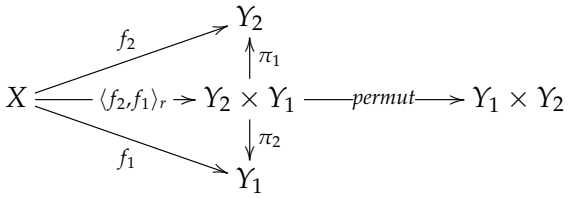
Proof. Obviously, $f_1 \sim f_2$ thanks to (w-unit). Since none of them is a modifier, then $f_1 \equiv f_2$ due to (wtos). □

Proposition 5.1.3. (lpair-u) *For each $f_1, f_2\colon X \to Y_1 \times Y_2$, if $\pi_1 \circ f_1 \sim \pi_1 \circ f_2$ and $\pi_2 \circ f_1 \equiv \pi_2 \circ f_2$, then $f_1 \equiv f_2$.*

Proof.
1. Starting from $\pi_2 \circ f_1 \equiv \pi_2 \circ f_2$, we obtain $\langle\ \rangle_{Y_2} \circ \pi_2 \circ f_1 \equiv \langle\ \rangle_{Y_2} \circ \pi_2 \circ f_2$ due to (replsubs). Besides, we have $\langle\ \rangle_{Y_2} \circ \pi_2 \equiv \langle\ \rangle_{Y_1 \times Y_2}$ thanks to (s-unit). Therefore, we get $\langle\ \rangle_{Y_1 \times Y_2} \circ f_1 \equiv \langle\ \rangle_{Y_1 \times Y_2} \circ f_2$.
2. Since we have $\pi_2 \circ f_1 \equiv \pi_2 \circ f_2$, by converting the strong equation into a weak equation, we get $\pi_2 \circ f_1 \sim \pi_2 \circ f_2$. In addition, $\pi_1 \circ f_1 \sim \pi_1 \circ f_2$ is also assumed so that we end up with $f_1 \sim f_2$ thanks to (lpair-ueq).

Now, the above items 1 and 2 suffice to ensure $f_1^{(2)} \equiv f_2^{(2)}$ due to (effect) rule introduced in Figure 5.3. □

Definition 5.1.4. For all $d \leq 1$, given $f_1\colon X \to Y_1$ and $f_2^{(d)}\colon X \to Y_2$, the *right pair* $\langle f_1, f_2\rangle_r = \mathit{permut} \circ \langle f_2, f_1\rangle_l$ where $\mathit{permut} = \langle \pi_2, \pi_1\rangle_l$.



Proposition 5.1.5. *For all $d \leq 1$, given $f_1\colon X \to Y_1$ and $f_2^{(d)}\colon X \to Y_2$, we have:*

- $\pi_1 \circ \langle f_1, f_2\rangle_r \equiv f_1$ (s-rpair-eq)

- $\pi_2 \circ \langle f_1, f_2 \rangle_r \sim f_2^{(d)}$ (w-rpair-eq)

Proof. • Due to (w-lpair-eq), we have $\pi_1 \circ \langle \pi_2, \pi_1 \rangle_l \sim \pi_2$. Since terms are all pure, the weak equation converts into the strong one: $\pi_1 \circ \langle \pi_2, \pi_1 \rangle_l \equiv \pi_2$. Through (replsubs), we obtain $\pi_1 \circ \langle \pi_2, \pi_1 \rangle_l \circ \langle f_2, f_1 \rangle_l \equiv \pi_2 \circ \langle f_2, f_1 \rangle_l$. Now, (s-lpair-eq) gives $\pi_1 \circ \langle \pi_2, \pi_1 \rangle_l \circ \langle f_2, f_1 \rangle_l \equiv f_1$ which folds into $\pi_1 \circ \langle f_1, f_2 \rangle_r \equiv f_1$.

- Thanks to (s-lpair-eq) and (stow), we get $\pi_2 \circ \langle \pi_2, \pi_1 \rangle_l \sim \pi_1$. The rule (wsubs) gives $\pi_2 \circ \langle \pi_2, \pi_1 \rangle_l \circ \langle f_2, f_1 \rangle_l \sim \pi_1 \circ \langle f_2, f_1 \rangle_l$. By using (w-lpair-eq), we obtain $\pi_2 \circ \langle \pi_2, \pi_1 \rangle_l \circ \langle f_2, f_1 \rangle_l \sim f_2^{(d)}$ which is actually $\pi_2 \circ \langle f_1, f_2 \rangle_r \sim f_2^{(d)}$.

□

Proposition 5.1.6. (rpair-u) *For each* $f_1, f_2 \colon X \to Y_1 \times Y_2$, *if* $\pi_1 \circ f_1 \equiv \pi_1 \circ f_2$ *and* $\pi_2 \circ f_1 \sim \pi_2 \circ f_2$, *then* $f_1 \equiv f_2$.

Proof. 1. Starting from $\pi_1 \circ f_1 \equiv \pi_1 \circ f_2$, we obtain $\langle\ \rangle_{Y_1} \circ \pi_1 \circ f_1 \equiv \langle\ \rangle_{Y_1} \circ \pi_1 \circ f_2$ due to (replsubs). Besides, we have $\langle\ \rangle_{Y_1} \circ \pi_1 \equiv \langle\ \rangle_{Y_1 \times Y_2}$ thanks to (s-unit). Therefore, we get $\langle\ \rangle_{Y_1 \times Y_2} \circ f_1 \equiv \langle\ \rangle_{Y_1 \times Y_2} \circ f_2$.

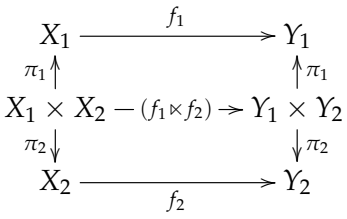
2. Since we have $\pi_1 \circ f_1 \equiv \pi_1 \circ f_2$, by converting the strong equation into a weak equation, we get $\pi_1 \circ f_1 \sim \pi_1 \circ f_2$. In addition, $\pi_2 \circ f_1 \sim \pi_2 \circ f_2$ is also assumed so that we end up with $f_1 \sim f_2$ thanks to (lpair-ueq).

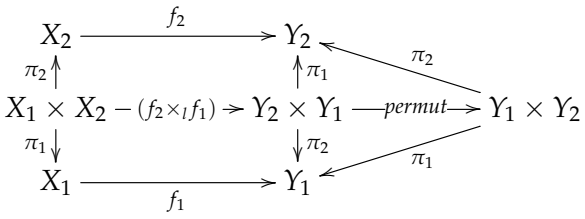
Now, the above items 1 and 2 suffice to ensure $f_1 \equiv f_2$ due to (effect) rule introduced in Figure 5.3. □

One can define the *left and right products of terms* respectively using left and right pairs.

Definition 5.1.7. • For all $d \leq 1$, given $f_1^{(d)} \colon X_1 \to Y_1$ and $f_2 \colon X_2 \to Y_2$, we obtain a *left product* $(f_1 \times_l f_2) = \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle_l \colon X_1 \times X_2 \to Y_1 \times Y_2$.

- For all $d \leq 1$, given $f_1 \colon X_1 \to Y_1$ and $f_2^{(d)} \colon X_2 \to Y_2$, we obtain a *right product* $(f_1 \times_r f_2) = \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle_r = \mathit{permut} \circ \langle f_2 \circ \pi_2, f_1 \circ \pi_1 \rangle_l \colon X_1 \times X_2 \to Y_1 \times Y_2$ such that $\mathit{permut} = \langle \pi_2, \pi_1 \rangle_l$.





Proposition 5.1.8. *For all* $d \leq 1$, *given* $f_1^{(d)} \colon X_1 \to Y_1$ *and* $f_2 \colon X_2 \to Y_2$, *we have:*

- $\pi_1 \circ (f_1 \times_l f_2) \sim f_1^{(d)} \circ \pi_1$ (w-lprod-eq)
- $\pi_2 \circ (f_1 \times_l f_2) \equiv f_2 \circ \pi_2$ (s-lprod-eq)

Proof. • By setting $f_1 := f_1 \circ \pi_1$ and $f_2 := f_2 \circ \pi_2$ within (w-lpair-eq), one gets $\pi_1 \circ \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle_l \sim f_1^{(d)} \circ \pi_1$ which folds into $\pi_1 \circ (f_1 \times_l f_2) \sim f_1^{(d)} \circ \pi_1$.

- Similarly, we set $f_1 := f_1 \circ \pi_1$ and $f_2 := f_2 \circ \pi_2$ within (s-lpair-eq), one gets $\pi_2 \circ \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle_l \equiv \pi_2 \circ f_2$ which folds into $\pi_2 \circ (f_1 \times_l f_2) \equiv f_2 \circ \pi_2$. □

Proposition 5.1.9. (lprod-u) *For each* $f_1, f_2 \colon X_1 \times X_2 \to Y_1 \times Y_2$, *if* $\pi_1 \circ f_1 \sim \pi_1 \circ f_2$ *and* $\pi_2 \circ f_1 \equiv \pi_2 \circ f_2$, *then* $f_1 \equiv f_2$.

Proof. It suffices to apply (lpair-u). □

Proposition 5.1.10. *For all* $d \leq 1$, *given* $f_1 \colon X_1 \to Y_1$ *and* $f_2^{(d)} \colon X_2 \to Y_2$, *we have:*

- $\pi_1 \circ (f_1 \times_r f_2) \equiv f_1 \circ \pi_1$ (s-rprod-eq)
- $\pi_2 \circ (f_1 \times_r f_2) \sim f_2^{(d)} \circ \pi_2$ (w-rprod-eq)

Proof.
- By setting $f_1 := f_1 \circ \pi_1$ and $f_2 := f_2 \circ \pi_2$ within (s-rpair-eq), one gets $\pi_1^{(0)} \circ \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle_r \equiv \pi_1 \circ f_1$ which folds into $\pi_1 \circ (f_1 \times_r f_2) \equiv \pi_1 \circ f_1$.
- Similarly, we set $f_1 := \pi_1 \circ f_1$ and $f_2 := \pi_2 \circ f_2$ within (w-rpair-eq), one gets $\pi_2 \circ \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle_r \sim \pi_2 \circ f_2^{(d)}$ which folds into $\pi_2 \circ (f_1 \times_r f_2) \sim \pi_2 \circ f_2^{(d)}$. □

Proposition 5.1.11. (rprod-u) *For each* $f_1, f_2 \colon X_1 \times X_2 \to Y_1 \times Y_2$, *if* $\pi_2 \circ f_1 \sim \pi_2 \circ f_2$ *and* $\pi_1 \circ f_1 \equiv \pi_1 \circ f_2$, *then* $f_1 \equiv f_2$.

Proof. It suffices to apply (rpair-u). □

Remark 5.1.12. The product of two modifier terms $f_1 \colon X_1 \to Y_1$ and $f_2 \colon X_2 \to Y_2$ is modeled by their *sequential products*, as introduced in [DDR11], which impose some order of evaluation of the arguments: a sequential product is obtained as the sequential composition of two semi-pure products. A semi-pure product, as far as we are concerned in this thesis, is a kind of product of an identity function (which is pure) with a modifier function.

Notice that we use some of these properties when proving the properties of programs with the state effect, in Section 5.3.

5.1.4 The interface rules

interface rules

$$\text{(lookup)}\ \frac{i \in Loc}{\mathtt{lookup}_i^{(1)} \colon \mathbb{1} \to V_i} \qquad \text{(update)}\ \frac{i \in Loc}{\mathtt{update}_i^{(2)} \colon V_i \to \mathbb{1}}$$

$$\text{(ax}_1\text{)}\ \frac{i \in Loc}{\mathtt{lookup}_i \circ \mathtt{update}_i \sim id_{V_i}} \qquad \text{(ax}_2\text{)}\ \frac{i, j \in Loc \quad i \neq j}{\mathtt{lookup}_j \circ \mathtt{update}_i \sim \mathtt{lookup}_j \circ \langle\ \rangle_{V_i}}$$

$$\text{(local-global)}\ \frac{g_1, g_2 \colon X \to \mathbb{1} \quad \text{for all } i \in Loc\ \mathtt{lookup}_i \circ g_1 \sim \mathtt{lookup}_i \circ g_2}{g_1 \equiv g_2}$$

Figure 5.5: $\mathcal{L}_{st}$: the interface rules

(ax$_1$) This rule states that one obtains the value v after both of the following cases:

(a) first storing v into a location i, then observing the same location,

(b) feeding v to the identity term.

Clearly, the equation between operations in (a) and (b) is weak since they have different manipulations on the state.

(ax$_2$) This rule states for each couple of different locations i and j that after below cases (c) and (d), one obtains the same result but different manipulations on the state:

(c) first storing a value v into a location j and then observing a different location i

(d) first forgetting the value v then observing the location i.

Notice that the operation in (c) is a modifier while the one in (d) is an accessor.

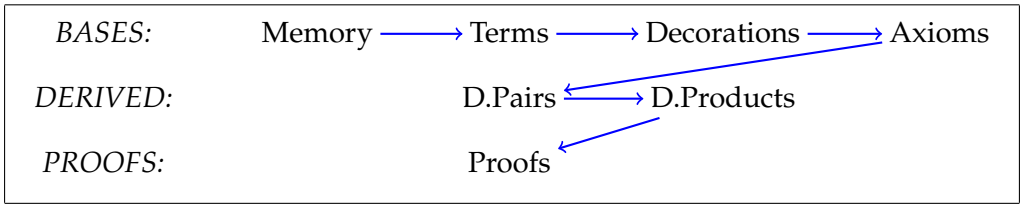
(local-global) This rule means that for each location i, the statement of the (effect) rule can be expressed as a pair of weak equations for $g_1 = \langle\ \rangle_Y \circ f_1$ and $g_2 = \langle\ \rangle_Y \circ f_2$: $g_1 \sim g_2$ and $\texttt{lookup}_i \circ g_1 \sim \texttt{lookup}_i \circ g_2$. Since $g_1, g_2 \colon X \to \mathbb{1}$ return the same result (void), there is no explicit need to check whether $g_1 \sim g_2$ or not. It suffices to check if $\texttt{lookup}_i \circ g_1 \sim \texttt{lookup}_i \circ g_2$ holds, in order to reason whether $g_1 \equiv g_2$ or not. Note that this rule needs to be reformulated when we formalize the *local state effect* (or *dynamic allocation*) which is beyond the scope of this thesis.

Now, the following result is easily obtained:

Theorem 5.1.13. *The logic $\mathcal{L}_{st}$ is sound with respect to the interpretation $\mathbf{C_{ST}}$ given in Definition 5.1.1.*

5.2 Coq implementation: $\mathcal{L}_{st}$

The main scope of this section is to formalize the *decorated logic for the state* ($\mathcal{L}_{st}$) in Coq [DDEP14]. To do so, we aim to enrich the implementation of the logic $\mathcal{L}_{com}$ that is already detailed in Sections 4.4.1 4.4.2 4.4.3: we will reuse the code blocks in order to preserve the integrity of the formalization with no repeated explanation. The organization of the modules is reflected in the Coq library `STATES-THESIS` as follows:



Remark 5.2.1. The complete `STATES-THESIS` library can be found on `https://forge.imag.fr/frs/download.php/695/STATES-THESIS.tar.gz`.

5.2.1 Memory

In order to enrich the terms of the logic $\mathcal{L}_{com}$ (or $\mathcal{L}_{mon}$), we first need to speak about some preliminaries: the set of memory locations is implemented by a Coq parameter, `Loc : Type`. Provided that each location may contain different type of values, we also implement an arrow type `V : Loc → Type` that is the type of values stored in any location.

```
Parameter Loc: Type.    Parameter V: Loc → Type.
```

5.2.2 Terms

We can implement the additional terms, as new constructors to the dependent type `term` that has been given in Section 4.4.1, as follows:

(1) $\langle f, g \rangle \colon X \to Y \times Z$ for each pair of terms $f \colon X \to Y$ and $g \colon X \to Z$, together with the canonical projections $\pi_1 \colon Y \times Z \to Y$ and $\pi_2 \colon Y \times Z \to Z$,

(2) $\langle\,\rangle_X \colon X \to \mathbb{1}$ for each type X,,

(3) $\texttt{lookup}_i \colon \mathbb{1} \to V_i$ for each location i,

(4) $\texttt{update}_i \colon V_i \to \mathbb{1}$ for each location i.

Thus, the implementation of terms in Coq looks like:

```
Inductive term: Type → Type → Type :=
 | comp:   forall {X Y Z: Type}, term X Y → term Y Z → term X Z
 | tpure:  forall {X Y: Type}, (X → Y) → term Y X
 | pair: forall {X Y Z: Type}, term X Z → term Y Z → term (X*Y) Z
 | lookup: forall i:Loc, term (V i) unit
 | update: forall i:Loc, term unit (V i).
Infix "o" := comp (at level 70).
```

Instead of the symbol $\langle\,\rangle$, we use the keyword `pair` in the implementation. The terms such as the identity, the pair projections, the empty pair and the constant function can be derived from the native Coq functions, with the use of `tpure` constructor, as follows:

```
Definition id {X: Type}     : term X X      := tpure id.
Definition pi1 {X Y: Type}  : term X (X*Y)  := tpure fst.
Definition pi2 {X Y: Type}  : term Y (X*Y)  := tpure snd.
Definition forget {X}       : term unit X   := tpure (fun _ ⇒ tt).
Definition constant {X: Type} (v: X): term X unit := tpure (fun tt ⇒ v).
```

Remark also that the pair projections are named `pi1` and `pi2` while the unique mapping $\langle\,\rangle_X$ from any type X to $\mathbb{1}$ is called `forget` in the implementation.

Remark 5.2.2. See the source `Terms.v` for related implementation details.

5.2.3 Decorations

Thereby, the decorations' implementation follows:

```
Inductive kind := pure | ro | rw.
Inductive is: kind → forall X Y, term X Y → Prop :=
 | is_tpure: forall X Y (f: X → Y), is pure (@tpure X Y f)
 | is_comp: forall k X Y Z (f: term X Y) (g: term Y Z), is ro f → is k f → is k g → is k (f o g)
 | is_pair: forall k X Y Z (f: term X Z) (g: term Y Z), is k f → is k g → is k (pair f g)
 | is_lookup: forall i, is ro (lookup i)
 | is_update: forall i, is rw (update i)
 | is_pure_ro: forall X Y (f: term X Y), is pure f → is ro f
 | is_ro_rw: forall X Y (f: term X Y), is ro f → is rw f.
Hint Constructors is.
```

Notice that instead of the decorations of the form (0), (1) and (2), we respectively use the keywords `pure`, `ro` and `rw` in the implementation. The decoration of any composed or paired off term depends on its components and always takes the upper decoration (`pure` $<$ `ro` $<$ `rw`). E.g., given a modifier term and a read-only term, their composition will be a modifier, as well. The decoration of a pair construction depends on its second component, since the first one should at most be a read-only term. Hence, we

cannot form pairs of two modifier terms. The pair construction always takes the upper decoration. For instance, given a pure term and a read-only term, their pair will be a read-only, as well. We declare the term `lookup` as an accessor. On the contrary, `update` is a modifier. It is trivial to derive that the pair projections are pure. For the sake of conciseness, we demonstrate only the first one:

```
Lemma is_pi1 X Y: is pure (@pi1 X Y).
Proof. apply is_tpure. Qed.
```

Since `pi1` is constructed through `tpure` and since any argument of `tpure` is by definition pure, it suffices to apply the constructor `is_tpure`. The process of *decoration checking* is crucial and troublesome in a decorated setting: the use of rules is determined after ensuring that the related terms have the intended decorations. It is possible to automate the verification of decorations. To do so, we create a new tactic named `decorate`, by using Delahaye's `Ltac` language [Del00]:

```
Ltac decorate := solve[
  repeat (apply is_comp || apply is_pair)
             ||
         (is_tpure || apply is_lookup || apply is_update || assumption)
             ||
         (apply is_pure_ro)
             ||
         (apply is_ro_rw) ].
```

The tactic `decorate` repeatedly checks if the goal term is a composition or a pair, if not, it tries to decide whether the term is pure constructed by `tpure` or one of the following terms: `lookup` and `update` or else a local assumption. If it is still not the case, it applies the hierarchy rules. All that are performed in the given sequence. Since these checks are all done inside the `solve` tactical, `decorate` fails in the absence of match.

```
Class PURE {X Y: Type} (f: term X Y) := isp : is pure f.
Hint Extern 0 (PURE _) ⇒ decorate : typeclass_instances.

Class RO {X Y: Type} (f: term X Y) := isro : is ro f.
Hint Extern 0 (RO _) ⇒ decorate : typeclass_instances.

Class RW {X Y: Type} (f: term X Y) := isrw : is rw f.
Hint Extern 0 (RW _) ⇒ decorate : typeclass_instances.
```

The assignment of decorations over terms is declared as constructors of Coq type classes parametrized by a `term`. Then, we extend the scope of the tactic `auto` with the optional patterns `(PURE _)`, `(RO _)` and `(RW _)`, the tactic `decorate` at cost zero. This is provided by the vernacular command `Extern (num) pattern => tactic`. The zero cost means that the tactic `auto` would non-recursively try the hints upon the usage.

Remark 5.2.3. See the source `Decorations.v` for related implementation details.

5.2.4 Axioms

Here we give the formalization of the rules/axioms in Coq.

```
Reserved Notation "x == y" (at level 80).          Reserved Notation "x ~ y" (at level 80).
Definition idem X Y (x y: term X Y) := x = y.
Inductive strong: forall X Y, relation (term X Y) :=
      (*congruence rules*)
 | refl X Y: Reflexive (@strong X Y)
 | sym: forall X Y, Symmetric (@strong X Y)
 | trans: forall X Y, Transitive (@strong X Y)
 | replsubs: forall X Y Z,  Proper (@strong X Y ==> @strong Y Z ==> @strong X Z) comp
      (*categorical rules*)
 | ids: forall X Y (f: term X Y), f o id == f
 | idt: forall X Y (f: term X Y), id o f == f
 | assoc: forall X Y Z T (f: term X Y) (g: term Y Z) (h: term Z T), f o (g o h) == (f o g) o h
      (*the hierarchy rule*)
 | wtos: forall X Y (f g: term X Y), RO f → RO g → f ~ g → f == g
      (*strong pair rules*)
 | s_lpair_eq: forall X Y' Y (f1: term Y X) (f2: term Y' X), RO f1 → pi2 o pair f1 f2 == f2
      (*the effect rule*)
 | effect: forall X Y (f g: term Y X), forget o f == forget o g → f ~ g → f == g
      (*the strong interface rule*)
 | local_global: forall X (f g: term unit X), (forall i: Loc, lookup i o f ~ lookup i o g) → f == g
    (*tpure preserves the pure composition*)
 | tcomp: forall X Y Z (f: Z → Y) (g: Y → X), tpure (compose g f) == tpure g o tpure f
with weak: forall X Y, relation (term X Y) :=
      (*congruence rules*)
 | wsym: forall X Y, Symmetric (@weak X Y)
 | wtrans: forall X Y, Transitive (@weak X Y)
 | pwrepl: forall X Y Z (g: term X Y), is pure g → Proper (@weak Y Z ==> @weak X Z) (comp g)
 | wsubs: forall X Y Z, Proper (@weak X Y ==> @idem Y Z ==> @weak X Z) comp
      (*the hierarchy rule*)
 | stow: forall X Y (f g: term X Y), f == g → f ~ g
      (*the weak pair rule*)
 | w_lpair_eq: forall X Y' Y (f1: term Y X) (f2: term Y' X), RO f1 → pi1 o pair f1 f2 ~ f1
 | w_unit: forall X (f g: term unit X), f ~ g
      (*weak interface rules*)
 | ax1: forall i, lookup i o update i ~ id
 | ax2: forall i j, i<>j → lookup j o update i ~ lookup j o forget
      (*the weak unicity rule*)
 | lpair_ueq: forall X Y Y'(f g: term (Y*Y') X), pi1 o f ~ pi1 o g → pi2 o f ~ pi2 o g → f ~ g
where "x == y" := (strong x y) and "x ~ y" := (weak x y).
```

On the details of additional rules. For `w_unit`, `s_lpair_eq`, `w_lpair_eq` and `lpair_ueq`, see Figure 5.4. The rule `effect` is given in Figure 5.3. Lastly, for `ax1`, `ax2` and `local_global`, refer back to Figure 5.5.

The derived rule (s-unit), given in Proposition 5.1.2, specializes (w-unit) in the absence of modifiers so that the weak equation converts into the strong one. Below is the Coq certified statement and its proof:

```
Lemma s_unit: forall X (f: term unit X), (RO f) → f == forget.
Proof. intros X f H. apply wtos; [exact H| decorate| apply w_unit]. Qed.
```

Remark 5.2.4. See the source `Axioms.v` for related implementation details.

5.2.5 Derived pairs and products

In order to speak about symmetric or (right) pairs as well as left and right products, we define the permutation term, denoted `permut`. It inputs two Coq `Type` instances `X` and `Y` and outputs an instance of type: `term (Y*X) (X*Y)`.

```
Definition permut {X Y}: term (X*Y) (Y*X) := pair pi2 pi1.
```

Clearly, `permut` is a pure term, since it is a left pair made of pure projections. Now, the right pair structure looks like:

```
Definition rpair {X Y Z} (f1: term Y X) (f2: term Z X): term (Y*Z) X := permut o pair f2 f1.
```

The decoration of a given right pair depends on its components:

```
Lemma is_rpair: forall k X Y Z (f1: term Y X) (f2: term Z X), RO f2 → is k f1 → is k f2
                → is k (rpair f1 f2).
Proof. intros k X Y Z f1 f2 H1 H2 H3. induction k; decorate. Qed.
```

After introducing the necessary instances, we induce on the `kind k`. Then, it suffices to `decorate` each goal: `is pure (rpair f1 f2)`, `is ro (rpair f1 f2)` and `is rw (rpair f1 f2)` locally provided: `H0: is pure f1`, `H1: is pure f2`; `H0: is ro f1`, `H1: is ro f2` and `H0: is rw f1`, `H1: is rw f2`.

The projection rules attached to right pairs, that are stated and proven in Proposition 5.1.5, are certified in Coq along with their proofs:

right pair: first projection

```
Lemma s_rpair_eq: forall X Y' Y (f1: term Y X) (f2: term Y' X), RO f2 → pi1 o rpair f1 f2 == f1.
Proof.
    intros X Y' Y f1 f2 H0. unfold rpair. unfold permut. rewrite assoc.
    cut (pi1 o pair pi2 pi1 == (@pi2 Y' Y)).
      intro H1. rewrite H1.
      apply s_lpair_eq. exact H0.
    apply wtos; try decorate. apply w_lpair_eq; decorate.
Qed.
```

After forming the environment for the assumptions, the proof continues with unfolding `rpair` and `permut` followed by rewriting associativity which shifts parentheses to the left. At this point, the goal looks like: `(pi1 o pair pi2 pi1) o pair f2 f1 == f1`. Here, we cut the `Prop` instance, `(pi1 o pair pi2 pi1 == (@pi2 Y' Y))` and introduce an instance of it named `H1`. We apply `H1` inside the goal and obtain `pi2 o pair f2 f1 == f1`. Now, it suffices to apply the rule `s_lpair_eq` and prove that `RO f2` which is exactly `H0`. It remains to prove the strong equation that we have already cut. There, we first convert the goal side strong equation into the weak equation provided that `pi1 o pair pi2 pi1` and `(@pi2 Y' Y)` are both accessors. So that the goal turns into `(pi1 o pair pi2 pi1 ∼ (@pi2 Y' Y))`. It suffices to apply `w_lpair_eq` and prove that `RO pi2` which is done by the use of tactic `decorate`.

right pair: second projection

```
Lemma w_rpair_eq: forall X Y' Y (f1: term Y X) (f2: term Y' X), RO f2 → pi2 o rpair f1 f2 ∼ f2.
Proof.
    intros X Y' Y f1 f2 H0. unfold rpair. unfold permut. rewrite assoc.
    rewrite s_lpair_eq; [apply w_lpair_eq; decorate | decorate].
Qed.
```

After some preliminary modifications on the goal (by following the first line in the proof), we obtain `(pi2 o pair pi2 pi1) o pair f2 f1 ∼ f2`. Here, we rewrite `s_lpair_eq` which results in two subgoals: `pi1 o pair f2 f1 ∼ f2` and `RO pi2`. The application of the rule `w_lpair_eq` followed by the `decorate` solves the first subgoal and `decorate` alone closes the second.

We also certify the proofs of Propositions 5.1.3 and 5.1.6 ensuring that left and right pairs are unique with respect to the strong equation:

left pair: unicity

```
Lemma lpair_u: forall X Y Y'(f1 f2: term (Y'*Y) X),
    (pi1 o f1 ∼ pi1 o f2) ∧ (pi2 o f1 == pi2 o f2) → f1 == f2.
Proof.
    intros X Y Y' f1 f2 (H0&H1). apply effect.
    (* ⟨ ⟩ ∘ f1 ≡ ⟨ ⟩ ∘ f2 *)
    cut(forget o (@pi2 Y' Y) == forget).
      intro H2. rewrite ←H2.
      setoid_rewrite ←assoc. apply replsubs; [reflexivity | exact H1].
    (* 1st cut *)
    setoid_rewrite s_unit;[reflexivity | decorate].
    (* f1 ∼ f2 *)
    apply lpair_ueq. exact H0. apply stow. exact H1.
Qed.
```

right pair: unicity

```
Lemma rpair_u: forall X Y Y'(f1 f2: term (Y'*Y) X),
    (pi1 o f1 == pi1 o f2) ∧ (pi2 o f1 ∼ pi2 o f2) → f1 == f2.
Proof.
    intros X Y Y' f1 f2 (H0&H1). apply effect.
    (* ⟨ ⟩ ∘ f1 ≡ ⟨ ⟩ ∘ f2 *)
    cut(forget o (@pi1 Y' Y) == forget).
      intro H2. rewrite ←H2.
      setoid_rewrite ←assoc. apply replsubs; [reflexivity | exact H0].
    (* 1st cut *)
    setoid_rewrite s_unit;[reflexivity | decorate].
    (* f1 ∼ f2 *)
    apply lpair_ueq. apply stow. exact H0. exact H1.
Qed.
```

Both proofs follow the same approach: first the (effect) rule is applied to the goal. This generates two subgoals to prove: `forget ∘ f1 ≡ forget ∘ f2` and `f1 ∼ f2`. Then, depending on the assumed context, we use either `forget ∘ pi1 ≡ forget` or `forget ∘ pi2 ≡ forget` (ensured by the rule (s-unit)) to close the first subgoals. For instance, considering the unicity of the right pairs, by rewriting `forget ∘ pi1 ≡ forget` inside the goal, we obtain `forget ∘ pi1 ∘ f1 ≡ forget ∘ pi1 ∘ f2`. By applying (replsubs), we get `pi1 ∘ f1 ≡ pi1 ∘ f2` which is a local assumption. For the second subgoals, we use (lpair-ueq) rule. For instance, for the unicity of the right pairs, applying (lpair-ueq) yields following subgoals: `pi1 ∘ f1 ∼ pi1 ∘ f2` and `pi2 ∘ f1 ≡ pi2 ∘ f2`. The former is an assumption after the free conversion of the weak equation into the strong one provided by the application of rule (stow). The latter is an assumption.

In addition, left and right product structures, that are detailed in Definition 5.1.7, are implemented in Coq as follows:

```
Definition lprod {X Y X' Y'} (f: term X X') (g: term Y Y') := pair (f o pi1) (g o pi2).
Definition rprod {X Y X' Y'} (f: term X X') (g: term Y Y') := permut o pair (g o pi2) (f o pi1).
```

One can simply prove that the decoration of a pair product depends on its components:

```
Lemma is_lprod: forall k X' X Y' Y (f1: term X X') (f2: term Y Y'), R0 f1 → is k f1 → is k f2
                → is k (lprod f1 f2).
Proof. intros k X' X Y' Y f1 f2 H1 H2 H3. induction k; decorate. Qed.
```

After introducing the necessary instances, we induce on the `kind k` then it suffices to `decorate` each goal: `is pure (prod f1 f2)`, `is ro (prod f1 f2)` and `is rw (prod f1 f2)` locally provided: `H0: is pure f1`, `H1: is pure f2`; `H0: is ro f1`, `H1: is ro f2` and `H0: is rw f1`, `H1: is rw f2`. The similar idea applies to the case of right products:

```
Lemma is_rprod: forall k X' X Y' Y (f1: term X X') (f2: term Y Y'), RO f2→ is k f1 → is k f2
                → is k (rprod f1 f2).
Proof. intros k X' X Y' Y f1 f2 H1 H2 H3. induction k; decorate. Qed.
```

The projection rules attached to left and right products, that are stated and proved in Propositions 5.1.8 and 5.1.10, are certified in Coq along with their proofs:

left and right products: first and second projection

```
Lemma w_lprod: forall X' X Y' Y (f: term X' X) (g: term Y' Y), RO f → pi1 o (prod f g) ∼ f o pi1.
Proof. intros X' X Y' Y f g H. apply w_lpair; decorate. Qed.

Lemma s_lprod: forall X' X Y' Y (f: term X' X) (g: term Y' Y), RO f → pi2 o (prod f g) == g o pi2.
Proof. intros X' X Y' Y f g H. apply s_lpair; decorate. Qed.

Lemma w_rprod: forall X' X Y' Y (f: term X' X) (g: term Y' Y), RO g → pi2 o (prod f g) ∼ g o pi2.
Proof. intros X' X Y' Y f g H. apply w_rpair; decorate. Qed.

Lemma s_rprod: forall X' X Y' Y (f: term X' X) (g: term Y' Y), RO g → pi1 o (prod f g) == f o pi1.
Proof. intros X' X Y' Y f g H. apply s_rpair; decorate. Qed.
```

They are nothing but specialized (w-lpair-eq), (s-lpair-eq), (w-rpair-eq) and (s-rpair-eq).

We lastly have the unicity properties of left and right products with respect to the strong equation, that are stated and proven in Propositions 5.1.9 and 5.1.11, certified in Coq:

left and right products: unicity

```
Lemma lprod_u: forall X X' Y Y'(f1 f2: term (Y*Y') (X*X')),
  (pi1 o f1 ∼ pi1 o f2) ∧(pi2 o f1 == pi2 o f2) → f1 == f2.
Proof. intros X X' Y Y' f1 f2 (H0&H1). apply lpair_u. split; [exact H0| exact H1]. Qed.

Lemma rprod_u: forall X X' Y Y'(f1 f2: term (Y*Y') (X*X')),
  (pi1 o f1 == pi1 o f2) ∧(pi2 o f1 ∼ pi2 o f2) → f1 == f2.
Proof. intros X X' Y Y' f1 f2 (H0&H1). apply rpair_u. split; [exact H0| exact H1]. Qed.
```

It suffices to respectively apply (lpair_u) and (rpair_u) to close the goals.

Remark 5.2.5. See the sources `Derived_Pairs.v` and `Derived_Products.v` for related implementation details.

5.3 Proving properties of the state

In [PP02, §3], Plotkin and Power have introduced seven properties of the global state. In addition, we introduce an eighth property which will become useful when we apply this idea of reasoning to a programming language such as `IMP` (or `while`). Here, as an example of use, we provide the decorated versions of these properties together with their proofs in a decorated setting and with the related formalizations in Coq.

(1)$_d$ Annihilation lookup-update. *Reading the value of a location i and then updating the location i with the obtained value is just like doing nothing.* $\forall\, i \in Loc$, $\mathtt{update}_i^{(2)} \circ \mathtt{lookup}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)} : \mathbb{1} \to \mathbb{1}$.

$(2)_d$ Interaction lookup-lookup. *Reading twice the same location i is the same as reading it once.* $\forall\, i \in Loc,\ \mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)} : \mathbb{1} \to V_i$.

$(3)_d$ Interaction update-update. *Storing a value x and then a value x′ at the same location i is just like storing the value x′ in the location.* $\forall\, i \in Loc,\ \mathtt{update}_i^{(2)} \circ \pi_2^{(0)} \circ (\mathtt{update}_i^{(2)} \times_r id_i^{(0)}) \equiv \mathtt{update}_i^{(2)} \circ \pi_2^{(0)} : V_i \times V_i \to \mathbb{1}$.

$(4)_d$ Interaction update-lookup. *When one stores a value x in a location i and then reads the location i, one gets the value x.* $\forall\, i \in Loc,\ \mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \sim id_{V_i}^{(0)} : V_i \to V_i$.

$(5)_d$ Commutation lookup-lookup. *The order of reading two different locations i and j does not matter.* $\forall\, i \neq j \in Loc,\ (id_{V_i}^{(0)} \times_r \mathtt{lookup}_j^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lookup}_i^{(1)} \equiv permut_{j,i}^{(0)} \circ (id_{V_j}^{(0)} \times_r \mathtt{lookup}_i^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lookup}_j^{(1)} : \mathbb{1} \to V_i \times V_j$ where $\pi_1^{-1(0)} := \langle id, \langle\ \rangle \rangle_l^{(0)}$.

$(6)_d$ Commutation update-update. *The order of storing in two different locations i and j does not matter.* $\forall\, i \neq j \in Loc,\ \mathtt{update}_j^{(2)} \circ \pi_2^{(0)} \circ (\mathtt{update}_i^{(2)} \times_r id_{V_j}^{(0)}) \equiv \mathtt{update}_i^{(2)} \circ \pi_1^{(0)} \circ (id_{V_i}^{(0)} \times_l \mathtt{update}_j^{(2)}) : V_i \times V_j \to \mathbb{1}$.

$(7)_d$ Commutation update-lookup. *The order of storing in a location i and reading another location j does not matter.* $\forall\, i \neq j \in Loc,\ \mathtt{lookup}_j^{(1)} \circ \mathtt{update}_i^{(2)} \equiv \pi_2^{(0)} \circ (\mathtt{update}_i^{(2)} \times_r id_{V_j}^{(0)}) \circ (id_{V_i}^{(0)} \times_l \mathtt{lookup}_j^{(1)}) \circ \pi_1^{-1(0)} : V_i \to V_j$.

$(8)_d$ Commutation lookup-constant. *Just after storing a constant c in a location i, observing the content of i is the same as regenerating the constant c.* $\forall\, i \in Loc,\ \forall\, c \in V_i$; $\mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \circ constant\ c^{(0)} \equiv constant\ c^{(0)} \circ \mathtt{update}_i^{(2)} \circ constant\ c^{(0)} : \mathbb{1} \to V_i$.

The decorated logic for the state ($\mathcal{L}_{st}$) is used to prove above the stated properties. Such proofs are enriched with Coq certifications. Within the Coq scripts, one can simply relate the Coq proof to the proof on the paper by observing the comments following crucial steps. Notice also that the use of *associativity of composition* in the Coq proofs just balances the proof tree into an intended shape. This is omitted in the proofs on the paper.

Lemma 5.3.1. Annihilation lookup-update (ALU). *Reading the value of a location i and then updating the location i with the obtained value is just like doing nothing.*

$$\forall i \in Loc, \mathtt{update}_i^{(2)} \circ \mathtt{lookup}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)} \tag{5.1}$$

Proof. (1) Due to (ax_1), we have $\mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \sim id_{V_i}^{(0)}$. By (wsubs), we obtain $\mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \circ \mathtt{lookup}_i^{(1)} \sim id_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)}$. We first throw the identity out by the use of (ids), then (idt) gives $\mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \circ \mathtt{lookup}_i^{(1)} \sim \mathtt{lookup}_i^{(1)} \circ id_{\mathbb{1}}^{(0)}$.

$$\dfrac{\dfrac{\dfrac{(\mathrm{ax}_1)\ \dfrac{\forall\ \mathrm{i} \in \mathrm{Loc}}{\mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \sim id_{V_i}}}{(\mathrm{wsubs})\ \mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \circ \mathtt{lookup}_i^{(1)} \sim id_{V_i} \circ \mathtt{lookup}_i^{(1)}}}{(\mathrm{ids})\ \mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \circ \mathtt{lookup}_i^{(1)} \sim \mathtt{lookup}_i^{(1)}}}{(\mathrm{idt})\ \mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \circ \mathtt{lookup}_i^{(1)} \sim \mathtt{lookup}_i^{(1)} \circ id_{\mathbb{1}}^{(0)}}$$

(2) We have $\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \sim \texttt{lookup}_k^{(1)} \circ \langle\ \rangle_{V_i}^{(0)}$, for each location k such that $k \neq i$, due to (ax_2). We get $\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim \texttt{lookup}_k^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \texttt{lookup}_i^{(1)}$ thanks to (wsubs). Besides, we have $\langle\ \rangle_{V_i}^{(0)} \circ \texttt{lookup}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)}$ using (s-unit). Therefore, we finally have $\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim \texttt{lookup}_k^{(1)} \circ id_{\mathbb{1}}^{(0)}$.

$$\text{(wsubs)}\ \frac{\text{(ax}_2\text{)}\ \dfrac{\forall\ \text{i k} \in \text{Loc s.t. i} \neq \text{k}}{\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \sim \texttt{lookup}_k^{(1)} \circ \langle\ \rangle_{V_i}^{(0)}}}{\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim \texttt{lookup}_k^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \texttt{lookup}_i^{(1)}} \qquad \text{(s-unit)}\ \frac{\vdots}{\texttt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \equiv id_{\mathbb{1}}^{(0)}}$$
$$\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim \texttt{lookup}_k^{(1)} \circ id_{\mathbb{1}}^{(0)}$$

From the items (1) and (2), (local-global) yields $\texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)}$.

□

In addition, one can start with the goal statement itself, continue with manipulations on it and finally end up with a truth value. This is actually constructing the proof tree with a bottom-up strategy. For instance, let us start with applying (local-global) on the above stated goal and continue as follows:

(1) for any location k, when $k = i$, the goal looks like $\texttt{lookup}_i^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim \texttt{lookup}_i^{(1)} \circ id_{\mathbb{1}}^{(0)}$.

(1.1) we apply (ids) and (idt) to obtain $\texttt{lookup}_i^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim id_{V_i}^{(0)} \circ \texttt{lookup}_i^{(1)}$.

(1.2) by applying (wsubs), we get $\texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim id_{V_i}^{(0)}$. Finally, the application of (ax_1) resolves the goal.

(2) when $k \neq i$, the goal becomes $\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim \texttt{lookup}_k^{(1)} \circ id_{\mathbb{1}}^{(0)}$.

(2.1) thanks to (s-unit), we have $\texttt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \equiv id_{\mathbb{1}}^{(0)}$ thus $\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \circ \texttt{lookup}_i^{(1)} \sim \texttt{lookup}_k^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \texttt{lookup}_i^{(1)}$.

(2.2) by applying (wsubs), we get $\texttt{lookup}_k^{(1)} \circ \texttt{update}_i^{(2)} \sim \texttt{lookup}_k^{(1)} \circ \langle\ \rangle_{V_i}^{(0)}$. Finally, the application of (ax_2) resolves the goal.

We express the formalization of the statement (ALU) along with its certified proof in Coq. Note that the proof proceeds by manipulations on the goal statement(s) to end up with some truth value. Therefore, it follows the same lines as the proof given just above. By using the apparent numbering, one can relate the proof steps in English to the ones in Coq.

```
Lemma ALU: forall i: Loc, update i o lookup i == id.
Proof.
    intro i.
    apply eq3. intro k. destruct (Loc_dec i k) as [Ha|Hb]. rewrite Ha.
    (* k = i *) (* (1) *)
    rewrite ids. setoid_rewrite ←idt at 6. rewrite assoc. (* (1.1) *)
    apply wsubs; [apply ax1| reflexivity]. (* (1.2) *)
    (* k <> i *)
    cut((@forget (V i)) o lookup i == (@id unit))·
      [intro H0| setoid_rewrite s_unit; [reflexivity| decorate| decorate]].
    rewrite ←H0. setoid_rewrite assoc. (* (2.1) *)
    apply wsubs; [apply ax2; exact Hb| reflexivity]. (* (2.2) *)
Qed.
```

where `Loc_dec` is a variant of excluded middle ensuring that two locations i and k are either the same or different.

```
Parameter Loc_dec: forall i j: Loc, {i=j}+{i<>j}.
```

Lemma 5.3.2. Interaction lookup-lookup (ILL). *Reading twice the same location i is the same as reading it once.*

$$\forall\, i \in Loc,\ \mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)} : \mathbb{1} \to V_i \tag{5.2}$$

Proof. By (s-unit), we have $\langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)}$. Then, the use of (replsubs) gives $\mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)} \circ id_{\mathbb{1}}^{(0)}$. By (ids), we simply conclude with $\mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)}$

$$\dfrac{\dfrac{\dfrac{\vdots}{\langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)}}\,\text{(s-unit)}}{\mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)} \circ id_{\mathbb{1}}^{(0)}}\,\text{(replsubs)}}{\mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)}}\,\text{(ids)}$$

□

Let us continue with another proof the same statement but this time following the bottom-up strategy: $\mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)}$:

(1) by (ids), we obtain $\mathtt{lookup}_i^{(1)} \circ \langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv \mathtt{lookup}_i^{(1)} \circ id_{\mathbb{1}}^{(0)}$.

(2) we apply (replsubs) and get $\langle\ \rangle_{V_i}^{(0)} \circ \mathtt{lookup}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)}$.

(3) finally, the use of (s-unit) closes the goal.

Below, we give the related formalization of (ILL) in Coq with its certified proof. The proof follows a the bottom-up strategy:

```
Lemma ILL: forall i, lookup i o forget o lookup i == lookup i.
Proof.
    intro i. rewrite ←assoc.
      setoid_rewrite ←ids at 6. (* (1) *)
      apply replsubs; [reflexivity| ]. (* (2) *)
      setoid_rewrite s_unit; [reflexivity| decorate| decorate]. (* (3) *)
Qed.
```

The proofs of the remaining properties can be found in Appendix A. By using them, we can prove program properties with the global state effect.

Remark 5.3.3. See the source `Proofs.v` for related implementation details.

5.4 Hilbert-Post completeness for the state effect

Now, in order to prove the completeness of the decorated theory for the state effect under suitable assumptions, we first determine canonical forms and then we study the equations between terms in such forms [DDE+15].

The logic $\mathcal{L}_{st}$ is precisely introduced and its categorical interpretation is studied in Section 5.1. Let the logic $\mathcal{L}_{st-\otimes}$ be the variant of $\mathcal{L}_{st}$ obtained by dropping the categorical pairs/products. Let the logic $\mathcal{L}_{meq+\mathbb{1}}$ be an extension to $\mathcal{L}_{meq}$ with the use of unit ($\mathbb{1}$) type and the following inference rules: $\frac{X}{\langle\,\rangle_X : X \to \mathbb{1}}$ and $\frac{f : X \to \mathbb{1}}{f \cong \langle\,\rangle_X}$. Now, the core theory of states $\mathcal{T}_{st}$ is defined as a theory of the logic $\mathcal{L}_{st-\otimes}$ generated from the fundamental equation $\mathtt{lookup}_i^{(1)} \circ \mathtt{update}_i^{(2)} \sim id_{V_i}^{(0)}$ and from some consistent theory $\mathcal{T}_{eq}$ of the logic $\mathcal{L}_{meq+\mathbb{1}}$; with the notations of Section 4.5, $\mathcal{T}_{st} = F(\mathcal{T}_{eq})$. In this section, we prove that the theory $\mathcal{T}_{st}$ of the logic $\mathcal{L}_{st-\otimes}$ is Hilbert-Post complete with respect to the logic $\mathcal{L}_{meq+\mathbb{1}}$.

Remark 5.4.1. Note that a Coq certification of the whole Hilbert-Post completeness proof, presented in this section, can be found in the package `hp-thesis`: `https://forge.imag.fr/frs/download.php/696/HPC-THESIS.tar.gz`. Check out the `HPCompletenessCoq.v` file inside the `st-hp` folder. Our main result is Theorem 5.4.9 about the relative Hilbert-Post completeness of the decorated theory $\mathcal{T}_{st}$ of states under suitable assumptions. It is assumed that there is only one location i and we write V, `lookup` and `update` instead of V_i, $\mathtt{lookup}\,\mathtt{i}^{(1)}$ and $\mathtt{update}\,\mathtt{i}^{(2)}$. The study of completeness proof with the signature including several locations and products is considered as a future goal.

Note also that we do not explicitly have the relative Hilbert-Post completeness (rHPC) formalization in Coq. Thanks to the second characterization of rHPC given in Corollary 4.5.10, it suffices to show that any formula e in the logic $\mathcal{L}_{st-\otimes}$ is ($\mathcal{T}_{st}$-)equivalent to some set of formulae E_0 in the logic $\mathcal{L}_{meq+\mathbb{1}}$:

$$\mathcal{T}_{st} + Th(E_0) = \mathcal{T}_{st} + Th(e).$$

This has been checked in Coq.

Lemma 5.4.2. *1. For all pure terms $u_1^{(0)}, u_2^{(0)} : V \to Y$, one has: $u_1^{(0)} \equiv u_2^{(0)} \iff u_1^{(0)} \circ \mathtt{lookup} \equiv u_2^{(0)} \circ \mathtt{lookup} \iff u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \equiv u_2^{(0)} \circ \mathtt{lookup} \circ \mathtt{update}$.*

2. For all pure terms $u^{(0)} : V \to Y$, $v^{(0)} : \mathbb{1} \to Y$, one has: $u^{(0)} \equiv v^{(0)} \circ \langle\,\rangle_V^{(0)} \iff u^{(0)} \circ \mathtt{lookup} \equiv v^{(0)}$.

3. For all modifiers $f_1^{(2)}, f_2^{(2)} : X \to V$, $\mathtt{update} \circ f_1^{(2)} \equiv \mathtt{update} \circ f_2^{(2)} \iff f_1^{(2)} \sim f_2^{(2)}$.

Proof. 1. Implications from left to right are clear. Conversely, $u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \equiv u_2^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \implies u_1^{(0)} \equiv u_2^{(0)}$: after converting the strong equation into a weak equation, the use of axiom (ax_1), since the term $u_1^{(0)}$ is pure, gives $u_1^{(0)} \sim u_2^{(0)}$. Now, neither of u_1 and u_2 are modifiers, so that $u_1^{(0)} \equiv u_2^{(0)}$.

2. First, since $\langle\,\rangle_V^{(0)} \circ \mathtt{lookup} : \mathbb{1} \to \mathbb{1}$ is an accessor we have $\langle\,\rangle_V^{(0)} \circ \mathtt{lookup} \equiv id_{\mathbb{1}}^{(0)}$. Now, if $u^{(0)} \equiv v^{(0)} \circ \langle\,\rangle_V^{(0)}$ then $u^{(0)} \circ \mathtt{lookup} \equiv v^{(0)} \circ \langle\,\rangle_V^{(0)} \circ \mathtt{lookup}$, so that $u^{(0)} \circ$

$\texttt{lookup} \equiv v^{(0)}$. Conversely, if $u^{(0)} \circ \texttt{lookup} \equiv v^{(0)}$ then $u^{(0)} \circ \texttt{lookup} \equiv v^{(0)} \circ \langle\,\rangle_V^{(0)} \circ \texttt{lookup}$, and by Point (1) this means that $u^{(0)} \equiv v^{(0)} \circ \langle\,\rangle_V^{(0)}$.

3. Assuming $\texttt{update} \circ f_1^{(2)} \equiv \texttt{update} \circ f_2^{(2)}$, we get $\texttt{lookup} \circ \texttt{update} \circ f_1^{(2)} \equiv \texttt{lookup} \circ \texttt{update} \circ f_2^{(2)}$, thanks to (replsubs). Now, we convert the strong equation into a weak one and apply (ax_1) on both sides so as to obtain $f_1^{(2)} \sim f_2^{(2)}$. Conversely, if $f_1^{(2)} \sim f_2^{(2)}$, by rewriting (wsubs), we obtain $id_V^{(0)} \circ f_1^{(2)} \sim id_V^{(0)} \circ f_2^{(2)}$. We then apply ($\text{ax}_2$) and get $\texttt{lookup} \circ \texttt{update} \circ f_1^{(2)} \sim \texttt{lookup} \circ \texttt{update} \circ f_2^{(2)}$. Since we consider a single location, (local-global) gives $\texttt{update} \circ f_1^{(2)} \equiv \texttt{update} \circ f_2^{(2)}$.

□

Proposition 5.4.3. *1. For each accessor $a^{(1)} : X \to Y$, either a is pure or there is a pure term $u^{(0)} : V \to Y$ and an accessor $v^{(1)} : X \to \mathbb{1}$ such that $a^{(1)} \equiv u^{(0)} \circ \texttt{lookup}^{(1)} \circ v^{(1)}$.*

2. For each modifier $f^{(2)} : X \to Y$, either f is an accessor or there is an accessor $a^{(1)} : X \to V$ and a pure term $u^{(0)} : V \to Y$ such that $f^{(2)} \equiv u^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a^{(1)}$.

Proof. 1. The proof proceeds by structural induction. If a is pure, then the result is obvious. If $a = \texttt{lookup}$, then it follows that $\texttt{lookup} \equiv id_V^{(0)} \circ \texttt{lookup} \circ id_{\mathbb{1}}^{(1)}$. Otherwise, a can be written as $a = a_1^{(1)} \circ a_2^{(1)}$ such that $f_1 : Z \to Y$ and $a_2 : X \to Z$. By induction, a_1 and a_2 are either pure or $a_1 \equiv u_1 \circ \texttt{lookup} \circ v_1$ and $a_2 \equiv u_2 \circ \texttt{lookup} \circ v_2$ for some pure terms $u_1^{(0)} : V \to Y$, $u_2^{(0)} : V \to Z$ and some accessors $v_1^{(1)} : Z \to \mathbb{1}$, $v_2^{(1)} : X \to \mathbb{1}$. So, there are four cases to consider.

(1.1) If both a_1 and a_2 are pure, then a is.

(1.2) If a_1 is pure while a_2 is an accessor, we get $f \equiv (f_1 \circ u_2)^{(0)} \circ \texttt{lookup} \circ v_2^{(1)}$.

(1.3) Symmetrically when a_2 is pure while a_1 is an accessor, we get $f \equiv u_1^{(0)} \circ \texttt{lookup} \circ (v_1 \circ a_2)^{(1)}$.

(1.4) If both are accessors, then $f \equiv u_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ v_2^{(1)}$. We have $v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \equiv id_{\mathbb{1}}^{(0)}$, thanks to (s-unit). The use of (replsubs) yields $u_1^{(0)} \circ \texttt{lookup}^{(1)} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup}^{(1)} \circ v_2^{(0)} \equiv u_1^{(0)} \circ \texttt{lookup}^{(1)} \circ v_2^{(0)}$. Hence, we obtain $f \equiv u_1^{(0)} \circ \texttt{lookup}^{(1)} \circ v_2^{(0)}$.

2. The proof proceeds by structural induction. If f is an accessor, then the result is obvious. If $f = \texttt{update}$, then it follows that $\texttt{update} \equiv \langle\,\rangle_V^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ id_V^{(1)}$ (notice that $\langle\,\rangle_V^{(0)} \circ \texttt{lookup} \equiv id_{\mathbb{1}}^{(0)}$ due to (s-unit)). Otherwise, f can be written in a way as $f = f_1^{(2)} \circ f_2^{(2)}$ such that $f_1 : Z \to Y$ and $f_2 : X \to Z$. By induction, either f_1 and f_2 are accessors or $f_1 \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)}$ and $f_2 \equiv u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$ for some pure term u_1, u_2 and some accessors a_1, a_2. So, there are four cases to consider.

(2.1) If both f_1 and f_2 are accessors, then f is also an accessor.

(2.2) if f_2 is a modifier whilst f_1 is an accessor, we obtain $f \equiv f_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$. Thanks to Point 1, $f_1 \equiv w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)}$ for some pure term $w_1^{(0)} : V \to Y$ and some accessor $v_1^{(1)} : Z \to \mathbb{1}$. Thus, the equation expands

into $f \equiv w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$. Due to (s-unit), we have $v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \equiv id_{\mathbb{1}}^{(0)}$. Thanks to (replsubs), we end up with $w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)} \equiv w_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$. Thus, $f \equiv w_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$.

(2.3) Symmetrically when f_1 is a modifier while f_2 is an accessor, we get $f \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ (a_1 \circ f_2)^{(1)}$.

(2.4) If both are modifiers, then $f \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$, such that $u_1^{(0)} : V \to Y$, $a_1^{(1)} : Z \to V$, $u_2^{(0)} : V \to Z$ and $a_2^{(1)} : X \to V$. Here, the reasoning proceeds on $a_1^{(1)}$. Therefore, we have two subcases:

(2.4.1) Let us first consider the case where a_1 is pure. Since $a_1^{(0)} \circ u_2^{(0)}$ is pure, from (pwrepl), we have $a_1^{(0)} \circ u_2^{(0)} \sim a_1^{(0)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update}$. Now, we apply Point 3 in Lemma 5.4.2 and get $\texttt{update}^{(2)} \circ a_1^{(0)} \circ u_2^{(0)} \circ \equiv \texttt{update}^{(2)} \circ a_1^{(0)} \circ u_2^{(0)} \circ \texttt{lookup}^{(1)} \circ \texttt{update}^{(2)}$. By (replsubs), we obtain $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(0)} \circ u_2^{(0)} \circ a_2^{(1)} \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(0)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$. Hence $f \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ (a_1 \circ u_2 \circ a_2)^{(1)}$.

(2.4.2) Let us also consider the case where a_1 is a non-pure accesor (it has lookup). Thanks to Point 1, we obtain $a_1^{(1)} \equiv w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)}$ such that $w_1^{(0)} : V \to V$ and $v_1^{(1)} : Z \to \mathbb{1}$. So that $f \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$. Due to (s-unit), $v_1^{(1)} \circ u_2^{(0)} \equiv \langle\ \rangle_V$. By (replsubs), $\texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \equiv \texttt{lookup} \circ \langle\ \rangle_V \circ \texttt{lookup}$. Thanks to Lemma 5.3.2, we obtain $\texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \equiv \texttt{lookup}$. By (replsubs), $w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \equiv w_1^{(0)} \circ \texttt{lookup} \circ \texttt{update}$. Here, we first convert the strong equation into a weak equation and then make use of (ax_1) on the right (since w_1 is pure) which gives $w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \sim w_1^{(0)}$. Now, applying Point 3 in Lemma 5.4.2 gives $\texttt{update}^{(2)} \circ w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \equiv \texttt{update}^{(2)} \circ w_1^{(0)}$. Thanks to (replsubs), we end up with $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ w_1^{(0)} \circ \texttt{lookup} \circ v_1^{(1)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)} \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ w_1^{(0)} \circ a_2^{(1)}$. Therefore, $f \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ (w_1 \circ a_2)^{(1)}$.

□

Corollary 5.4.4. *For each accessor $a^{(1)} : X \to Y$, either a is pure or there is a pure term $v^{(0)} : V \to Y$ such that $a^{(1)} \equiv v^{(0)} \circ \texttt{lookup}^{(1)} \circ \langle\ \rangle_X^{(0)}$.*

Proof. If the accessor $a^{(1)} : X \to Y$ is not pure, then it can be written in a unique way as $a^{(1)} = v^{(0)} \circ \texttt{lookup} \circ b^{(1)}$ for some pure term $v^{(0)} : V \to Y$ and some accessor $b^{(1)} : X \to \mathbb{1}$, thanks to the Point 1 in 5.4.3. Due to (s-unit), we have $b^{(1)} \equiv \langle\ \rangle_X^{(0)}$, then the result follows. □

Thanks to Propositions 5.4.3, in order to study equations in the logic $\mathcal{L}_{st}$ we may restrict our study to pure terms, accessors of the form $v^{(0)} \circ \texttt{lookup} \circ \langle\ \rangle_X^{(0)}$ and modifiers of the

form $u^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a^{(1)}$.

Now, Proposition 5.4.5 shows that

(1) equations between modifiers can be reduced to some equations between accessors,

(2) equations between accessors can be reduced to some equations between pure terms.

Proposition 5.4.5. *1. For all $a_1^{(1)}, a_2^{(1)} : X \to V$ and $u_1^{(0)}, u_2^{(0)} : V \to Y$, let $f_1^{(2)} = u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} : X \to Y$ and $f_2^{(2)} = u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)} : X \to Y$. Then*

$$\begin{cases} f_1 \sim f_2 \iff u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)} \\ f_1 \equiv f_2 \iff a_1^{(1)} \equiv a_2^{(1)} \text{ and } u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)} \end{cases}$$

2. For all $a_1^{(1)} : X \to V$, $u_1^{(0)} : V \to Y$ and $a_2^{(1)} : X \to Y$, let $f_1^{(2)} = u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} : X \to Y$. Then

$$\begin{cases} f_1 \sim a_2^{(1)} \iff u_1^{(0)} \circ a_1^{(1)} \equiv a_2^{(1)} \\ f_1 \equiv a_2^{(1)} \iff u_1^{(0)} \circ a_1^{(1)} \equiv a_2^{(1)} \text{ and } a_1^{(1)} \equiv \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \end{cases}$$

3. Let us assume that $\langle\,\rangle_X^{(0)}$ is an epimorphism with respect to accessors. For all $v_1^{(0)}, v_2^{(0)} : V \to Y$ let $a_1^{(1)} = v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} : X \to Y$ and $a_2^{(1)} = v_2^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} : X \to Y$. Then

$$a_1^{(1)} \equiv a_2^{(1)} \iff v_1^{(0)} \equiv v_2^{(0)}$$

4. Let us assume that $\langle\,\rangle_V^{(0)}$ is an epimorphism with respect to accessors and that there exists a pure term $k_X^{(0)} : \mathbb{1} \to X$. For all $v_1^{(0)} : V \to Y$ and $v_2^{(0)} : X \to Y$, let $a_1^{(1)} = v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} : X \to Y$. Then

$$a_1^{(1)} \equiv v_2^{(0)} \iff v_1^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_V^{(0)} \text{ and } v_2^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_X^{(0)}$$

Proof. 1. We have four implications to show:

(1.1) $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \sim u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \implies u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)}$: By ($\text{ax}_1$) and (wprepl), since $u_1^{(0)}$ and $u_2^{(0)}$ are both pure, we obtain $u_1^{(0)} \circ a_1^{(1)} \sim u_2^{(0)} \circ a_2^{(1)}$. Due to the lack of modifiers, we end up with $u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)}$.

(1.2) $u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)} \implies u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \sim u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)}$: We first convert the strong equation into a weak equation. Then, due to (ids) we get $u_1^{(0)} \circ id^{(0)} \circ a_1^{(1)} \sim u_2^{(0)} \circ id^{(0)} \circ a_2^{(1)}$. By rewriting ($\text{ax}_1$) on both sides (since u_1 is a pure term), we get $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \sim u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$.

(1.3) $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)} \implies a_1^{(1)} \equiv a_2^{(1)}$ and $u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)}$:

(1.3.1) Given $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$, we get $\langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv \langle\ \rangle_Y^{(0)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$ thanks to (replsubs) rule. Thanks to (s-unit), we have $\langle\ \rangle_Y^{(0)} \circ u_i^{(0)} \circ \texttt{lookup} \equiv id_{\mathbb{1}}^{(0)}$, for each $i \in \{1,2\}$. Therefore, we obtain $\texttt{update} \circ a_1^{(1)} \equiv \texttt{update} \circ a_2^{(1)}$. Now, by applying Point 3 in Lemma 5.4.2, we have $a_1^{(1)} \sim a_2^{(1)}$. The lack of modifiers yields $a_1^{(1)} \equiv a_2^{(1)}$.

(1.3.2) First, we convert the strong equation into a weak equation and apply (ax_1) on both sides so as to obtain $u_1^{(0)} \circ a_1^{(1)} \sim u_2^{(0)} \circ a_2^{(1)}$. Since there is no modifiers involved, we conclude with $u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)}$.

(1.4) $a_1^{(1)} \equiv a_2^{(1)}$ and $u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)} \implies u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$. We show in below two steps that they have the same effect and the same result:

(1.4.1) Starting from $a_1^{(1)} \equiv a_2^{(1)}$, we get $id_{\mathbb{1}}^{(0)} \circ \texttt{update} \circ a_1^{(1)} \equiv id_{\mathbb{1}}^{(0)} \circ \texttt{update} \circ a_2^{(1)}$ thanks to (replsubs). Due to (s-unit), we have $\langle\ \rangle_Y^{(0)} \circ u_i^{(0)} \circ \texttt{lookup} \equiv \langle\ \rangle_{\mathbb{1}}^{(0)} \equiv id_{\mathbb{1}}^{(0)}$ for each $i \in \{1,2\}$. Therefore, we obtain $\langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv \langle\ \rangle_Y^{(0)} \circ u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$.

(1.4.2) Starting from $u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)}$, we have $u_1^{(0)} \circ id_V^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ id_V^{(0)} \circ a_2^{(1)}$ thanks to (ids). Here, we first convert the strong equation into the weak equation and then apply (ax_1) on both sides (provided that $u_1^{(0)}$ and $u_2^{(0)}$ are pure) so as to obtain $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \sim u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$.

Now, the (effect) rule yields $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$ given above items (1.4.1) and (1.4.2).

2. Here, we again have four cases to prove:

(2.1) $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \sim a_2^{(1)} \implies a_2^{(1)} \equiv u_1^{(0)} \circ a_1^{(1)}$: By ($\text{ax}_1$) and the fact that u_1 is a pure term, we get $u_1^{(0)} \circ a_1^{(1)} \sim a_2^{(1)}$. The lack of modifiers gives $u_1^{(0)} \circ a_1^{(1)} \equiv a_2^{(1)}$.

(2.2) $a_2^{(1)} \equiv u_1^{(0)} \circ a_1^{(1)} \implies u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \sim a_2^{(1)}$: We start with converting the strong equation into a weak equation: $a_2^{(1)} \sim u_1^{(0)} \circ a_1^{(1)}$. Thanks to (ids), we get $a_2^{(1)} \sim u_1^{(0)} \circ id^{(0)} \circ a_1^{(1)}$. By ($\text{ax}_1$), we obtain $a_2^{(1)} \sim u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)}$.

(2.3) $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv a_2^{(1)} \implies \texttt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv a_1^{(1)}$ and $u_1^{(0)} \circ a_1^{(1)} \equiv a_2^{(1)}$:

(2.3.1) Given $u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv a_2^{(1)}$. we can have $\langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv \langle\ \rangle_Y^{(0)} \circ a_2^{(1)}$ thanks to (replsubs). Due to (s-unit), we have $\langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \texttt{lookup} \equiv id_{\mathbb{1}}$. Therefore, $\texttt{update} \circ a_1^{(1)} \equiv \langle\ \rangle_Y^{(0)} \circ a_2^{(1)}$. Again by (s-unit), we have $\langle\ \rangle_Y^{(0)} \circ a_2^{(1)} \equiv \langle\ \rangle_X^{(0)}$. Hence, $\texttt{update} \circ a_1^{(1)} \equiv \langle\ \rangle_X^{(0)}$. The (replsubs) gives $\texttt{lookup} \circ \texttt{update} \circ a_1^{(1)} \equiv \texttt{lookup} \circ \langle\ \rangle_X^{(0)}$. Now, we convert the strong equation into a weak equation and then apply

(ax_1) so as to obtain $a_1^{(1)} \sim \mathtt{lookup} \circ \langle\ \rangle_X^{(0)}$. The lack of modifiers yields $a_1^{(1)} \equiv \mathtt{lookup} \circ \langle\ \rangle_X^{(0)}$.

(2.3.2) We get $u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1^{(1)} \sim a_2^{(1)}$ by converting the strong equation into a weak equation. On the left, we can apply (ax_1), since u_1 is a pure term, and get $u_1^{(0)} \circ a_1^{(1)} \sim a_2^{(1)}$. The lack of modifiers yields $u_1^{(0)} \circ a_1^{(1)} \equiv a_2^{(1)}$.

(2.4) $\mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv a_1^{(1)}$ and $u_1^{(0)} \circ a_1^{(1)} \equiv a_2^{(1)} \implies u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1^{(1)} \equiv a_2^{(1)}$:

(2.4.1) Starting from $\mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv a_1^{(1)}$, due to (replsubs), we obtain $\langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv \langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1^{(1)}$. Since Lemma 5.3.1 states that $\mathtt{update} \circ \mathtt{lookup} \equiv id_V^{(0)}$, we get $\langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv \langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1^{(1)}$. Besides, we have $\langle\ \rangle_Y^{(0)} \circ u_1^{(0)} \circ \mathtt{lookup} \equiv id_1^{(0)}$, thanks to (s-unit). Thus, $\langle\ \rangle_X^{(0)} \equiv \mathtt{update} \circ a_1^{(1)}$.

(2.4.2) Given $u_1^{(0)} \circ a_1^{(1)} \equiv a_2^{(1)}$, we first convert the strong equation into a weak equation and then apply (ids) so as to get $u_1^{(0)} \circ id_V^{(0)} \circ a_1^{(1)} \sim a_2^{(1)}$. Since, u_1 is a pure term, we can apply (ax_1) and obtain $u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1^{(1)} \sim a_2^{(1)}$. The above point (2.4.1) gives $u_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \sim a_2^{(1)}$. Now, the lack of modifiers yields $u_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv a_2^{(1)}$. Again due to the above point (2.4.1), we end up with $u_1^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1^{(1)} \equiv a_2^{(1)}$.

3. We want to show that $a_1^{(1)} \equiv a_2^{(1)} \iff v_1^{(0)} \equiv v_2^{(0)}$:

(3.1) Let us start with $v_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv v_2 \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \implies v_1^{(0)} \equiv v_2^{(0)}$. Since $\langle\ \rangle_X^{(0)}$ is an epimorphism with respect to accessors, we get $v_1^{(0)} \circ \mathtt{lookup} \equiv v_2^{(0)} \circ \mathtt{lookup}$. By Point 1 in Lemma 5.4.2, we end up with $v_1^{(0)} \equiv v_2^{(0)}$.

(3.2) Conversely, if $v_1^{(0)} \equiv v_2^{(0)}$ then $v_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv v_2 \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)}$ due to (replsubs).

4. Now for $a_1^{(1)} \equiv v_2^{(0)} \iff v_1^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_V^{(0)}$ and $v_2^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_X^{(0)}$:

(4.1) Let us first consider the left to right implication.

(4.1.1) $v_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv v_2^{(0)}$: Since $v_1^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_V^{(0)}$, we get $v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_V^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv v_2^{(0)}$. Due to (s-unit), we have $\langle\ \rangle_V^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv \langle\ \rangle_X^{(0)}$. Therefore, $v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_X^{(0)} \equiv v_2^{(0)}$.

(4.1.2) $v_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv v_2^{(0)}$: We get $v_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv v_2^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_X^{(0)}$, thanks to above item (4.1.2). Since $\langle\ \rangle_V^{(0)}$ is an epimorphism with respect to accessors, we have $v_1^{(0)} \circ \mathtt{lookup} \equiv v_2^{(0)} \circ k_X^{(0)}$. Now, Point 2 in Lemma 5.4.2 yields $v_1^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_V^{(0)}$.

(4.2) Conversely, $v_1^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_V^{(0)}$ and $v_2^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\ \rangle_X^{(0)} \implies u_1^{(0)} \circ \mathtt{lookup} \circ \langle\ \rangle_X^{(0)} \equiv v_2^{(0)}$:

(4.2.1) Starting from $v_1^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_V^{(0)}$, we get $v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_V^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)}$. We have $\langle\,\rangle_V^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \equiv \langle\,\rangle_X^{(0)}$ thanks to (s-unit). Therefore, $v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_X^{(0)}$. Now, given $v_2^{(0)} \equiv v_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_X^{(0)}$, we end up with $v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \equiv v_2^{(0)}$.

□

Definition 5.4.6. A type X is *inhabited* if there exists a pure term $k_X^{(0)} : \mathbb{1} \to X$. A type $\mathbb{O}$ is *empty* if for each type Y there is a pure term $[\,]_Y^{(0)} : \mathbb{O} \to Y$, and every term $f^{(2)} : \mathbb{O} \to Y$ is such that $f^{(2)} \equiv [\,]_Y^{(0)}$.

Remark 5.4.7. When X is inhabited then for any $k_X^{(0)} : \mathbb{1} \to X$ we have $\langle\,\rangle_X^{(0)} \circ k_X^{(0)} \equiv id_{\mathbb{1}}^{(0)}$, so that $\langle\,\rangle_X^{(0)}$ is a split epimorphism; it follows that $\langle\,\rangle_X^{(0)}$ is an epimorphism with respect to all terms, and especially with respect to accessors.
Now, Corollary 5.4.8 shows that equations between modifiers can be reduced to equations between pure terms. It also makes the proof in Coq easier to read.

Corollary 5.4.8. *Let us assume that $\langle\,\rangle_X^{(0)}$ is an epimorphism with respect to accessors. Then:*

1. *For all $f_1^{(1)}, f_2^{(1)} : X \to Y$, we have one of the following cases:*
   - *(a)* $\exists a_1^{(0)},\ a_2^{(0)} : V \to Y,\ \exists b_1^{(0)},\ b_2^{(0)} : X \to Y,$
     $f_1^{(1)} \equiv f_2^{(1)} \iff a_1^{(0)} \equiv a_2^{(0)}$ *and* $b_1^{(0)} \equiv b_2^{(0)}$,
   - *(b)* $\exists a_1^{(0)}\ a_2^{(0)} : V \to Y,\ f_1^{(1)} \equiv f_2^{(1)} \iff a_1^{(0)} \equiv a_2^{(0)}$,
   - *(c)* $\exists a_1^{(0)}\ a_2^{(0)} : X \to Y,\ f_1^{(1)} \equiv f_2^{(1)} \iff a_1^{(0)} \equiv a_2^{(0)}$
2. *For all $f_1^{(2)}, f_2^{(2)} : X \to Y$, we have one of the following cases:*
   - *(a)* $\exists a_1^{(1)},\ a_2^{(1)} : X \to V,\ \exists b_1^{(1)},\ b_2^{(1)} : X \to Y$
     $f_1^{(2)} \equiv f_2^{(2)} \iff a_1^{(1)} \equiv a_2^{(1)}$ *and* $b_1^{(1)} \equiv b_2^{(1)}$,
   - *(b)* $\exists a_1^{(1)}\ a_2^{(1)} : X \to Y,\ f_1^{(2)} \equiv f_2^{(2)} \iff a_1^{(1)} \equiv a_2^{(1)}$

Proof. The proof is immediate from Proposition 5.4.5. See full proof in Appendix B. □

Theorem 5.4.9. *If every non-empty type is inhabited and if V is non-empty, the theory of states $\mathcal{T}_{st}$ of the logic $\mathcal{L}_{st-\otimes}$ is relatively Hilbert-Post complete with respect to the pure sublogic $\mathcal{L}_{meq+\mathbb{1}}$.*

Proof. The proof relies upon Corollary 5.4.8. The theory $\mathcal{T}_{st}$ is consistent: it cannot be proved that $\texttt{update}^{(2)} \equiv \langle\,\rangle_V^{(0)}$ because the logic $\mathcal{L}_{st}$ is sound with respect to its intended model and the interpretation of this equation in the intended model is false as soon as V has at least two elements: indeed, for each state s and each $x \in V$, $\texttt{lookup} \circ \texttt{update}(x, s) = x$ because of (ax_1) while $\texttt{lookup} \circ \langle\,\rangle_V^{(0)}(x, s) = \texttt{lookup}(s)$ does not depend on x. Let us consider an equation (strong or weak) between terms with domain X in $\mathcal{L}_{st}$; we distinguish two cases, whether X is empty or not. When X is empty, then all terms from X to Y are strongly equivalent to $[\,]_Y^{(0)}$, so that the given equation is equivalent to the empty set of equations between pure terms. When X is non-empty then it is inhabited. Thanks to Remark 5.4.7, we have that $\langle\,\rangle_X^{(0)}$ is an epimorphism with respect to accessors. Thus, Corollary 5.4.8 proves that the given equation is equivalent to a finite set of equations between pure terms. Thus, in both cases, the result follows from Corollary 4.5.10. □

The case distinction in Theorem 5.4.9 comes from the fact that the existence of a pure term $k_X^{(0)} : \mathbb{1} \to X$, which is used in Point 4 of Proposition 5.4.5, is incompatible with the intended model of states if X is interpreted as the empty set.

Remark 5.4.10. This can be generalized to an arbitrary number of locations. The logic $\mathcal{L}_{st}$ and the theory $\mathcal{T}_{st}$ have to be generalized as in [DDFR12a], then Proposition 5.4.3 has to be adapted using the basic properties of `lookup` and `update`, as stated in [PP02]; these properties can be deduced from the decorated theory for states, as proved in [DDEP14]. The rest of the proof generalizes accordingly, as in [Pre10].

Remark 5.4.11. See the source `HPCompleteCoq.v` inside `st-hp` folder for related implementation details.

5.5 Chapter summary

In this Chapter;

(1) The logic $\mathcal{L}_{st}$ has been built as an extension to the logic $\mathcal{L}_{com}$ and interpreted via the *Kleisli-on-coKleisli construction* applied to the states comonad.

(2) The logic $\mathcal{L}_{st}$ has been formalized in Coq. This formalization has been used to prove and certify primitive properties of the programs with the state effect.

(3) The base language of the logic $\mathcal{L}_{st}$ (with no use of products) has been proved to be Hilbert-Post complete (for a single location) and this proof has been checked in Coq.

6

The exceptions effect

Exception handling is provided by most modern programming languages. It allows to deal with anomalous or exceptional events which require special processing. This brings a flexibility into the coding but in order to prove the correctness of such programs one has to take into account the interactions with exceptions. In this chapter, each interaction with exceptions is treated as a computational effect: a term $f : X \to Y$ is not interpreted as a function $f : X \to Y$ unless it is pure. Indeed, a term which may raise an exception has the interpretation $f : X \to Y + E$, while a term which may catch an exception is interpreted as $f : X + E \to Y + E$, where "+" is the disjoint union operator and E is the set of exceptions. In this chapter, we introduce the *decorated logic for programmers' language for exceptions* ($\mathcal{L}_{exc-pl}$) and the *decorated logic for exceptions* ($\mathcal{L}_{exc}$). The logic $\mathcal{L}_{exc-pl}$ aims to model the treatment of exception handling as in modern programming languages such as Java [GJSB05, Ch. 14] and C++ [Dra12, §15], in a decorated setting. It uses decorations only on terms to classify according to their behaviors with respect to exceptions: a term is either pure or a propagator. This logic has two distinguished constructs: `throw` to raise exceptions and `try/catch` to handle them. The logic $\mathcal{L}_{exc}$ is built dually to the logic $\mathcal{L}_{st}$ formalizing the state effect [DDFR12b]. Thus, we obtain the decorations of the logic $\mathcal{L}_{exc}$, for terms and equations, from the logic $\mathcal{L}_{mon}$ and we introduce the interface functions `tag` and `untag` for raising and catching exceptions, respectively. Furthermore, we use a decorated version of categorical coproducts in order to deal with the case distinction which is encapsulated in the handling of exceptions. In addition, we provide a translation of the logic $\mathcal{L}_{exc-pl}$ into the logic $\mathcal{L}_{exc}$ and prove that the rules are correct with respect to $\mathcal{L}_{exc}$.

In Figure 6.1, we instantiate the monad T in Figure 4.4 with the monad of exceptions:

$\mathcal{L}_{exc}$		Interpretation of $\mathcal{L}_{exc}$
catcher	$f^{(2)} : X \to Y$	$f : X + E \to Y + E$
propagator/thrower	$f^{(1)} : X \to Y$	$f : X \to Y + E$
pure term	$f^{(0)} : X \to Y$	$f : X \to Y$
strong equation	$f^{(2)} \equiv g^{(2)} : X \to Y$	$f = g : X + E \to Y + E$
weak equation	$f^{(2)} \sim g^{(2)} : X \to Y$	$f \circ inl_{X,E} = g \circ inl_{X,E} : X \to Y + E$
		where $inl_{X,E} : X \to Y + E$ is the left coprojection

Figure 6.1: The decorated logic $\mathcal{L}_{exc}$ and its interpretation: an overview.

Note that, in this chapter, the keywords *thrower* and *propagator* are interchangeably used. The former indicates the terms that are allowed to throw exceptions and the latter indicates the ones that must propagate the already thrown exceptions. Both are interpreted

in the same way (Figure 6.1).
In Figure 6.2, we instantiate the monad T in Figure 4.4 with the monad of exceptions but we exclude catchers and weak equations:

$\mathcal{L}_{exc-pl}$		Interpretation of $\mathcal{L}_{exc-pl}$
propagator/thrower	$f^{(1)} : X \to Y$	$f : X \to Y + E$
pure term	$f^{(0)} : X \to Y$	$f : X \to Y$
strong equation	$f^{(1)} \equiv g^{(1)} : X \to Y$	$f = g : X \to Y + E$

Figure 6.2: The decorated logic $\mathcal{L}_{exc-pl}$ and its interpretation: an overview.

We start, in Section 6.1, with the *decorated logic for the exception* ($\mathcal{L}_{exc}$) with its interpretation given through the coKleisli-on-Kleisli construction associated to the exceptions monad. We present the *decorated logic for the programmers' language for exceptions* ($\mathcal{L}_{exc-pl}$) in Section 6.2, with its interpretation via the Kleisli adjunction associated to the exceptions monad. The translation of the logic $\mathcal{L}_{exc-pl}$ into the logic $\mathcal{L}_{exc}$ is given in Section 6.3. The Coq implementations of the logics $\mathcal{L}_{exc}$ and $\mathcal{L}_{exc-pl}$ and the translation of the logic $\mathcal{L}_{exc-pl}$ into the logic $\mathcal{L}_{exc}$ are respectively presented in Sections 6.4, 6.5 and 6.6. We prove some properties of the exceptions effect in a decorated setting in Section 6.7. The logic $\mathcal{L}_{exc-pl}$, as well as the logic $\mathcal{L}_{exc}$ without coproducts, are proven to be *relatively Hilbert-Post complete* in Sections 6.8 and 6.9.

6.1 The decorated logic for exceptions

The decorated logic for exceptions ($\mathcal{L}_{exc}$) extends the decorated logic for a monad ($\mathcal{L}_{mon}$) with the sum types (sorts), the empty type $\mathbb{O}$ and the type EV_e of parameters for each exception name $e \in EName$ where $EName$ is a finite set. Similar to the terms (operations) of the logic $\mathcal{L}_{mon}$, each term in $\mathcal{L}_{exc}$ has a source and a target type. Additionally, there is a (left) copair term $[f \mid g]_l : X_1 + X_2 \to Y$ for each couple of terms $f : X_1 \to Y$ and $g : X_2 \to Y$. For each sum type $X+Y$, there are canonical inclusions $inl : X \to X+Y$ and $inr : Y \to X+Y$. The symbol $[\,]_X$ denotes the unique term from the empty type $\mathbb{O}$ to X for each type X. The term $\mathtt{tag}_e : EV_e \to \mathbb{O}$ stands to encapsulate an ordinary parameter with an exception of name e while the term $\mathtt{untag}_e : \mathbb{O} \to EV_e$ is used to recover the parameter. The "$\downarrow$" symbol denotes the *downcast* term that takes as input a term and prevents it from catching exceptions (Section 6.1.5). We give the syntax of $\mathcal{L}_{exc}$ in Figure 6.3 and its inference rules in Figures 6.4, 6.5, 6.6 and 6.7, in addition to the ones stated in Figure 4.5.

Grammar of the decorated logic for the exception: ($e \in EName$)

Types:	$\mathtt{t}, \mathtt{s}$	::=	$\mathtt{X} \mid \mathtt{Y} \mid \cdots \mid \mathtt{t}+\mathtt{s} \mid \mathbb{O} \mid \mathtt{EV}_e$
Terms:	$\mathtt{f}, \mathtt{g}$	::=	$\mathtt{id}_\mathtt{t} \mid \mathtt{a} \mid \mathtt{b} \mid \cdots \mid \mathtt{g} \circ \mathtt{f} \mid [\mathtt{f} \mid \mathtt{g}]_\mathtt{l} \mid \mathtt{inl} \mid \mathtt{inr} \mid [\,]_\mathtt{t} \mid \mathtt{tag}_e \mid \mathtt{untag}_e \mid \downarrow\mathtt{f}$
Decoration for terms:	$(\mathtt{d})$	::=	$(0) \mid (1) \mid (2)$
Equations:	$\mathtt{e}$	::=	$\mathtt{f} \equiv \mathtt{g} \mid \mathtt{f} \sim \mathtt{g}$

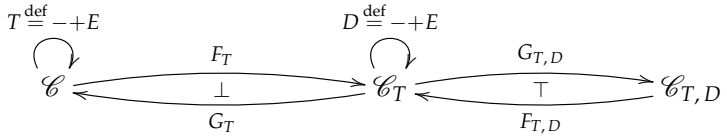
Figure 6.3: $\mathcal{L}_{exc}$: syntax

Each term has a decoration which is denoted as a superscript (0), (1) or (2): a *pure* term has the decoration (0), a *propagator* (or *thrower*) has (1) and a *catcher* term comes with the decoration (2). Similarly, each equation is formed by two terms with the same source and target as well as a decoration: denoted by "$\sim$" if it is *weak* or by "$\equiv$" if it is *strong*.

Let $\mathscr{C}$ be a category with finite coproducts and a distinguished object of exceptions E. Let $(T = - + E, \eta, \mu)$ be the exceptions monad defined over $\mathscr{C}$. Let us assume that E is such that the mono-requirement is satisfied (Definition 3.1.5). For instance, this property is always satisfied when $\mathscr{C}$ is the category of sets.

The interpretation of $\mathcal{L}_{exc}$ is given via the coKleisli-on-Kleisli construction associated to a monad (detailed in Section 3.2) applied to the exceptions monad. Recall that in Section 3.2.2, we have introduced the adjunctions $F_T \dashv G_T$ and $F_{T,D} \dashv G_{T,D}$ with the faithful functors $F_T \colon \mathscr{C} \to \mathscr{C}_T$ and $G_{T,D} \colon \mathscr{C}_T \to \mathscr{C}_{T,D}$. This gives raise to a hierarchy among morphisms in $\mathscr{C}_{T,D}$. This hierarchy is useful for interpreting the decorations: *pure* terms are in $\mathscr{C}$, *propagators* are in $\mathscr{C}_T$ and *catchers* are in $\mathscr{C}_{T,D}$.

Definition 6.1.1. Let $\mathbf{C_{EXC}}$ be the interpretation of the syntax for the logic $\mathcal{L}_{exc}$ with the following details:



$$\eta : Id \Rightarrow T \qquad F_T \dashv G_T \qquad \varepsilon : T \Rightarrow Id$$

(1) The types are interpreted as the objects of $\mathscr{C}$.

(1.1) the empty type $\mathbb{O}$ is interpreted by the *initial object* of the category $\mathscr{C}$.

(1.2) for each e in $EName$, the type EV_e is interpreted as an object $EVal_e$.

(1.3) for each couple of types X and Y, the sum types $X + Y$ are interpreted as the binary coproducts in $\mathscr{C}$.

Now, we can define the object of exceptions as $E = \Sigma_{e \in EName}(EVal_e)$. The coprojections are denoted $in_e \colon EVal_e \to E$, for each exception name e. The object E in $\mathscr{C}$ is not the interpretation of a "type of exceptions". Indeed, the use of decorations in the logic $\mathcal{L}_{exc}$ provides a signature without any occurrence of such a "type of exceptions". So that signature is kept close to the syntax. Besides, for each object X in $\mathscr{C}$, the left coprojection $inl_{X,E} \colon X \to X + E$ is η_X and the right coprojection $inr_{X,E} \colon E \to X + E$, up to the isomorphism between E and $\mathbb{O} + E$, is $T([\,]_X)$.

(2) The terms are interpreted as morphisms as follows:

(2.1) a *pure* term $f^{(0)} \colon X \to Y$ in $\mathscr{C}$ as $f \colon X \to Y$ in $\mathscr{C}$

(2.2) a *propagator* term $f^{(1)} \colon X \to Y$ in $\mathscr{C}_T$ as $f \colon X \to Y + E$ in $\mathscr{C}$

(2.3) a *catcher* term $f^{(2)} \colon X \to Y$ in $\mathscr{C}_{T,D}$ as $f \colon X + E \to Y + E$ in $\mathscr{C}$

(3) The terms $f^{(1)} \colon X_1 \to Y$ and $g^{(2)} \colon X_2 \to Y$ are interpreted as $f \colon X_1 \to Y + E$ and $g \colon X_2 + E \to Y + E$ in $\mathscr{C}$. Then, $[f \mid g]_l^{(2)} \colon X_1 + X_2 \to Y$ is interpreted as the categorical copair $[f \mid g] \colon X_1 + X_2 + E \to Y + E$. It is called the *left copair* of f and g.

(4) The pure coprojections (or inclusions) $inl^{(0)}\colon X \to X + Y$ and $inr^{(0)}\colon Y \to X + Y$ are interpreted as the canonical coprojections $inl\colon X \to X + Y$ and $inr\colon Y \to X + Y$ associated to copairs.

(5) The pure term $[\ ]_X^{(0)}\colon \mathbb{O} \to X$ in $\mathscr{C}$ is interpreted as the unique mapping from the initial object $\mathbb{O}$ to the object X in $\mathscr{C}$.

(6) For each e in *EName*, the term $\mathtt{tag}_e^{(1)}\colon EV_e \to \mathbb{O}$ is a *thrower* (or *propagator*) in $\mathscr{C}_T$ and interpreted as $\mathtt{tag}_e = in_e\colon EVal_e \to E$ in $\mathscr{C}$ (up to the isomorphism between $\mathbb{O} + E$ and E).

(7) For each e in *EName*, the term $\mathtt{untag}_e^{(2)}\colon \mathbb{O} \to EV_e$ is a *catcher* in $\mathscr{C}_{T,D}$ and its interpretation, $\mathtt{untag}_e\colon E \to EVal_e + E$, is characterized by the following equalities: for each f in *EName*, such that $e \neq f$, $\mathtt{untag}_e \circ in_f = inr_{EVal_e,E} \circ in_f\colon EVal_f \to EVal_e + E$ and $\mathtt{untag}_e \circ in_e = inl_{EVal_e,E}\colon EVal_e \to EVal_e + E$ in $\mathscr{C}$.

(8) A strong equation between catchers $f^{(2)} \equiv g^{(2)}\colon X \to Y$ in $\mathscr{C}_{T,D}$ is interpreted by an equality $f = g\colon X + E \to Y + E$ in $\mathscr{C}$. Similarly, a strong equation between propagators $f^{(1)} \equiv g^{(1)}\colon X \to Y$ in $\mathscr{C}_T$ is interpreted by an equality $f = g\colon X \to Y + E$ in $\mathscr{C}$. And a strong equation between pure terms $f^{(0)} \equiv g^{(0)}\colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g\colon X \to Y$ in $\mathscr{C}$. Intuitively, two terms are strongly equal if they agree on ordinary and exceptional arguments.

(9) A weak equation between catchers $f^{(2)} \sim g^{(2)}\colon X \to Y$ is interpreted by an equality $f \circ \eta_X = g \circ \eta_X\colon X \to Y + E$ in $\mathscr{C}$. Similarly, a weak equation between propagators $f^{(1)} \sim g^{(1)}\colon X \to Y$ in $\mathscr{C}_T$ is interpreted by an equality $f = g\colon X \to Y + E$ in $\mathscr{C}$. And a weak equation between pure terms $f^{(0)} \sim g^{(0)}\colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g\colon X \to Y$ in $\mathscr{C}$. Intuitively, two terms are weakly equal if they agree on ordinary arguments, but maybe not on exceptional arguments.

The rules of the logic $\mathcal{L}_{mon}$, as stated in Figure 4.5, are rules of the logic $\mathcal{L}_{exc}$. Now, we introduce the additional rules of the logic $\mathcal{L}_{exc}$ in several steps, with some comments.

6.1.1 The effect rule

the effect rule

$$\text{(eeffect)}\ \frac{f_1, f_2\colon X \to Y \quad f_1 \sim f_2 \quad f_1 \circ [\ ]_X \equiv f_2 \circ [\ ]_X}{f_1 \equiv f_2}$$

Figure 6.4: $\mathcal{L}_{exc}$: the effect rule

(eeffect) This rule states that weak and strong equations are related with the property that $f_1 \equiv f_2$ if and only if $f_1 \sim f_2$ and $f_1 \circ [\ ]_X \equiv f_2 \circ [\ ]_X$. In other words, two terms f_1 and f_2 are strongly equal if and only if they have the same behavior on ordinary arguments ($f_1 \sim f_2$) and the same behavior on exceptional ones ($f_1 \circ [\ ]_X \equiv f_2 \circ [\ ]_X$).

6.1.2 The copair rules

rules for left the copairs

$$\text{(empty)}\ \frac{X}{[\,]_X^{(0)} : \mathbb{O} \to X} \qquad \text{(w-empty)}\ \frac{f : \mathbb{O} \to X}{f \sim [\,]_X}$$

$$\text{(lcopair)}\ \frac{f_1^{(d)} : X_1 \to Y \quad f_2 : X_2 \to Y}{[f_1 \mid f_2]_l : X_1 + X_2 \to Y}\ (\text{for all } d \leq 1)$$

$$\text{(coproj)}\ \frac{X_1 \quad X_2}{inl^{(0)} : X_1 \to X_1 + X_2 \quad inr^{(0)} : X_2 \to X_1 + X_2}$$

$$\text{(w-lcopair-eq)}\ \frac{f_1^{(d)} : X_1 \to Y \quad f_2 : X_2 \to Y}{[f_1 \mid f_2]_l \circ inl \sim f_1}\ (\text{for all } d \leq 1)$$

$$\text{(s-lcopair-eq)}\ \frac{f_1^{(d)} : X_1 \to Y \quad f_2 : X_2 \to Y}{[f_1 \mid f_2]_l \circ inr \equiv f_2}\ (\text{for all } d \leq 1)$$

(lcopair-ueq)

$$\frac{f_1 , f_2 : X_1 + X_2 \to Y \quad f_1 \circ inl \sim f_2 \circ inl \quad f_1 \circ inr \sim f_2 \circ inr}{f_1 \sim f_2}$$

Figure 6.5: $\mathcal{L}_{exc}$: rules for left copairs

(w-empty) This rule intuitively means that any term $f : \mathbb{O} \to X$ with no input parameter is said to have an equivalence on ordinary arguments with the unique mapping $[\,]_X : \mathbb{O} \to X$.

(lcopair) The rule (lcopair) states that the left copair $[f_1 \mid f_2]_l$ is defined only when f_1 is pure or is a propagator. Indeed, when both f_1 and f_2 are catchers, such a construction would lead to conflicts on exceptional arguments. When f_1 is a propagator, with (w-copair-eq), we ensure that ordinary arguments from X_1 are treated by $[f_1 \mid f_2]_l^{(2)}$ as they would be by $f_1^{(1)}$ and with (s-copair-eq) that ordinary arguments from X_2 and exceptional arguments are treated by $[f_1 \mid f_2]_l^{(2)}$ as they would be by $f_2^{(2)}$.

(lcopair-ueq) This rule ensures that a left copair structure is unique up to the weak equations.

6.1.3 Some properties of copairs

In this section, we start with a property of the "empty copair" and then prove the unicity of left copairs up to the strong equation. Afterwards, we build the symmetric (or right) copairs by using the left copairs and prove some of their properties. Lastly, we construct the left and right coproducts, by respectively using left and right copairs, and similarly prove some related properties.

Proposition 6.1.2. (s-empty) *For all d, $d' \leq 1$, given two terms of the form $f_1^{(d)}$, $f_2^{(d')} : \mathbb{O} \to X$ for each X, then $f_1 \equiv f_2$.*

Proof. Obviously, $f_1 \sim f_2$ thanks to (w-empty). Since none of them is a catcher, then $f_1 \equiv f_2$ due to (wtos). □

Proposition 6.1.3. (lcopair-u) *For each* $f_1, f_2 \colon X_1 + X_2 \to Y$, *if* $f_1 \circ inl \sim f_2 \circ inl$ *and* $f_2 \circ inr \equiv f_2 \circ inr$, *then* $f_1 \equiv f_2$.

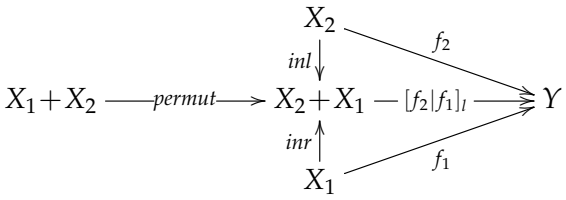
Proof. 1. Starting from $f_1 \circ inr \equiv f_2 \circ inr$, we obtain $f_1 \circ inr \circ [\,] \equiv f_2 \circ inr \circ [\,]$ due to (replsubs). Besides, we have $inr \circ [\,]_{X_2} \equiv [\,]_{X_1+X_2}$ thanks to (s-empty). Therefore, we get $f_1 \circ [\,]_{X_1+X_2} \equiv f_2 \circ [\,]_{X_1+X_2}$.

2. Since we have $f_1 \circ inr \equiv f_2 \circ inr$, by converting the strong equation into a weak equation, we get $f_1 \circ inr \sim f_2 \circ inr$. In addition, $f_1 \circ inl \sim f_2 \circ inl$ is also assumed so that we end up with $f_1 \sim f_2$ thanks to (lcopair-ueq).

Now, the above items 1 and 2 suffice to ensure $f_1 \equiv f_2$ due to (eeffect) rule introduced in Figure 6.4. □

It is possible to build symmetric *right copairs* as in Definition 6.1.4 and reason about their properties.

Definition 6.1.4. For all $d \leq 1$, given $f_1 \colon X_1 \to Y$ and $f_2^{(d)} \colon X_2 \to Y$, the *right copair* $[f_1 \mid f_2]_r = [f_2 \mid f_1]_l \circ permut$ where $permut = [inr \mid inl]_l$.



Proposition 6.1.5. *For all* $d \leq 1$, *given* $f_1 \colon X_1 \to Y$ *and* $f_2^{(d)} \colon X_1 \to Y$, *we have:*

- $[f_1 \mid f_2]_r \circ inl \equiv f_1$ (s-rcopair-eq)
- $[f_1 \mid f_2]_r \circ inr \sim f_2^{(d)}$ (w-rcopair-eq).

Proof.

- Due to (w-lcopair-eq), we have $[inr \mid inl]_l \circ inl \sim inr$. Lack of catchers yields $[inr \mid inl]_l \circ inl \equiv inr$. Through (replsubs), we obtain $[f_2 \mid f_1]_l^{(2)} \circ [inr \mid inl]_l \circ inl \equiv [f_2 \mid f_1]_l \circ inr$. Now, (s-lcopair-eq) gives $[f_2 \mid f_1]_l \circ [inr \mid inl]_l \circ inl \equiv f_1$ which folds into $[f_1 \mid f_2]_r \circ inl \equiv f_1$.
- Thanks to (s-lcopair-eq) and (stow), we get $[inr \mid inl]_l \circ inr \sim inl$. The rule (wrepl) gives $[f_2 \mid f_1] \circ [inr \mid inl]_l \circ inr \sim [f_2 \mid f_1] \circ inl$. By using (w-lcopair-eq), we obtain $[f_2 \mid f_1] \circ [inr \mid inl]_l \circ inr \sim f_2^{(d)}$ which is actually $[f_1 \mid f_2]_r \circ inr \sim f_2^{(d)}$.

□

Proposition 6.1.6. (rcopair-u) *For each* $f_1, f_2 \colon X_1 + X_2 \to Y$, *if* $f_1 \circ inl \equiv f_2 \circ inl$ *and* $f_2 \circ inr \sim f_2 \circ inr$. *Then* $f_1 \equiv f_2$.

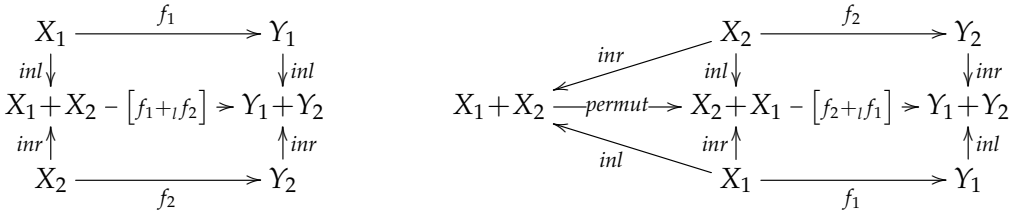
Proof. 1. Starting from $f_1 \circ inl \equiv f_2 \circ inl$, we obtain $f_1 \circ inl \circ [\,] \equiv f_2 \circ inl \circ [\,]$ due to (replsubs). Besides, we have $inl \circ [\,]_{X_1} \equiv [\,]_{X_1+X_2}$ thanks to (s-empty). Therefore, we get $f_1 \circ [\,]_{X_1+X_2} \equiv f_2 \circ [\,]_{X_1+X_2}$.

2. Since we have $f_1 \circ inl \equiv f_2 \circ inl$, by converting the strong equation into a weak equation, we get $f_1 \circ inl \sim f_2 \circ inl$. In addition, $f_1 \circ inl \sim f_2 \circ inl$ is also assumed so that we end up with $f_1 \sim f_2$ thanks to (lcopair-ueq).

Now, the above items 1 and 2 suffice to ensure $f_1^{(2)} \equiv f_2^{(2)}$ due to (eeffect) rule introduced in Figure 6.4. □

One can also define the *left-right coproducts of terms* respectively using left and right copairs.

Definition 6.1.7.
- For all $d \leq 1$, given $f_1^{(d)} \colon X_1 \to Y_1$ and $f_2 \colon X_2 \to Y_2$, we obtain a *left coproduct* $f_1 +_l f_2 = [inl \circ f_1 \mid inr \circ f_2]_l \colon X_1 + X_2 \to Y_1 + Y_2$.
- For all $d \leq 1$, given $f_1 \colon X_1 \to Y_1$ and $f_2^{(d)} \colon X_2 \to Y_2$, we obtain a *right coproduct* $f_1 +_r f_2 = [(inl \circ f_1) \mid (inr \circ f_2)]_r = [(inr \circ f_2) \mid (inl \circ f_1)]_l \circ permut$ such that $permut = [inr \mid inr]_l$.



Proposition 6.1.8. *For all $d \leq 1$, given $f_1^{(d)} \colon X_1 \to Y_1$ and $f_2 \colon X_2 \to Y_2$, we have:*

- $(f_1 +_l f_2) \circ inl \sim inl \circ f_1^{(d)}$ (w-lcoprod-eq)
- $(f_1 +_l f_2) \circ inr \equiv inr \circ f_2$ (s-lcoprod-eq)

Proof.
- By setting $f_1 := inl \circ f_1$ and $f_2 := inr \circ f_2$ within (w-lcopair-eq), one gets $[(inl \circ f_1) \mid (inr \circ f_2)]_l \circ inl \sim inl \circ f_1^{(1)}$ which folds into $(f_1 +_l f_2) \circ inl \sim inl \circ f_1^{(d)}$.
- Similarly, we set $f_1 := inl \circ f_1$ and $f_2 := inr \circ f_2$ within (s-lcopair-eq) and get $[(inl \circ f_1) \mid (inr \circ f_2)]_l \circ inr \equiv inr \circ f_2$ which is $(f_1 +_l f_2) \circ inr \equiv inr \circ f_2$. □

Proposition 6.1.9. (lcoprod-u) *For each $f_1, f_2 \colon X_1 + X_2 \to Y_1 + Y_2$, if $f_1 \circ inl \sim \circ f_2 \circ inl$ and $f_2 \circ inr \equiv \circ f_2 \circ inr$, then $f_1 \equiv f_2$.*

Proof. It suffices to apply (lcopair-u). □

Proposition 6.1.10. *For all $d \leq 1$, given $f_1 \colon X_1 \to Y_1$ and $f_2^{(d)} \colon X_2 \to Y_2$. Then;*

- $(f_1 +_r f_2) \circ inl \equiv inl \circ f_1^{(d)}$ (s-rcoprod-eq)
- $(f_1 +_r f_2) \circ inr \sim inr \circ f_2$ (w-rcoprod-eq)

Proof.
- By setting $f_1 := inl \circ f_1$ and $f_2 := inr \circ f_2$ within (s-rcopair-eq), one gets $[(inl \circ f_1) \mid (inr \circ f_2)]_r \circ inl \equiv inl \circ f_1^{(d)}$ which folds into $(f_1 +_r f_2) \circ inl \equiv inl \circ f_1^{(d)}$.
- Similarly, we set $f_1 := inl \circ f_1$ and $f_2 := inr \circ f_2$ within (w-rcopair-eq) and get $[(inl \circ f_1) \mid (inr \circ f_2)]_r \circ inr \sim inr \circ f_2$ which is $(f_1 +_r f_2) \circ inr \sim inr \circ f_2$. □

Proposition 6.1.11. (rcoprod-u) *For each $f_1, f_2 \colon X_1 + X_2 \to Y_1 + Y_2$, if $f_1 \circ inl \equiv \circ f_2 \circ inl$ and $f_2 \circ inr \sim \circ f_2 \circ inr$, then $f_1 \equiv f_2$.*

Proof. It suffices to apply (rcopair-u). □

Notice that we use some of these properties when proving the properties of programs with exceptions, in Section 6.7.

6.1.4 The interface rules

interface rules

$$\text{(tag)}\ \frac{e \in EName}{\mathtt{tag}_e^{(1)} : EV_e \to \mathbb{O}} \qquad \text{(untag)}\ \frac{e \in EName}{\mathtt{untag}_e^{(2)} : \mathbb{O} \to EV_e}$$

$$\text{(eax}_1\text{)}\ \frac{}{\mathtt{untag}_e \circ \mathtt{tag}_e \sim id_{EV_e}} \qquad \text{(eax}_2\text{)}\ \frac{\text{for each exception names (e, f) such that e} \neq \text{f}}{\mathtt{untag}_e \circ \mathtt{tag}_f \sim [\,]_{EV_e} \circ \mathtt{tag}_f}$$

$$\text{(elocal-global)}\ \frac{\text{for each exception name e, } g_1,\, g_2 : \mathbb{O} \to Y \quad g_1 \circ \mathtt{tag}_e \sim g_2 \circ \mathtt{tag}_e}{g_1 \equiv g_2}$$

Figure 6.6: $\mathcal{L}_{exc}$: the interface rules

(eax$_1$) This rule states that encapsulating an argument with an exception of name `e` followed by an immediate recovery is equivalent to "doing nothing" up to weak equation. This is because, left side of the equation may recover from an exceptional argument while the right side cannot, due to being pure.

(eax$_2$) Encapsulating an ordinary argument with an exception of name `f` and then recovering from a different exception of name `e` would just lead `f` to be propagated. It is assumed by the rule (eax$_2$) that this have "the same" behavior with encapsulating an ordinary argument with an exception of name `f` with no recovery attempt afterwards. Notice that this is only an equivalence on ordinary arguments.

(elocal-global) This rule means that for each exception name `e`, the statement of the (eeffect) rule can be expressed as a pair of weak equations for $g_1 := f_1 \circ [\,]_Y$ and $g_2 := f_2 \circ [\,]_Y$: $g_1 \sim g_2$ and $g_1 \circ \mathtt{tag\,e} \sim g_2 \circ \mathtt{tag\,e}$. Due to $g_1, g_2 : \mathbb{O} \to X$, they have apparently the same behavior on ordinary arguments. So that there is no explicit need to check whether $g_1 \sim g_2$ is true. It suffices to check if $g_1 \circ \mathtt{tag\,e} \sim g_2 \circ \mathtt{tag\,e}$ holds, in order to decide whether $g_1 \equiv g_2$ or not.

6.1.5 The downcast rule

the downcast rule

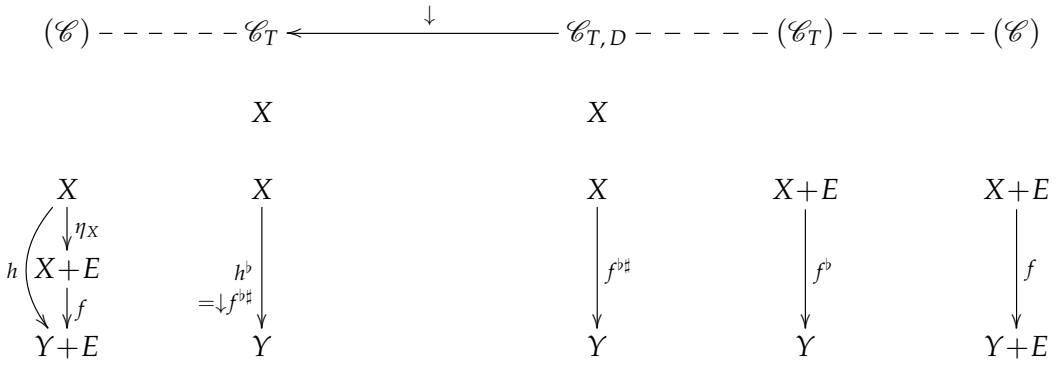
$$\text{(downcast)}\ \frac{f^{(2)} : X \to Y}{(\downarrow f)^{(1)} : X \to Y} \qquad \text{(w-downcast)}\ \frac{f : X \to Y}{\downarrow f \sim f}$$

Figure 6.7: $\mathcal{L}_{exc}$: the downcast rule

The left adjoint functor $F_{T,D} : \mathscr{C}_{T,D} \to \mathscr{C}_T$ maps each object X to $X + E$ (Section 3.2.2). There is another "mapping" what we call the *downcast*, defined from $\mathscr{C}_{T,D}$ to $\mathscr{C}_T$ as follows:

Definition 6.1.12. The "mapping" *downcast*, denoted "$\downarrow$" and defined from $\mathscr{C}_{T,D}$ to $\mathscr{C}_T$, is the identity on the objects and it maps each morphism $f^{\flat\sharp} : X \to Y$ in $\mathscr{C}_{T,D}$ to $h^\flat = \downarrow(f^{\flat\sharp}) : X \to Y$ in $\mathscr{C}_T$ where $h = f \circ \eta_X : X \to Y+E$ in $\mathscr{C}$ (notation as in Section 3.2.2).

Notice however that this "mapping" is not a functor: it does preserve identities but for $f_1^{\flat\sharp} : X \to Y$ and $f_2^{\flat\sharp} : Y \to Z$ in $\mathscr{C}_{T,D}$ we have $h^\flat = \downarrow(f_2^{\flat\sharp} \circ f_1^{\flat\sharp}) : X \to Z$ in $\mathscr{C}_T$ such that $h = f_2 \circ f_1 \circ \eta_X : X \to Z + E$ in $\mathscr{C}$, while $k^\flat = \downarrow(f_2^{\flat\sharp}) \circ \downarrow(f_1^{\flat\sharp}) : X \to Z$ in $\mathscr{C}_T$ such that $k = \mu_Z \circ Tf_2 \circ T\eta_Y \circ f_1 \circ \eta_X$ in $\mathscr{C}$. Whenever $Tf_2 \circ T\eta_Y = T\eta_Z \circ f_2$ then $\downarrow(f_2) \circ \downarrow(f_1) = \downarrow(f_2 \circ f_1)$. But in general $\downarrow(f_2) \circ \downarrow(f_1) \neq \downarrow(f_2 \circ f_1)$.



(downcast) This rule states that the "mapping" downcast exists and it is interpreted by Definition 6.1.12.

(w-downcast) This rule states that the term $\downarrow f$ behaves as f, if the argument is ordinary. If the argument is exceptional, it prevents f from catching the exceptional argument.

Now, the following result is easily obtained:

Theorem 6.1.13. *The logic $\mathcal{L}_{exc}$ is sound with respect to the interpretation $\mathbf{C_{EXC}}$ given in Definition 6.1.1.*

6.2 Decorated logic for the programmer's language for exceptions

Let us call the usual language for exceptions, with `throw` and `try/catch` blocks, the *programmers' language*. The documentation on the behavior of exceptions in many languages (for instance in Java [GJSB05]) makes use of a *core language* which we have already studied in Section 6.1. There, the empty type plays an important role together with the fundamental operations for dealing with exceptions: `tag` is used for raising while `untag` is for recovering from an exception. However, in the following, we present a logic for the *programmers' language*, with no mention of the *core language*. We call it the decorated logic for the programmers' language for exceptions and denote it by $\mathcal{L}_{exc-pl}$.

Let $\mathcal{L}'_{mon}$ be the sublogic of the logic $\mathcal{L}_{mon}$ obtained by dropping catcher terms and weak equations. The logic $\mathcal{L}_{exc-pl}$ extends the logic $\mathcal{L}'_{mon}$ with the type EV_e of parameters for each exception name $e \in EName$ where $EName$ is a finite set. In addition to the terms of the logic $\mathcal{L}'_{mon}$, the term $\mathtt{throw}_{X,e} : EV_e \to X$ stands to raise an exception of name e while the fact of catching exceptions (i.e., of name e) is hidden inside the term $\mathtt{try}(a)\mathtt{catch}(e \Rightarrow b) : X \to Y$ for each couple of terms $a : X \to Y$ and $b : EV_e \to Y$.

We give the syntax of the logic $\mathcal{L}_{exc-pl}$ in Figure 6.8 and its inference rules in Figures 4.5 and 6.9.

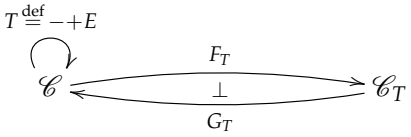
`Syntax:`	$(e \in EName)$		
Types:		$\mathtt{t}$	$::= \quad \mathtt{X} \mid \mathtt{Y} \mid \cdots \mid \mathtt{EV}_e$
Terms:		$\mathtt{f}, \mathtt{g}$	$::= \quad \mathtt{id}_\mathtt{t} \mid \mathtt{a} \mid \mathtt{b} \mid \cdots \mid \mathtt{g} \circ \mathtt{f} \mid$
			$\mathtt{throw}_{\mathtt{t},e} \mid \mathtt{try}(\mathtt{f})\,\mathtt{catch}(e \Rightarrow \mathtt{g})$
Decoration for terms:		$(\mathtt{d})$	$::= \quad (0) \mid (1)$
Equations:		$\mathtt{e}$	$::= \quad \mathtt{f} \equiv \mathtt{g}$

Figure 6.8: $\mathcal{L}_{exc-pl}$: syntax

As in Section 4.2, each term has a source and a target type as well as a decoration which is denoted as a superscript (0), (1): a *pure* term has the decoration (0), a *thrower* has (1). All terms must propagate exceptions; propagators are allowed to raise exceptions while pure terms are not.

Let $\mathscr{C}$ be the category of sets with finite coproducts and a distinguished object of exceptions E. Let $(T = - + E, \eta, \mu)$ be the exceptions monad defined over $\mathscr{C}$. Thus, the mono-requirement is satisfied (Definition 3.1.5). The interpretation of $\mathcal{L}_{exc-pl}$ is given via the Kleisli adjunction associated to a monad (detailed in Section 3.1.2) applied to the exceptions monad. Recall that in Section 3.1.2, we have introduced the adjunction $F_T \dashv G_T$ with the faithful functor $F_T : \mathscr{C} \to \mathscr{C}_T$. This gives raise to a hierarchy among morphisms in $\mathscr{C}_T$. This hierarchy is used to interpret the decorations: *pure* terms are in $\mathscr{C}$, *propagators* are in $\mathscr{C}_T$. Notice that this respectively corresponds to *values* and *computations* in Moggi's seminal paper [Mog91].

Definition 6.2.1. Let $\mathbf{C}_{\mathbf{EXC-PL}}$ be the interpretation of the syntax for the logic $\mathcal{L}_{exc-pl}$ with the following details:



$$\eta : Id \Rightarrow T \qquad\qquad \varepsilon : D \Rightarrow Id$$

(1) The types are interpreted as the objects of $\mathscr{C}$.

(1.1) for each e in $EName$, the type EV_e is interpreted as an object $EVal_e$.

(1.2) for each couple of types X and Y, the sum types $X + Y$ are interpreted as the binary coproducts in $\mathscr{C}$.

Now, we can define the object of exceptions as $E = \Sigma_{e \in EName}(EVal_e)$. The inclusions are denoted $in_e : EVal_e \to E$, for each exception name e. The object E in $\mathscr{C}$ is not the interpretation of a "type of exceptions". Indeed, the use of decorations in the logic $\mathcal{L}_{exc}$ provides a signature without any occurrence of such a "type of exceptions". So that signature is kept close to the syntax.

(2) The terms are interpreted as morphisms as follows:

(2.1) a *pure* term $f^{(0)} : X \to Y$ in $\mathscr{C}$ as $f : X \to Y$ in $\mathscr{C}$

(2.2) a *thrower* term $f^{(1)}\colon X \to Y$ in $\mathscr{C}_T$ as $f\colon X \to Y + E$ in $\mathscr{C}$

(3) A strong equation between throwers $f^{(1)} \equiv g^{(1)}\colon X \to Y$ in $\mathscr{C}_T$ is interpreted by an equality $f = g\colon X \to Y + E$ in $\mathscr{C}$. And a strong equation between pure terms $f^{(0)} \equiv g^{(0)}\colon X \to Y$ in $\mathscr{C}$ is interpreted by an equality $f = g\colon X \to Y$ in $\mathscr{C}$.

(4) The composition of two throwers $f^{(1)}\colon X \to Y$ and $g^{(1)}\colon Y \to C$ in $\mathscr{C}_T$ is interpreted by the Kleisli composition $g \circ f = \mu_C \circ Tg \circ f\colon X \to C + E$ in $\mathscr{C}$.

(5) The term $\mathtt{throw}^{(1)}_{Y,e}\colon EV_e \to Y$ in $\mathscr{C}_T$ is interpreted as $\mathtt{throw}_{Y,e} = inr_{Y,E} \circ in_e\colon EVal_e \to Y + E$ in $\mathscr{C}$.

(6) The behavior of the term $\mathtt{try}(a)\mathtt{catch}(e \Rightarrow b)$ corresponds to the Java mechanism for exceptions [GJSB05, Ch. 14] and [Jac01]: if the first exception occurring in $a^{(1)}$ is of name e, then the computation continues with $b^{(1)}$. Formally, for each pair of throwers $a^{(1)}\colon X \to Y$, $b^{(1)}\colon EV_e \to Y$ and for each exception name e;

- if $a^{(1)} = v^{(1)} \circ \mathtt{throw}^{(1)}_{Z,e} \circ u^{(0)}$ for some terms $v^{(1)}\colon Z \to Y$, $u^{(0)}\colon X \to EV_e$, then $\mathtt{try}(a)\,\mathtt{catch}(e \Rightarrow b)^{(1)}\colon X \to Y$ in $\mathscr{C}_T$ has the same interpretation as $b^{(1)} \circ u^{(0)}\colon X \to Y$ in $\mathscr{C}_T$.
- otherwise $\mathtt{try}(a)\,\mathtt{catch}(e \Rightarrow b)^{(1)}$ in $\mathscr{C}_T$ is interpreted as $a\colon X \to Y + E$ in $\mathscr{C}$.

In addition to the rules of the logic $\mathcal{L}'_{mon}$, we have the following rules related to `throw` and `try/catch` structures:

rules for the programmers' language

$$\text{(throw)}\ \frac{Y \quad e \in EName}{\mathtt{throw}^{(1)}_{Y,e}\colon EV_e \to Y} \qquad \text{(try-catch)}\ \frac{a\colon X \to Y \quad b\colon EV_e \to Y \quad e \in EName}{\mathtt{try}(a)\,\mathtt{catch}(b \Rightarrow e)^{(1)}\colon X \to Y}$$

$$\text{(ppt)}\frac{a : X \to Y}{a \circ \mathtt{throw}_{X,e} \equiv \mathtt{throw}_{Y,e}} \qquad \text{(rcv)}\frac{u_1^{(0)}, u_2^{(0)} : X \to EV_e \quad \mathtt{throw}_{Y,e} \circ u_1 \equiv \mathtt{throw}_{Y,e} \circ u_2}{u_1 \equiv u_2}$$

$$\text{(try)}\ \frac{a_1,\, a_2\colon X \to Y \quad b\colon EV_e \to Y \quad a_1 \equiv a_2}{\mathtt{try}(a_1)\mathtt{catch}(e \Rightarrow b) \equiv \mathtt{try}(a_2)\mathtt{catch}(e \Rightarrow b)} \qquad \text{(try}_0\text{)}\frac{u^{(0)}\colon X \to Y \quad b\colon EV_e \to Y}{\mathtt{try}(u)\mathtt{catch}(e \Rightarrow b) \equiv u}$$

$$\text{(try}_1\text{)}\ \frac{u^{(0)}\colon X \to EV_e \quad b\colon EV_e \to Y}{\mathtt{try}(\mathtt{throw}_{Y,e} \circ u)\mathtt{catch}(e \Rightarrow b) \equiv b \circ u}$$

$$\text{(try}_2\text{)}\ \frac{\text{for each } (e,f) \in EName, \text{ such that } e \neq f \quad u^{(0)}\colon X \to EV_f \quad b\colon EV_e \to Y}{\mathtt{try}(\mathtt{throw}_{Y,f} \circ u)\mathtt{catch}(e \Rightarrow b) \equiv \mathtt{throw}_{Y,f} \circ u}$$

Figure 6.9: $\mathcal{L}_{exc-pl}$: rules for the programmers' language

(ppt) The rule states that exceptions are always propagated.

(rcv) It ensures that the parameter used for throwing an exception may be recovered.

(try) It states that the strong equation is compatible with the `try/catch`.

(try$_0$) With this rule we assume that the pure code inside the `try` part never triggers the code inside the `catch` part.

(try$_1$) By this rule, we assume that the code inside the `catch` part is executed as soon as an exception is thrown inside the `try` part.

(try$_2$) This rule states that an exception cannot be handled, if the pattern matching on exception names is not successful. This means that the exception is propagated.

Now, the following result is easily obtained:

Theorem 6.2.2. *The logic $\mathcal{L}_{exc-pl}$ is sound with respect to the interpretation $\mathbf{C}_{\mathbf{EXC-PL}}$ given in Definition 6.2.1.*

6.3 Translating the logic $\mathcal{L}_{exc-pl}$ into the logic $\mathcal{L}_{exc}$

The decorated logic for the programmer's language of exceptions ($\mathcal{L}_{exc-pl}$) does not include the private `tag` and `untag` operations, but the public `throw` and `try/catch` constructs. In this section, we show that they can be built in terms of `tag` and `untag` in the logic $\mathcal{L}_{exc}$. The main ingredients for building the logic $\mathcal{L}_{exc-pl}$ from the logic $\mathcal{L}_{exc}$ are the coproducts $X \cong X + \mathbb{O}$ and the *downcasting conversion* with the downcast rules (See Figure 6.7).

Remark 6.3.1. Note that, for the sake of conciseness, here we assume that only one exception name is handled in a `try/catch` expression: the general case is treated in [DDR13].

Definition 6.3.2. For each type Y and each exception name e, the propagator $\mathtt{throw}^{(1)}_{Y,e}$ is:

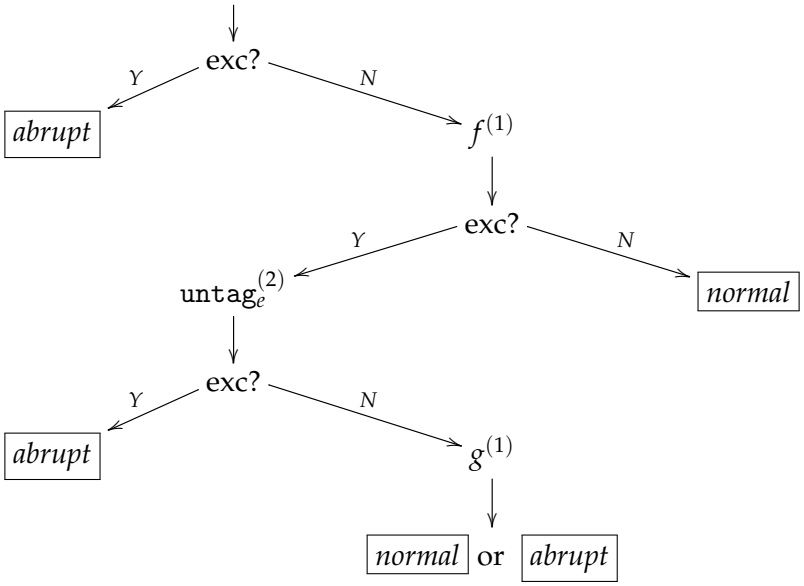
$$\mathtt{throw}^{(1)}_{Y,e} = [\ ]^{(0)}_{Y} \circ \mathtt{tag}^{(1)}_{e} : EV_e \to Y$$

This means that raising an exception with name e is first tagging the given ordinary value as an exception of name e and then converting it to the given type Y.

Definition 6.3.3. For each propagators $f^{(1)} : X \to Y$, $g^{(1)} : EV_e \to Y$ and each exception name e, the propagator $\mathtt{try}(f)\mathtt{catch}(e \Rightarrow g)^{(1)}$ is defined in three steps, as follows:

$$\begin{array}{lcll}
\mathtt{CATCH}(e \Rightarrow g)^{(2)} & = & [\ id^{(0)}_{Y} \mid g^{(1)} \circ \mathtt{untag}^{(2)}_{e}\ ]_{l} : & Y+\mathbb{O} \to Y \\
\mathtt{TRY}(f)\mathtt{CATCH}(e \Rightarrow g)^{(2)} & = & \mathtt{CATCH}(e \Rightarrow g)^{(2)} \circ inl^{(0)}_{Y,\mathbb{O}} \circ f^{(1)} : & X \to Y \\
\mathtt{try}(f)\mathtt{catch}(e \Rightarrow g)^{(1)} & = & \downarrow(\mathtt{TRY}(f)\mathtt{CATCH}(e \Rightarrow g)^{(2)}) : & X \to Y
\end{array}$$

To handle an exception, the intermediate expressions $\mathtt{CATCH}(e \Rightarrow g)$ and $\mathtt{TRY}(f)\mathtt{CATCH}(e \Rightarrow g)$ are private catchers and the expression $\mathtt{try}(f)\mathtt{catch}(e \Rightarrow g)$ is a public propagator: the downcast operator prevents it from catching exceptions with name e which might have been raised before the $\mathtt{try}(f)\mathtt{catch}(e \Rightarrow g)$ expression is considered. The definition of $\mathtt{try}(f)\mathtt{catch}(e \Rightarrow g)$ corresponds to the Java mechanism for exceptions [GJSB05, Ch. 14] and [Jac01] with the following control flow, where `exc?` means "*is this value an exception?*", an *abrupt* termination returns an uncaught exception and a *normal* termination returns an ordinary value.

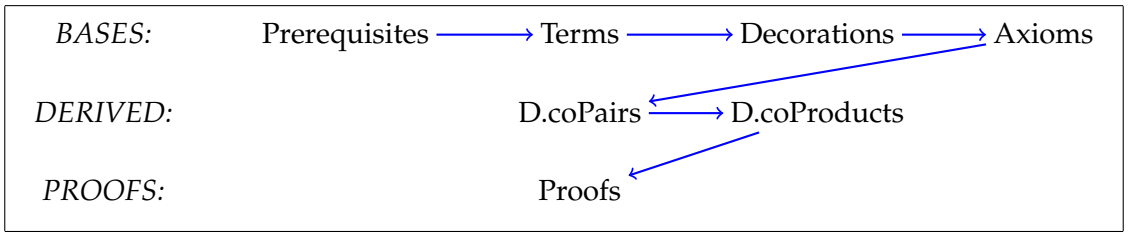


Theorem 6.3.4. *If the pure term* $[\,]_Y : \mathbb{O} \to Y$ *is a monomorphism with respect to propagators for each type* Y*, the above stated translation of the logic* $\mathcal{L}_{exc-pl}$ *into the logic* $\mathcal{L}_{exc}$ *is correct.*

Proof. It is shown by Propositions 6.7.3, 6.7.4, 6.7.5, 6.7.6 6.7.7 and 6.7.8 that the images of (six) basic properties of `throw` and `try/catch` are satisfied. □

6.4 The logic $\mathcal{L}_{exc}$ in Coq

The main scope of this section is to formalize the *decorated logic for exceptions* ($\mathcal{L}_{exc}$) in Coq [DDER14]. To do so, we aim to enrich the implementation of the logic $\mathcal{L}_{mon}$ that is already detailed in Sections 4.4.1 4.4.2 and 4.4.4: we will reuse the code blocks in order to preserve the integrity of the formalization with no repeated explanation. The organization of the modules is reflected in the Coq library `EXCEPTIONS-THESIS` as follows:



Remark 6.4.1. The complete `EXCEPTIONS-THESIS` library can be found on `https://forge.imag.fr/frs/download.php/694/EXCEPTIONS-THESIS.tar.gz`.

6.4.1 Prerequisites

In order to enrich the terms of the logic $\mathcal{L}_{mon}$, we first need to speak about some preliminaries: the set of exception names is implemented as a Coq parameter `EName : Type`. Provided that there might be several exceptional values of a certain exception name e, we implement an arrow type `EV : EName` $\to$ `Type` that is the type of exceptional values of a

certain exception name. Notice also that the implementation follows the same approach as the one for states already detailed in Section 5.2.

```
Parameter EName: Type.      Parameter EV: EName → Type.
```

6.4.2 Terms

We implement the additional terms as new constructors to the dependent type `term` given in Section 4.4.1:

(1) $[f \mid g]: X+Y \to Z$ for each couple of terms $f: X \to Z$ and $g: Y \to Z$, together with the canonical projections $inl: X \to X+Z$ and $inr: Z \to X+Z$,

(2) $[\,]_X: \mathbb{O} \to X$ for each type X,

(3) $\downarrow f: X \to Y$, for each types X, Y and term $f: X \to Y$,

(4) $\texttt{tag}_e: EV_e \to \mathbb{O}$ for each exception name e,

(5) $\texttt{untag}_e: \mathbb{O} \to EV_e$ for each exception name e.

Thus, the implementation of terms in Coq looks like:

```
Inductive term: Type → Type → Type :=
 | comp:    forall {X Y Z: Type}, term X Y → term Y Z → term X Z
 | copair: forall {X Y Z: Type}, term Z X → term Z Y → term Z (X + Y)
 | downcast: forall {X Y: Type} (f: term X Y), term X Y
 | tpure:  forall {X Y: Type}, (X → Y) → term Y X
 | tag: forall e:EName, term Empty_set (EV t)
 | untag: forall e:EName, term (EV t) Empty_set.
Infix "o" := comp (at level 70).
```

Instead of the symbols $[\ |\ ]$ and $\downarrow$, we respectively use the keywords `copair` and `downcast` in the implementation. We derive terms such as the identity, the copair coprojections and the empty copair from the native Coq functions, with the use of `tpure` constructor, as follows:

```
Definition id {X: Type}       : term X X         := tpure id.
Definition coproj1 {X Y}      : term (X + Y) X   := tpure inl.
Definition coproj2 {X Y}      : term (X + Y) Y   := tpure inr.
Definition emptyfun (X: Type) (e: Empty_set) : X := match e with end.
Definition empty X: term X Empty_set := tpure (emptyfun X).
```

Remark also that the copair coprojections are named `coproj1` and `coproj2` while the empty pair is called `empty` in the implementation.

Remark 6.4.2. See the source `Terms.v` for related implementation details.

6.4.3 Decorations

Thereby, the decorations' implementation follows:

```
Inductive kind := epure | ppg | ctc.
Inductive is: kind → forall X Y, term X Y → Prop :=
 | is_tpure: forall X Y (f: X → Y), is epure (@tpure X Y f)
 | is_comp: forall k X Y Z (f: term X Y) (g: term Y Z), is k f → is k g → is k (f o g)
 | is_copair: forall k X Y Z (f: term Z X) (g: term Z Y), is ppg f → is k f → is k g → is k (copair f g)
 | is_downcast: forall X Y (f: term X Y), is ppg (@downcast X Y f)
 | is_tag: forall t, is ppg (tag t)
 | is_untag: forall t, is ctc (untag t)
 | is_pure_ppg: forall X Y (f: term X Y), is epure f → is ppg f
 | is_ppg_ctc: forall X Y (f: term X Y), is ppg f → is ctc f.
Hint Constructors is.
```

Notice that instead of the decorations of the form (0), (1) and (2), we respectively use the keywords `epure`, `ppg` and `ctc` in the implementation. The decoration of any composed and co-paired off term depends on its components and always takes the upper decoration (`epure` $<$ `ppg` $<$ `ctc`). E.g., given a catcher term and a propagator term, their composition and copair will be a catcher, as well. We declare the term `tag` as a propagator by using the keyword `ppg`. On the contrary, `untag` is a catcher with the decoration keyword `ctc`. The term `downcast f` is a propagator, for each catcher term `f`. It is trivial to derive that the copair coprojections are pure. For the sake of conciseness, we demonstrate only the first one:

```
Lemma is_coproj1 X Y: is pure (@coproj1 X Y). Proof. apply is_tpure. Qed.
```

Since `coproj1` is constructed through `tpure` and since any argument of `tpure` is by definition epure, it suffices to apply the constructor `is_tpure`. Recall that the process of decoration checking is crucial and troublesome in a decorated setting: to automatize the verification of the decorations, as in Section 5.2.3, we build a new tactic named `edecorate`, by using Delahaye's `Ltac` language [Del00]:

```
Ltac edecorate := solve[
  repeat (apply is_comp || apply is_copair)
       ||
  (apply is_tpure || apply is_downcast || apply is_tag || apply is_untag || assumption)
       ||
  (apply is_pure_ppg)
       ||
  (apply is_ppg_ctc) ].
```

The tactic `edecorate` repeatedly checks if the goal term is a composition or a copair, if not so, it tries to decide whether it is a pure term constructed via `tpure` or one of the following terms: `downcast`, `tag` and `untag` or else a local assumption. If it is still not the case, it applies the hierarchy rules. All that are performed in the given sequence. Since these checks are done inside the `solve` tactical, `edecorate` fails in the absence of match.

```
Class PURE {X Y: Type} (f: term X Y) := ispure : is pure f.
Hint Extern 0 (PURE _) ⇒ edecorate : typeclass_instances.

Class PPG {X Y: Type} (f: term X Y) := isppg : is ppg f.
Hint Extern 0 (PPG _) ⇒ edecorate : typeclass_instances.

Class CTC {X Y: Type} (f: term X Y) := isctch : is ctc f.
Hint Extern 0 (CTC _) ⇒ edecorate : typeclass_instances.
```

The assignment of decorations over terms is declared as constructors of Coq type classes parametrized by a `term`. Then, we extend the scope of the tactic `auto` with the optional patterns `(PURE _)`, `(PPG _)`, `(CTC _)`, the tactic `edecorate` at cost zero. This is provided by the vernacular command `Extern (num) pattern => tactic`. The zero cost here means that the tactic `auto` would non-recursively try the hints upon the usage.

Remark 6.4.3. See the source `Decorations.v` for related implementation details.

6.4.4 Axioms

Here we give to the formalization of the rules/axioms in Coq.

```
Reserved Notation "x == y" (at level 80).  Reserved Notation "x ~ y" (at level 80).
Definition idem X Y (x y: term X Y) := x = y.
Inductive strong: forall X Y, relation (term X Y) :=
    (*congruence rules*)
| refl: forall X Y (f: term X Y), f == f
| sym: forall X Y, Symmetric (@strong X Y)
| trans: forall X Y, Transitive (@strong X Y)
| replsubs: forall X Y Z, Proper (@strong X Y ==> @strong Y Z ==> @strong X Z) comp
    (*categorical rules*)
| ids: forall X Y (f: term X Y), f o id == f
| idt: forall X Y (f: term X Y), id o f == f
| assoc: forall X Y Z T (f: term X Y) (g: term Y Z) (h: term Z T), f o (g o h) == (f o g) o h
    (*the hierarchy rule*)
| wtos: forall X Y (f g: term X Y), PPG f → PPG g → f ~ g → f == g
    (*strong copair rules*)
| s_lcopair_eq: forall X X' Y (f1: term Y X) (f2: term Y X'), PPG f1 → (copair f1 f2) o coproj2 == f2
    (*the effect rule*)
| eeffect: forall X Y (f g: term Y X), f ~ g → (f o (@empty X) == g o (@empty X)) → f == g
    (*the strong interface rule*)
| elocal_global: forall X (f g: term X Empty_set), (forall t: EName, f o tag t ~ g o tag t) → f == g
        (*tpure preserves the pure composition*)
| tcomp: forall X Y Z (f: Z → Y) (g: Y → X), tpure (compose g f) == tpure g o tpure f
with weak: forall X Y, relation (term X Y) :=
    (*congruence rules*)
| wsym: forall X Y, Symmetric (@weak X Y)
| wtrans: forall X Y, Transitive (@weak X Y)
| wrepl : forall X Y C, Proper (@idem C Y ==> @weak Y X ==> @weak C X) comp
| pwsubs : forall X Y C, Proper (@weak C Y ==> @pure_id Y X ==> @weak C X) comp
    (*the hierarchy rule*)
| stow: forall X Y (f g: term X Y), f == g → f ~ g
    (*the weak copair rule*)
| w_lcopair_eq: forall X X' Y (f1: term Y X) (f2: term Y X'), PPG f1 → (copair f1 f2) o coproj1 ~ f1
| w_empty: forall X (f: term X Empty_set), f ~ (@empty X)
    (*the down-casting rule*)
| w_downcast: forall X Y (f: term X Y), f ~ (@downcast X Y f)
    (*weak interface rules*)
| eax1: forall t: EName, untag t o tag t ~ (@id (Val t))
| eax2: forall t1 t2: EName, t1 <> t2 → untag t2 o tag t1 ~ (@empty (Val t2)) o tag t1
    (*the weak unicity rule*)
| lcopair_ueq: forall X X' Y (f g: term Y (X + X')), (f o coproj1 ~ g o coproj1) →
     (f o coproj2 ~ g o coproj2) → f ~ g
  where "x == y" := (strong x y) and "x ~ y" := (weak x y).
```

On the details of additional rules. For `w_empty`, `s_lcopair_eq`, `w_lcopair_eq` and `lcopair_ueq`, see Figure 6.5. The rule `eeffect` is given in Figure 6.4. For `eax1`, `eax2` and `elocal_global`, refer back to Figure 6.6. Lastly, `w_downcast` is detailed in Figure 6.7.

Before detailing properties related to copairs and coproducts, let us hereby give the certified Coq proof of the derived property (s-empty) as it is given in Proposition 6.1.2. The rule (s-empty) claims that the equation among parallel terms with domain $\mathbb{O}$ is strong, provided that they are pure or propagators.

```
Lemma s_empty: forall X (f: term X Empty_set), PPG f → f == (@empty X).
Proof. intros X f H. apply wtos; [exact H | edecorate | apply w_empty]. Qed.
```

The proof converts the goal-side strong equation into weak provided that both `f` and `empty` are non-catchers. Now, it suffices to apply (w-empty) to close the goal.

Remark 6.4.4. See the source `Axioms.v` for related implementation details.

6.4.5 Derived copairs and coproducts

In order to speak about symmetric or (right) copairs as well as left and right coproducts, we define the permutation term, denoted `permut`. It inputs two Coq `Type` instances `X` and `Y` and outputs an instance of type: `term (Y*X) (X*Y)`.

```
Definition permut {X Y} : term (X + Y) (Y + X) := copair coproj2 coproj1.
```

Clearly, `permut` is a *pure* term since it is a left copair made of *pure* projections. Now, the right copair structure looks like:

```
Definition rcopair {X Y Z} (f1: term X Y) (f2: term X Z) : term X (Y + Z) := copair f2 f1 o (@permut Z Y).
```

The decoration of a given right copair depends on its components:

```
Lemma is_rcopair: forall k X Y Z (f1: term Z X) (f2: term Z Y), PPG f2 → is k f1 → is k f2
                  → is k (rcopair f1 f2).
Proof. intros k X Y Z f1 f2 H1 H2 H3. induction k; edecorate. Qed.
```

After introducing the necessary instances, we induce on the `kind k`. Then it suffices to `edecorate` each goal: `is pure (rcopair f1 f2)`, `is ppg (rcopair f1 f2)` and `is ctc (rcopair f1 f2)` locally provided: `H0: is pure f1`, `H1: is pure f2`; `H0: is ppg f1`, `H1: is ppg f2` and `H0: is ctc f1`, `H1: is ctc f2`.

The projection rules attached to right copairs, that are stated and proven in Proposition 6.1.5, are certified in Coq along with their proofs:

right copair: first coprojection

```
Lemma s_rcopair_eq: forall X Z Y (f1: term X Y) (f2: term X Z), PPG f2 → rcopair f1 f2 o coproj1 == f1.
Proof.
    intros X Y' Y f1 f2 H0. unfold rcopair, permut. rewrite ←assoc.
    cut (copair coproj2 coproj1 o coproj1 == (@coproj2 Y' Y)).
      intro H1. rewrite H1.
      apply s_lcopair_eq; exact H0.
    (*1st cut*)
    apply wtos;[edecorate | edecorate | ]. apply w_lcopair_eq; edecorate.
Qed.
```

After forming the environment of the assumptions, the proof continues with unfolding `rcopair` and `permut` followed by rewriting associativity which shifts parentheses to the right. At this point, the goal looks like: `copair f2 f1 o (copair coproj2 coproj1 o coproj1) == f1`. We cut the `Prop` instance, `(copair coproj2 coproj1 o coproj1 == (@coproj2 Z Y))` and introduce an instance of it named `H1`. We rewrite `H1` and obtain `copair f2 f1 o coproj2 == f1`. Now, it suffices to apply the rule `s_lcopair_eq` and prove that `PPG f2` which is an assumption. It is necessary to prove the strong equation that we have already cut. There, we first convert the goal side strong equation into a weak equation provided that `copair coproj2 coproj1 o coproj1` and `(@coproj2 Z Y)` are both propagator. So that the goal turns into `(copair coproj2 coproj1 o coproj1 ∼ coproj2)`. It suffices to apply `w_lpair_eq` and prove that `PPG coproj2` which is closed by `edecorate`.

right copair: second coprojection

```
Lemma w_rcopair_eq: forall X Z Y (f1: term X Y) (f2: term X Z), PPG f2 → rcopair f1 f2 o coproj2 ∼ f2.
Proof.
    intros X Y' Y f1 f2 H. unfold rcopair, permut. rewrite ←assoc.
    rewrite s_lcopair_eq;[| edecorate]. rewrite w_lcopair_eq;[reflexivity| edecorate].
Qed.
```

After some preliminary modifications on the goal (by following the first line in the proof), we obtain `copair f2 f1 o (copair coproj2 coproj1 o coproj2) ∼ f2`. We rewrite `s_lcopair_eq` which results in two subgols: `copair f2 f1 o coproj1 ∼ f2` and `PPG coproj2`. The application of the rule `w_lcopair_eq` followed by the `edecorate` solves the first subgoal and `decorate` alone closes the second.

We also certify the proofs of Propositions 6.1.3 and 6.1.6 ensuring that left and right copairs are unique with respect to strong equation:

left copair: unicity

```
Lemma lcopair_u: forall X Y Y' (f1 f2: term X (Y' + Y)),
  (f1 o coproj1 ∼ f2 o coproj1) ∧ (f1 o coproj2 == f2 o coproj2) → f1 == f2.
Proof.
    intros X Y Y' f1 f2 (H0&H1). apply eeffect.
    (* f1 ∘ [ ] ≡ f2 ∘ [ ] *)
    cut((@coproj2 Y' Y) o (@empty Y) == (@empty (Y' + Y))).
     intro H2. rewrite ←H2.
     setoid_rewrite assoc. rewrite H1. reflexivity.
    (* 1st cut *)
    setoid_rewrite s_empty; [reflexivity| edecorate].
    (* f1 ∼ f2 *)
    apply lcopair_ueq. exact H0. apply stow. exact H1.
Qed.
```

right copair: unicity

```
Lemma rcopair_u: forall X Y Y' (f1 f2: term X (Y' + Y)),
  (f1 o coproj1 == f2 o coproj1) ∧ (f1 o coproj2 ∼ f2 o coproj2) → f1 == f2.
Proof.
    intros X Y Y' f1 f2 (H0&H1). apply eeffect.
    (* f1 ∘ [ ] ≡ f2 ∘ [ ] *)
    cut((@coproj1 Y' Y) o (@empty Y') == (@empty (Y' + Y))).
      intro H2. rewrite ←H2.
      setoid_rewrite assoc. rewrite H0. reflexivity.
    (* 1st cut *)
    setoid_rewrite s_empty; [reflexivity| edecorate].
    (* f1 ∼ f2 *)
    apply lcopair_ueq. apply stow. exact H0. exact H1.
Qed.
```

Both proofs follow the same approach: first the (eeffect) rule is applied to the goal of the form `f1 == f2`. This generates two subgoals to prove: `f1 o empty == f2 o empty` and `f1 ∼ f2`. Then, depending on the assumptions, we use the fact `coproj1 o empty == empty` or `coproj2 o empty == empty` ensured by (s-empty) to close the first subgoal. For the second subgoals, we use (lcopair-eq) rule to conclude with `f1 == f2`.

In addition, one can derive the left coproducts out of copairs and coprojections as:

```
Definition lcoprod {X1 Y1 X2 Y2} (f1: term X1 X2) (f2: term Y1 Y2) : term (X1 + Y1) (X2 + Y2)
:= copair (coproj1 o f1) (coproj2 o f2).
```

Now, right coproduct structure looks like:

```
Definition rcoprod {X Y X' Y'} (f1: term X X') (f2: term Y Y') : term (X + Y) (X' + Y')
:= rcopair (coproj1 o f1) (coproj2 o f2).
```

One can simply prove that the decoration of a *term coproduct* depends on its components:

```
Lemma is_lcoprod: forall k X' X Y' Y (f1: term X X') (f2: term Y Y'), PPG f1 → is k f1 → is k f2
                  → is k (lcoprod f1 f2).
Proof. intros k X' X Y' Y f1 f2 H1 H2 H3. induction k; edecorate. Qed.
```

After introducing the necessary instances, we induce on the `kind k`. Then, it suffices to `edecorate` each goal: `is pure (lcoprod f1 f2)`, `is ppg (lcoprod f1 f2)` and `is ctc (lcoprod f1 f2)` locally provided: `H0: is pure f1`, `H1: is pure f2`; `H0: is ppg f1`, `H1: is ppg f2` and `H0: is ctc f1`, `H1: is ctc f2`. The similar idea applies to the case of right coproducts:

```
Lemma is_rcoprod: forall k X' X Y' Y (f1: term X X') (f2: term Y Y'), PPG f2 → is k f1 → is k f2
                  → is k (rcoprod f1 f2).
Proof. intros k X' X Y' Y f1 f2 H1 H2 H3. induction k; edecorate. Qed.
```

The coprojection (or inclusion) rules attached to left and right coproducts, that are stated and proved in Propositions 6.1.8 and 6.1.10, are certified in Coq along with their proofs:

left and right coproducts: first and second coprojections

```
Lemma s_lcoprod_eq: forall X1 X2 Y1 Y2 (f: term X1 X2) (g: term Y1 Y2),
  PPG f → (lcoprod f g) o coproj1 ∼ coproj1 o f.
Proof. intros X1 X2 Y1 Y2 f g H. apply w_lcopair_eq; edecorate. Qed.

Lemma w_lcoprod_eq: forall X1 X2 Y1 Y2 (f: term X1 X2) (g: term Y1 Y2),
  PPG f → (lcoprod f g) o coproj2 == coproj2 o g.
Proof. intros X1 X2 Y1 Y2 f g H. apply s_lcopair_eq; edecorate. Qed.

Lemma s_rcoprod_eq: forall X1 X2 Y1 Y2 (f: term X1 X2) (g: term Y1 Y2),
  PPG g → (rcoprod f g) o coproj1 == coproj1 o f.
Proof. intros X1 X2 Y1 Y2 f g H. apply s_rcopair_eq; edecorate. Qed.

Lemma w_rcoprod_eq: forall X1 X2 Y1 Y2 (f: term X1 X2) (g: term Y1 Y2),
  PPG g → (rcoprod f g) o coproj2 ∼ coproj2 o g.
Proof. intros X1 X2 Y1 Y2 f g H. apply w_rcopair_eq; edecorate. Qed.
```

They are nothing but the specialized versions of (w-lcopair-eq), (s-lcopair-eq), (w-rcopair-eq) and (s-rcopair-eq).

We lastly have the unicity properties of left and right coproducts with respect to strong equation, that are stated and proven in Propositions 5.1.9 and 5.1.11, certified in Coq:

left and right coproducts: unicity

```
Lemma lcoprod_u: forall X1 X2 Y1 Y2 (f1 f2: term (Y2 + Y1) (X2 + X1)),
   (f1 o coproj1 ∼ f2 o coproj1) ∧ (f1 o coproj2 == f2 o coproj2) → f1 == f2.
Proof. intros X1 X2 Y1 Y2 f1 f2 (H0&H1). apply lcopair_u. split; [exact H0 | exact H1]. Qed.

Lemma rcoprod_u: forall X1 X2 Y1 Y2 (f1 f2: term (Y2 + Y1) (X2 + X1)),
   (f1 o coproj1 == f2 o coproj1) ∧ (f1 o coproj2 ∼ f2 o coproj2) → f1 == f2.
Proof. intros X1 X2 Y1 Y2 f1 f2 (H0&H1). apply rcopair_u. split; [exact H0 | exact H1]. Qed.
```

It suffices to respectively apply (lcopair_u) and (rcopair_u) to close the goals.

Remark 6.4.5. See the sources `Derived_coPairs.v` and `Derived_coProducts.v` for related implementation details.

6.5 The logic $\mathcal{L}_{exc-pl}$ in Coq

The Coq implementation of the logic $\mathcal{L}_{exc-pl}$ follows the same approach with the one of the logic $\mathcal{L}_{exc}$, as in Section 6.4, and it can be found in the `EXCEPTIONS-THESIS` library.

6.5.1 Terms

By using the same preliminaries, the implementation of terms looks like:

```
Inductive termpl: Type → Type → Type :=
 | pl_tpure: forall {X Y: Type}, (X → Y) → termpl Y X
 | pl_comp: forall {X Y Z: Type}, termpl X Y → termpl Y Z → termpl X Z
 | throw: forall {X} (e: EName), termpl X (Val e)
 | try_catch: forall {X Y} (e: EName), termpl Y X → termpl Y (Val e) → termpl Y X.
Notation "a 'O' b" := (pl_comp a b) (at level 70).
```

Terms are inductively defined via a dependent type called `termpl`. In addition to the main constructors `throw` and `try_catch`, via `pl_tpure`, we introduce Coq side pure functions. The constructor `pl_comp` enables to compose compatible terms. Notice that there is neither `tag` nor `untag` involved. Lastly, the keyword '`O`' is used to notate the composition of compatible terms. For the ease of further usage, we introduce some basic Coq side functions inside the decorated environment via the use of the `pl_tpure` constructor.

```
Definition pl_id {X: Type} : termpl X X          := pl_tpure Datatypes.id.
```

The term `pl_id` makes use of the native `id` function in Coq and constructs the `termpl X X` type for each `X : Type`.

Remark 6.5.1. See the source `Terms.v` for related implementation details.

6.5.2 Decorations

Since there is no catcher term in the logic $\mathcal{L}_{exc-pl}$, the enumerated type `kindpl` implements decorations with two constructors: `pl_pure` and `pl_ppg`.

```
Inductive kindpl := pl_pure | pl_ppg.
Inductive is_pl: kindpl → forall X Y, termpl X Y → Prop :=
| is_pl_tpure: forall X Y (f: X → Y), is_pl pl_pure (@pl_tpure X Y f)
| is_pl_comp: forall k X Y Z (f: termpl X Y) (g: termpl Y Z), is_pl k f → is_pl k g → is_pl k (f O g)
| is_throw: forall X (e: EName), is_pl pl_ppg (@throw X e)
| is_try_catch: forall X Y (e: EName) (a: termpl Y X) (b: termpl Y (Val e)), is_pl pl_ppg (@try_catch _ _ e a b)
| is_pl_pure_ppg: forall X Y (f: termpl X Y), is_pl pl_pure f → is_pl pl_ppg f.
Hint Constructors is_pl.
```

Anything defined over `pl_tpure` is declared to be pure. The decoration of a composed term depends on its components. The terms `throw` and `try_catch` are defined to be propagators. The hierarchy among decorations is also there as the last constructor: a pure term can be seen as propagator.

```
Class PL_EPURE {X Y: Type} (f: termpl X Y) := isplp : is_pl pl_epure f.
Class PL_PPG   {X Y:Type} (f: termpl X Y) := isplppg : is_pl pl_ppg f.
```

Remark 6.5.2. See the source `Decorations.v` for related implementation details.

6.5.3 Axioms

There is only one type of equation relating (strong equation, denoted $*==$) the terms.

```
Reserved Notation "x ∗ == y" (at level 80).
Inductive pl_strong: forall X Y, relation (termpl X Y) :=
| pl_refl: forall X Y (f: termpl X Y), f ∗ == f
| pl_sym: forall X Y, Symmetric (@pl_strong X Y)
| pl_trans: forall X Y, Transitive (@pl_strong X Y)
| pl_assoc: forall X Y Z T (f: termpl X Y) (g: termpl Y Z) (h: termpl Z T), f O (g O h) ∗ == (f O g) O h
| pl_ids: forall X Y (f: termpl X Y), f O pl_id ∗ == f
| pl_idt: forall X Y (f: termpl X Y), pl_id O f ∗ == f
| pl_replsubs: forall X Y Z, Proper (@pl_strong X Y ==> @pl_strong Y Z ==> @pl_strong X Z) (pl_comp)
   (*for throw and try/catch*)
| ppt : forall X Y e (a: termpl X Y), a O (@throw Y e) ∗ == (@throw X e)
| rcv : forall X Y e (u1 u2: termpl (Val e) Y), (@throw X e) O u1 ∗ == (@throw X e) O u2 → u1 ∗ == u2
| try : forall X Y e (a1 a2: termpl X Y) (b: termpl X (Val e)), a1 ∗ == a2 →
          try_catch e a1 b ∗ == try_catch e a2 b
| try0: forall X Y e (u: termpl X Y) (b: termpl X (Val e)), PL_EPURE u → try_catch e u b ∗ == u
| try1: forall X Y e (u: termpl (Val e) Y) (b: termpl X (Val e)), PL_EPURE u →
    try_catch e ((@throw X e) O u) b ∗ == b O u
| try2: forall X Y e f (u: termpl (Val f) X) (b: termpl Y (Val e)), e <> f → PL_EPURE u →
    try_catch e ((@throw Y f) O u) b ∗ == (@throw Y f) O u
     (*pl_tpure preserves the pure composition*)
| pl_tcomp: forall X Y Z (f: Z → Y) (g: Y → X), pl_tpure (compose g f) ∗ == pl_tpure g O pl_tpure f
 where "x ∗ == y" := (pl_strong x y).
```

The only crucial point is about the rules concerning `throw` and `try_catch` blocks. For the related discussion, see Figure 6.9.

6.6 Translating $\mathcal{L}_{exc-pl}$ into $\mathcal{L}_{exc}$ in Coq

The terms of the logic $\mathcal{L}_{exc-pl}$ can simply be translated into the logic $\mathcal{L}_{exc}$ as follows:

```
Fixpoint translate X Y (t: termpl X Y): (term X Y) :=
 match t with
   | pl_tpure X Y f      ⇒ tpure f
   | pl_comp _ _ _ a b   ⇒ (@translate _ _ a) o (@translate _ _ b)
   | throw Y e           ⇒ (@empty Y) o tag e
   | try_catch X Y e a b ⇒ downcast(copair (@id Y) ((@translate _ _ b) o untag e) o coproj1 o (@translate _ _ a))
 end.
```

Any pure term in the logic $\mathcal{L}_{exc-pl}$ is still pure in the logic $\mathcal{L}_{exc}$. The term compositions in the logic $\mathcal{L}_{exc-pl}$ corresponds to the composition of translated terms in the logic $\mathcal{L}_{exc}$. We translate the terms `throw` and `try/catch` as they are given in Definitions 6.3.2 and 6.3.3. This translation is used to prove Theorem 6.3.4.

Remark 6.6.1. See the source `Terms.v` for related implementation details.

6.7 Proofs involving the exceptions effect

In this section, we detail some primitive program properties with the exceptions effect and prove them in a decorated setting (as done in Section 5.3 for programs with the state effect). We also provide corresponding formalizations in Coq.

$(1)_d$ atu. *Untagging an exception of name e and then raising it, is just like doing nothing.*
$\forall\, e \in EName$, $\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \equiv id_{\mathrm{O}}^{(0)} : \mathrm{O} \to \mathrm{O}$.

$(2)_d$ cuu. *Untagging two distinct exception names can be done in any order.*
$\forall\, e \neq r \in EName,\ (\mathtt{untag}_e +_r id_{EV_r})^{(2)} \circ inr^{(0)} \circ \mathtt{untag}_r^{(2)} \equiv$
$(id_{EV_e} +_l \mathtt{untag}_r)^{(2)} \circ inl^{(0)} \circ \mathtt{untag}_e^{(2)} : \mathbf{O} \to EV_e + EV_r.$

$(3)_d$ ppt. *A propagator term always propagates an exception.*
$\forall\, e \in EName,\ a^{(1)} : X \to Y,\ \ a^{(1)} \circ [\ ]_X^{(0)} \circ \mathtt{tag}_e^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} : EV_e \to Y.$

$(4)_d$ rcv. *The parameter used for throwing an exception may be recovered.*
$\Big(\forall\, f^{(1)}, g^{(1)} : X \to \mathbf{O},\ \ [\ ]_Y^{(0)} \circ f^{(1)} \equiv [\ ]_Y^{(0)} \circ g^{(1)} \implies f^{(1)} \equiv g^{(1)}\Big) \implies$
$\Big(\forall\, e \in EName,\ u_1^{(0)}, u_2^{(0)} : X \to EV_e,$
$\big([\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u_1^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u_2^{(0)}\big) \implies u_1^{(0)} \equiv u_2^{(0)}\Big).$

$(5)_d$ try. *The strong equation is compatible with* `try/catch`.
$\forall\, e \in EName,\ a_1^{(1)}, a_2^{(1)} : X \to Y,\ b^{(1)} : EV_e \to Y,\ a_1^{(1)} \equiv a_2^{(1)} \implies$
$\Big(\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_1\big)^{(1)} \equiv$
$\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_2\big)^{(1)}\Big).$

$(6)_d$ try$_0$. *Pure code inside* `try` *never triggers the code inside* `catch`.
$\forall\, e \in EName,\ u^{(0)} : X \to Y,\ b^{(1)} : EV_e \to Y,$
$\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ u\big)^{(1)} \equiv id_Y^{(0)} \circ u^{(0)} : X \to Y.$

$(7)_d$ try$_1$. *The code inside* `catch` *part is executed as soon as an exception is thrown inside* `try`.
$\forall\, e \in EName,\ u^{(0)} : X \to EV_e,\ b^{(1)} : EV_e \to Y,$
$\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ [\ ]_Y \circ \mathtt{tag}_e \circ u\big)^{(1)} \equiv b^{(1)} \circ u^{(0)} : X \to Y.$

$(8)_d$ try$_2$. *An exception cannot be handled, if the particular exception name is not matched. The exception is propagated.*
$\forall\ (e \neq f) \in EName,\ u^{(0)} : X \to EV_f,\ b^{(1)} : EV_e \to Y,$
$\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ [\ ]_Y \circ \mathtt{tag}_f \circ u\big)^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)} : X \to Y.$

The decorated logic for exceptions ($\mathcal{L}_{exc}$) is used to prove above the stated properties. Such proofs are enriched with Coq certifications. Within the Coq scripts, one can simply relate the Coq proof to the proof in English by following the comments following crucial steps. The use of *associativity of composition* in the Coq proofs just balances the proof tree into an intended shape. This is omitted in the proofs on the paper.

Proposition 6.7.1. Annihilation tag-untag (atu). *Untagging an exception of name e and then raising it, is just like doing nothing.*

$$\forall\, e \in EName,\ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \equiv id_{\mathbf{O}}^{(0)} : \mathbf{O} \to \mathbf{O} \tag{6.1}$$

Proof. (1) Due to (eax$_1$), we have $\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim id_{V_e}^{(0)}$. Thanks to the (wrepl), we obtain $\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim \mathtt{tag}_e^{(1)} \circ id_{V_e}^{(0)}$. This can be written as $\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim id_{\mathbf{O}}^{(0)} \circ \mathtt{tag}_e^{(1)}$ by respectively using (ids) and (idt).

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim id_{EV_e}^{(0)}}\,(\forall\ e \in EName)\ (\mathrm{eax}_1)}{\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim \mathtt{tag}_e^{(1)} \circ id_{EV_e}^{(0)}}\,(\mathrm{wrepl})}{\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim \mathtt{tag}_e^{(1)}}\,(\mathrm{ids})}{\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim id_{\mathbf{O}}^{(0)} \circ \mathtt{tag}_e^{(1)}}\,(\mathrm{idt})$$

(2) We have $\mathtt{tag}_e^{(1)} \circ [\ ]_{V_e}^{(0)} \equiv id_{\mathrm{O}}^{(0)}$ provided by (s-empty). Thanks to (replsubs), we get $\mathtt{tag}_e^{(1)} \circ [\ ]_{V_e}^{(0)} \circ \mathtt{tag}_r^{(1)} \equiv id_{\mathrm{O}}^{(0)} \circ \mathtt{tag}_r^{(1)}$, for each exception name r such that $e \neq r$. It is allowed to switch from the strong equation into a weak equation: $\mathtt{tag}_e^{(1)} \circ [\ ]_{V_e}^{(0)} \circ \mathtt{tag}_r^{(1)} \sim id_{\mathrm{O}}^{(0)} \circ \mathtt{tag}_r^{(1)}$. The use of (eax$_2$) on the left, enabled by (wrepl), yields $\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_r^{(1)} \sim id_{\mathrm{O}}^{(0)} \circ \mathtt{tag}_r^{(1)}$.

$$\text{(wrepl)}\ \frac{\text{(ax}_2\text{)}\ \dfrac{\forall\ \mathrm{e}\ \mathrm{r} \in \mathrm{EName\ s.t.\ e} \neq \mathrm{r}}{\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_r^{(1)} \sim [\ ]_{EV_e} \circ \mathtt{tag}_r^{(1)}}}{\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_r^{(1)} \sim \mathtt{tag}_e^{(1)} \circ [\ ]_{EV_e} \circ \mathtt{tag}_r^{(1)}} \qquad \text{(s-empty)}\ \frac{\vdots}{\mathtt{tag}_e^{(1)} \circ [\ ]_{EV_e} \equiv id_{\mathrm{O}}^{(0)}}$$
$$\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_r^{(1)} \sim id_{\mathrm{O}}^{(0)} \circ \mathtt{tag}_r^{(1)}$$

Given items (1), (2) and the rule (elocal-global), we end up with $\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \equiv id_{\mathrm{O}}^{(0)}$. □

In addition to the above proof, it is possible to start from the goal statement itself and end up with some truth value. This is actually constructing the proof tree with a bottom-up strategy. For instance, let us consider the above statement (atu): we start with applying the (local-global) rule and proceed as follows:

(1) for any $r \in \mathit{EName}$, when $e = r$, the goal looks like $\mathtt{tag}\, e \circ \mathtt{untag}\, e \circ \mathtt{tag}\, e \sim id_{\mathrm{O}} \circ \mathtt{tag}\, e$.

(1.1) we apply (idt) and (ids) to obtain $\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim \mathtt{tag}_e^{(1)} \circ id_{EV_e}^{(0)}$.

(1.2) by applying (wrepl), we get $\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim id_{EV_e}^{(0)}$. Finally, the application of (eax$_1$) resolves the goal.

(2) when $e \neq r$, the goal becomes $\mathtt{tag}\, e \circ \mathtt{untag}\, e \circ \mathtt{tag}\, r \sim id_{\mathrm{O}} \circ \mathtt{tag}\, r$.

(2.1) thanks to (s-empty), we have $\mathtt{tag}\, e \circ [\ ]_{EV_e} \equiv id_{\mathrm{O}}$, thus $\mathtt{tag}_e^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_r^{(1)} \sim \mathtt{tag}_e^{(1)} \circ [\ ]_{EV_e} \circ \mathtt{tag}_r^{(1)}$.

(2.1) by applying (wrepl) we get $\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_r^{(1)} \sim [\ ]_{EV_e} \circ \mathtt{tag}_r^{(1)}$. Finally, the application of (eax$_2$) resolves the goal.

Below, we formalize the statement (atu) together with its certified proof in Coq. This proof proceeds by manipulations on the goal and ends up with "true". Therefore, it follows the same lines as the proof given just above. By using the apparent numbering, one can relate the proof steps in English to the ones in Coq.

```
Lemma ATU: forall e: EName, (tag e) o (untag e) == (@id Empty_set).
Proof.
  intro e.
   apply elocal_global.
    intro r. destruct(Exc_dec r e) as [Ha|Hb]. rewrite Ha.
      (* case e = r *) (* (1) *)
      rewrite idt. setoid_rewrite ←ids at 6. rewrite ←assoc. (* (1.1) *)
      apply wrepl; [reflexivity| apply eax1]. (* (1.2) *)
      (* case e <> r *) (* (2) *)
      cut(tag e o (@empty (Val e)) == (@id (Empty_set))).
        [intro H0| setoid_rewrite s_empty; [reflexivity| edecorate| edecorate]].
      rewrite ←H0. setoid_rewrite ←assoc. (* (2.1) *)
      apply wrepl; [reflexivity| apply eax2; exact Hb]. (* (2.2) *)
Qed.
```

Proposition 6.7.2. Commutation untag-untag (cuu). *Untagging two distinct exception names can be done in any order.*

$$\begin{aligned}\forall\, t \neq s \in \mathit{EName},\ (\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \equiv \\ (\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \colon \mathbb{O} \to EV_t + EV_s\end{aligned} \tag{6.2}$$

Proof. (1) We have $(\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \sim \mathit{inr}^{(0)} \circ \mathit{id}^{(0)}_{EV_s}$ due to (s-rcoprod-eq). We can use (ids) to have $(\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathit{id}^{(0)}_{EV_s} \sim \mathit{inr}^{(0)} \circ \mathit{id}^{(0)}_{EV_s}$. Thanks to ($\mathrm{eax}_1$), enabled by (wrepl) on both sides, we get $(\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_s^{(1)} \sim \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_s^{(1)}$. Thanks to (s-lcoprod-eq), we obtain $(\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_s^{(1)} \sim (\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{tag}_s^{(1)}$. The use of (s-empty) yields $(\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_s^{(1)} \sim (\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ [\ ]^{(0)}_{EV_t} \circ \mathtt{tag}_s^{(1)}$. We now obtain by ($\mathrm{eax}_2$), enabled by (wrepl) on the right, that $(\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_s^{(1)} \sim (\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_s^{(1)}$.

(2) Symmetrically, there is $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \sim \mathit{inl}^{(0)} \circ \mathit{id}^{(0)}_{EV_t}$ due to (w-lcoprod-eq). We use (ids) to handle $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathit{id}^{(0)}_{EV_t} \sim \mathit{inl}^{(0)} \circ \mathit{id}^{(0)}_{EV_t}$. Now, by ($\mathrm{eax}_1$), on both sides we get $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_t^{(1)} \sim \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_t^{(1)}$. Thanks to (s-rcoprod-eq), we get $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_t^{(1)} \sim (\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{tag}_t^{(1)}$. It follows the use of (s-empty) that $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_t^{(1)} \sim (\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ [\ ]^{(0)}_{EV_s} \circ \mathtt{tag}_t^{(1)}$. We get by ($\mathrm{eax}_2$) on the right that $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_t^{(1)} \sim (\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_t^{(1)}$.

(3) In addition, by (s-empty), one has $\mathit{inr}^{(0)} \circ [\ ]^{(0)}_{EV_s} \equiv \mathit{inl}^{(0)} \circ [\ ]^{(0)}_{EV_t}$. It is possible to get $\mathit{inr}^{(0)} \circ [\ ]^{(0)}_{EV_s} \circ \mathtt{tag}_r^{(1)} \equiv \mathit{inl}^{(0)} \circ [\ ]^{(0)}_{EV_t} \circ \mathtt{tag}_r^{(1)}$, for each exception name r such that $r \neq s$ and $r \neq t$. It is free to switch from the strong equation into a weak equation: $\mathit{inr}^{(0)} \circ [\ ]^{(0)}_{EV_s} \circ \mathtt{tag}_r^{(1)} \sim \mathit{inl}^{(0)} \circ [\ ]^{(0)}_{EV_t} \circ \mathtt{tag}_r^{(1)}$. Here, we make use of ($\mathrm{eax}_2$), allowed by (wrepl) on both sides, so as to get $\mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_r^{(1)} \sim \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_r^{(1)}$. Thanks to (s-lcoprod-eq), we obtain $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{tag}_r^{(1)} \sim \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_r^{(1)}$ and symmetrically, (s-rcoprod-eq) yields $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{tag}_r^{(1)} \sim (\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{tag}_r^{(1)}$. Now, by (s-empty) on both sides, we get $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ [\ ]^{(0)}_{EV_t} \circ \mathtt{tag}_r^{(1)} \sim (\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ [\ ]^{(0)}_{EV_s} \circ \mathtt{tag}_r^{(1)}$. And by ($\mathrm{eax}_2$) on both sides, we end up with $(\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)} \circ \mathtt{tag}_r^{(1)} \sim (\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \circ \mathtt{tag}_r^{(1)}$.

Now, provided above items (1), (2), (3) and the rule (elocal-global), we conclude that $(\mathtt{untag}_t +_r \mathit{id}_{EV_s})^{(2)} \circ \mathit{inr}^{(0)} \circ \mathtt{untag}_s^{(2)} \equiv (\mathit{id}_{EV_t} +_l \mathtt{untag}_s)^{(2)} \circ \mathit{inl}^{(0)} \circ \mathtt{untag}_t^{(2)}$ for each exception names s and t such that $s \neq t$. □

Proposition 6.7.3. Propagate (ppt). *A propagator term always propagates an exception.*

$$\forall\, e \in \mathit{EName},\ a^{(1)} \colon X \to Y,\ \ a^{(1)} \circ [\ ]^{(0)}_X \circ \mathtt{tag}_e^{(1)} \equiv [\ ]^{(0)}_Y \circ \mathtt{tag}_e^{(1)} \colon EV_e \to Y. \tag{6.3}$$

Proof. For each $a^{(1)} : X \to Y$, (s-empty) implies that $a^{(1)} \circ [\ ]_X^{(0)} \equiv [\ ]_Y^{(0)}$, so that $a^{(1)} \circ [\ ]_X^{(0)} \circ \mathtt{tag}_e^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)}$ by (subs). □

Proposition 6.7.4. Recover (rcv). *The parameter used for throwing an exception may be recovered.*

$$\begin{aligned}\Big(\forall f^{(1)}, g^{(1)} \colon X \to \mathbb{O},\ [\ ]_Y^{(0)} \circ f^{(1)} \equiv [\ ]_Y^{(0)} \circ g^{(1)} \implies f^{(1)} \equiv g^{(1)}\Big) \implies \\ \Big(\forall e \in \mathit{EName},\ u_1^{(0)}, u_2^{(0)} : X \to \mathit{EV}_e, \\ [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u_1^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u_2^{(0)} \implies u_1^{(0)} \equiv u_2^{(0)}\Big).\end{aligned} \tag{6.4}$$

Proof. If $[\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u_1^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u_2^{(0)}$, since $[\ ]_Y$ is a monomorphism with respect to propagators, we have $\mathtt{tag}_e^{(1)} \circ u_1^{(0)} \equiv \mathtt{tag}_e^{(1)} \circ u_2^{(0)}$. Now, (replsubs) gives $\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \circ u_1^{(0)} \equiv \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \circ u_2^{(0)}$. By (stow), we obtain $\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \circ u_1^{(0)} \sim \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \circ u_2^{(0)}$. Since u_1 and u_2 are pure, we are enabled to use (eax_1) on both sides so as to handle $u_1^{(0)} \sim u_2^{(0)}$. Since there is no modifiers, we simply end up with $u_1^{(0)} \equiv u_2^{(0)}$. □

Proposition 6.7.5. try. *The strong equation is compatible with* `try/catch`.

$$\begin{aligned}&\forall e \in \mathit{EName},\ a_1^{(1)}, a_2^{(1)} \colon X \to Y,\ b^{(1)} : \mathit{EV}_e \to Y, \\ a_1^{(1)} \equiv a_2^{(1)} \implies &\Big(\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_1\big)^{(1)} \equiv \\ &\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_2\big)^{(1)}\Big).\end{aligned} \tag{6.5}$$

Proof. Given $a_1^{(1)} \equiv a_2^{(1)}$, we can trivially obtain $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ \mathit{inl}^{(0)} \circ [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ a_1^{(1)} \equiv [id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ \mathit{inl}^{(0)} \circ [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ a_2^{(1)}$ thanks to (replsubs). After the free conversion of the strong equation into a weak one, we can use (w-downcast) to get $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_1\big)^{(1)} \sim \downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_2\big)^{(1)}$. Since both sides are throwers we convert weak equality back into strong: $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_1\big)^{(1)} \equiv \downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ [\ ]_Y \circ \mathtt{tag}_e \circ a_2\big)^{(1)}$. □

Proposition 6.7.6. try_0. *Pure code inside* `try` *never triggers the code inside* `catch`.

$$\begin{aligned}&\forall e \in \mathit{EName},\ u^{(0)} \colon X \to Y,\ b^{(1)} : \mathit{EV}_e \to Y, \\ &\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ u\big)^{(1)} \equiv id_Y^{(0)} \circ u^{(0)} \colon X \to Y.\end{aligned} \tag{6.6}$$

Proof. Due to (w-lcopair-eq), we have $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ \mathit{inl}^{(0)} \sim id_Y^{(0)}$. By using (pw-subs), we obtain $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ \mathit{inl}^{(0)} \circ u^{(0)} \sim id_Y^{(0)} \circ u^{(0)}$. Thanks to (w-downcast) ensuring $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ u\big)^{(1)} \sim [id_Y \mid u \circ \mathtt{untag}_e]_l^{(2)} \circ \mathit{inl}^{(0)} \circ u^{(0)}$, we end up with $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ u\big)^{(1)} \sim id_Y^{(0)} \circ u^{(0)}$ by (w-lcopair-eq). Lack of catchers gives $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ u\big)^{(1)} \equiv id_Y^{(0)} \circ u^{(0)}$. □

Proposition 6.7.7. try_1. *The code inside* `catch` *is executed as soon as an exception is thrown inside* `try`.

$$\begin{aligned}&\forall e \in \mathit{EName},\ u^{(0)} \colon X \to \mathit{EV}_e,\ b^{(1)} : \mathit{EV}_e \to Y, \\ &\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ \mathit{inl} \circ [\ ]_Y \circ \mathtt{tag}_e \circ u\big)^{(1)} \equiv b^{(1)} \circ u^{(0)} \colon X \to Y.\end{aligned} \tag{6.7}$$

Proof. Thanks to (eax$_1$), we have $\mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \sim id_{V_e}$. Due to (pwsubs) and (wrepl), we get $b^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_e^{(1)} \circ u^{(0)} \sim b^{(1)} \circ id_{V_e} \circ u^{(0)}$. Besides, (s-lcopair-eq) yields $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ inr \equiv b^{(1)} \circ \mathtt{untag}_e^{(2)}$. So that we obtain $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ inr^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u^{(0)} \sim b^{(1)} \circ u^{(0)}$. Alongside these, with (s-empty), we handle $inr^{(0)} \equiv inr^{(0)} \circ [\ ]_Y^{(0)}$. So that we get $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ inl^{(0)} \circ [\ ]_Y^{(0)} \circ \mathtt{tag}_e^{(1)} \circ u^{(0)} \sim b^{(1)} \circ u^{(0)}$. Now, (w-downcast) yields $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ [\ ]_Y \circ \mathtt{tag}_e \circ u\big)^{(1)} \sim b^{(1)} \circ u^{(0)}$. The lack of catchers gives $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ [\ ]_Y \circ \mathtt{tag}_e \circ u\big)^{(1)} \equiv b^{(1)} \circ u^{(0)}$. □

Proposition 6.7.8. try$_2$. *An exception cannot be handled, if the particular exception name is not matched. The exception is propagated.*

$$\begin{aligned} &\forall\ (e \neq f) \in EName,\ u^{(0)} \colon X \to EV_f,\ b^{(1)} : EV_e \to Y, \\ &\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l \circ inl \circ [\ ]_Y \circ \mathtt{tag}_f \circ u\big)^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)} \colon X \to Y. \end{aligned} \tag{6.8}$$

Proof. Due to (s-lcopair-eq), we have $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ inr_{Y,\mathbf{O}}^{(0)} \equiv b^{(1)} \circ \mathtt{untag}_e^{(2)}$. By using (subs), we get $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ inr_{Y,\mathbf{O}}^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u \equiv b^{(1)} \circ \mathtt{untag}_e^{(2)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)}$. Here, we first convert the strong equation into a weak equation and then use (eax$_2$) on the right side, since u is pure, so as to obtain $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ inr_{Y,\mathbf{O}}^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)} \sim b^{(1)} \circ [\ ]_{EV_f}^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)}$. Since $b^{(1)} \circ [\ ]_{EV_f}^{(0)} \equiv [\ ]_Y^{(0)}$, due to (s-empty), we get $[id_Y \mid b \circ \mathtt{untag}_e]_l^{(2)} \circ inr_{Y,\mathbf{O}}^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)} \sim [\ ]_Y^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)}$. Now, the rule (w-downcast) yields $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l\big)^{(1)} \circ inr_{Y,\mathbf{O}}^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)} \sim [\ ]_Y^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)}$. The lack of catchers gives $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l\big)^{(1)} \circ inr_{Y,\mathbf{O}}^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)}$. We have $inr_{Y,\mathbf{O}}^{(0)} \equiv inl_{Y,\mathbf{O}}^{(0)} \circ [\ ]_Y^{(0)}$, thanks to (s-empty). Therefore, $\downarrow\big([id_Y \mid b \circ \mathtt{untag}_e]_l\big)^{(1)} \circ inl_{Y,\mathbf{O}}^{(0)} \circ [\ ]_Y^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}_f^{(1)} \circ u^{(0)}$. □

Remark 6.7.9. See the source `Proofs.v` for related implementation details.

6.8 Hilbert-Post completeness for the logic $\mathcal{L}_{exc-pl}$

The *pure sublogic* $\mathcal{L}_{exc-pl}^{(0)}$, for dealing with pure terms, can be seen as any logic extending a *monadic equational logic* $\mathcal{L}_{meq}$. For instance, $\mathcal{L}_{exc-pl}^{(0)}$ may be an equational logic, with n-ary operations for arbitrary n. However, the rules for $\mathcal{L}_{exc-pl}$ do not allow to form tuples of decorated terms, so that the term $\mathtt{op}(f, g)$ (where op is a pure operation of arity 2) is not well-formed, unless f and g are pure. It is well known that there is no "canonical" interpretation for such terms; however, the interpretation where f is runned before g can be formalized thanks to strong monads [Mog91] or sequential products [DDR11]. In this chapter, in order to focus on completeness issues, we avoid such situations.

This pure sublogic $\mathcal{L}_{exc-pl}^{(0)}$ is extended to form the corresponding *decorated logic for the programmers' language for exceptions* $\mathcal{L}_{exc-pl}$ by applying the rules in Figure 6.9, followed by the intended meanings.

The *theory of programmers' language for the exception* $\mathcal{T}_{exc-pl}$ is the theory of $\mathcal{L}_{exc-pl}$ generated from some chosen theory $\mathcal{T}^{(0)}$ of $\mathcal{L}^{(0)}_{exc-pl}$; with the notations of Section 4.5, $\mathcal{T}_{exc-pl} = F(\mathcal{T}^{(0)})$. The soundness of the intended model follows, see, e.g., [DDEP14, §5.1] and [DDFR12b], with the description of the handling of exceptions in Java, see for instance [GJSB05, Ch. 14], or in C++ [Dra12, §15]. Now, in order to prove the completeness of the decorated theory for exceptions under suitable assumptions, we first determine canonical forms and then we study the equations between terms in canonical forms.

Remark 6.8.1. Note also that Coq certifications of Hilbert-Post completeness proof, presented in this section, can be found in the package `hp-thesis`: `https://forge.imag.fr/frs/download.php/696/HPC-THESIS.tar.gz`. See the `HPCompletenessCoq.v` source file inside the `exc_pl-hp` folder.

Proposition 6.8.2. *For each term $a^{(1)} : X \to Y$, either there exists some pure term $u^{(0)} : X \to Y$ such that $a \equiv u$ or there exists some pure term $u^{(0)} : X \to EV$ such that $a \equiv \mathtt{throw}_Y \circ u$.*

Proof. The proof proceeds by structural induction. If a is pure the result is obvious, otherwise a can be written in a unique way as $a = b \circ \mathtt{op} \circ v$ where v is pure, $\mathtt{op}$ is either $\mathtt{throw}_Z$ for some Z or $\mathtt{try}(c)\mathtt{catch}(d)$ for some c and d, and b is the remaining part of a.

- If $a = b^{(1)} \circ \mathtt{throw}_Z \circ v^{(0)}$, then by (propagate) $a \equiv \mathtt{throw}_Y \circ v^{(0)}$.
- If $a = b^{(1)} \circ (\mathtt{try}(c^{(1)})\mathtt{catch}(d^{(1)})) \circ v^{(0)}$, then by induction we consider two subcases.
  - If $c \equiv w^{(0)}$ then by (try_0) $a \equiv b^{(1)} \circ w^{(0)} \circ v^{(0)}$ and by induction we consider two subcases: if $b \equiv t^{(0)}$ then $a \equiv (t \circ w \circ v)^{(0)}$ and if $b \equiv \mathtt{throw}_Y \circ t^{(0)}$ then $a \equiv \mathtt{throw}_Y \circ (t \circ w \circ v)^{(0)}$.
  - If $c \equiv \mathtt{throw}_Z \circ w^{(0)}$ then by (try_1) $a \equiv b^{(1)} \circ d^{(1)} \circ w^{(0)} \circ v^{(0)}$ and by induction we consider two subcases: if $b \circ d \equiv t^{(0)}$ then $a \equiv (t \circ w \circ v)^{(0)}$ and if $b \circ d \equiv \mathtt{throw}_Y \circ t^{(0)}$ then $a \equiv \mathtt{throw}_Y \circ (t \circ w \circ v)^{(0)}$. □

Thanks to Proposition 6.8.2, in order to study equations in the logic $\mathcal{L}_{exc-pl}$ we may restrict our study to pure terms and to propagators of the form $\mathtt{throw}_Y \circ v$ where v is pure.

Proposition 6.8.3. *For all $v_1^{(0)}, v_2^{(0)} : X \to EV$ let $a_1^{(1)} = \mathtt{throw}_Y \circ v_1 : X \to Y$ and $a_2^{(1)} = \mathtt{throw}_Y \circ v_2 : X \to Y$. Then $a_1^{(1)} \equiv a_2^{(1)} \iff v_1^{(0)} \equiv v_2^{(0)}$.*

Proof. Clearly, if $v_1^{(0)} \equiv v_2^{(0)}$ then $a_1^{(1)} \equiv a_2^{(1)}$. Conversely, if $a_1 \equiv a_2$, i.e., if $\mathtt{throw}_Y \circ v_1^{(0)} \equiv \mathtt{throw}_Y \circ v_2^{(0)}$, then by rule (recover) it follows that $v_1^{(0)} \equiv v_2^{(0)}$. □

Assumption 6.8.4. In the logic $\mathcal{L}_{exc-pl}$, for all $v_1^{(0)} \colon X \to EV$, $v_2^{(0)} \colon X \to Y$, let $a_1^{(1)} = \mathtt{throw}_Y^{(1)} \circ v_1^{(0)} \colon X \to Y$. Then, $a_1^{(1)} \equiv v_2^{(0)} \iff$ for all $f^{(0)}, g^{(0)} \colon X \to Y$, $f^{(0)} \equiv g^{(0)}$.

Let $\mathscr{C}$ be the category of sets. Thanks to item (1) in Proposition 4.2.4 and point (6) in Definition 6.2.1, $\mathtt{throw}_Y^{(1)} \circ v_1^{(0)} \equiv v_2^{(0)} \colon X \to Y$ is interpreted as $\mu_Y \circ T(\mathtt{throw}_Y) \circ \eta_{EV} \circ v_1 = \eta_Y \circ v_2 \colon X \to Y + E$ in $\mathscr{C}$. It is not possible to have this equality for some $x \in X$, since the left hand side is in the E summand while the right hand side is in Y summand of the disjoint union $Y + E$. This means that these two sides are distinct as soon as their domain X is interpreted as a non-empty set. If the interpretation of X is the

empty set, then both sides of the assumption are true: $\forall f, g : \emptyset \to Y, f = g$. Similarly, $\forall a_1 : \emptyset \to Y + E, v_2 : \emptyset \to Y, a_1 = \eta_Y \circ v_2$. For this reason, Assumption 6.8.4 is sound.

Theorem 6.8.5. *Under Assumption 6.8.4, the theory of exceptions $\mathcal{T}_{exc-pl}$ is relatively Hilbert-Post complete with respect to the pure sublogic $\mathcal{L}^{(0)}_{exc-pl}$ of $\mathcal{L}_{exc-pl}$.*

Proof. Using Corollary 4.5.10, the proof relies upon Propositions 6.8.2, 6.8.3. The theory $\mathcal{T}_{exc-pl}$ is consistent: it cannot be proved that $\mathtt{throw}^{(1)}_{EV} \equiv id^{(0)}_{EV}$ because the logic $\mathcal{L}_{exc-pl}$ is sound with respect to its intended model and the interpretation of this equation in the intended model is false: indeed, $\mathtt{throw}_{EV}(p) \in E$ for each $p \in EV$, and since $EV + E$ is a disjoint union we have $\mathtt{throw}_{EV}(p) \neq p$. Propositions 6.8.2 and 6.8.3 together with Assumption 6.8.4 prove that the given equation is equivalent to a set of pure equations. □

6.9 Hilbert-Post completeness for the logic $\mathcal{L}_{exc}$

The logic $\mathcal{L}_{exc}$ is precisely introduced and its categorical interpretation is studied in Section 6.1. Let the logic $\mathcal{L}_{exc-\oplus}$ be a variant of $\mathcal{L}_{exc}$ obtained by dropping the categorical copairs/coproducts. Let the logic $\mathcal{L}_{meq+\mathbb{O}}$ be an extension to $\mathcal{L}_{meq}$ with the use of empty ($\mathbb{O}$) type and the following inference rules: $\frac{X}{[\,]_X : \mathbb{O} \to X}$ and $\frac{f : \mathbb{O} \to X}{f \cong [\,]_X}$. Now, the core theory of exceptions $\mathcal{T}_{exc}$ is defined as a theory of the logic $\mathcal{L}_{exc-\oplus}$ generated from the fundamental equation $\mathtt{untag}^{(2)}_e \circ \mathtt{tag}^{(1)}_e \sim id^{(0)}_{EV_e}$ and from some consistent theory $\mathcal{T}_{eq}$ of the logic $\mathcal{L}_{meq+\mathbb{O}}$; with the notations of Section 4.5, $\mathcal{T}_{exc} = F(\mathcal{T}_{eq})$. In this section, we prove that the theory $\mathcal{T}_{exc}$ of the logic $\mathcal{L}_{exc-\oplus}$ (not of the logic $\mathcal{L}_{exc}$) is Hilbert-Post complete with respect to the logic $\mathcal{L}_{meq+\mathbb{O}}$.

Remark 6.9.1. Note that Coq certifications of the Hilbert-Post completeness proof, presented in this section, can be found in the package `hp-thesis`: `https://forge.imag.fr/frs/download.php/696/HPC-THESIS.tar.gz`. Check out the `HPCompletenessCoq.v` file inside the `exc_cl-hp` folder. Our main result is Theorem 6.9.9 about the relative Hilbert-Post completeness of the decorated theory $\mathcal{T}_{exc}$ of exceptions under suitable assumptions. It is assumed that there is only one exception name e and we write EV, `tag` and `untag` instead of EV_e, $\mathtt{tag}^{(1)}_e$ and $\mathtt{untag}^{(2)}_e$.The study of completeness proof with the signature including several exception names and coproducts is considered as a future goal.

Note also that we do not explicitly have the relative Hilbert-Post completeness (rHPC) formalization in Coq. Thanks to the second characterization of rHPC given in Corollary 4.5.10, it suffices to show that any formula e in the logic $\mathcal{L}_{exc-\oplus}$ is ($\mathcal{T}_{exc}$-)equivalent to some set of formulae E_0 in the logic $\mathcal{L}_{meq+\mathbb{O}}$:

$$\mathcal{T}_{exc} + Th(E_0) = \mathcal{T}_{exc} + Th(e).$$

This has been checked in Coq.

Lemma 6.9.2. 1. *The fundamental strong equation for exceptions is $\mathtt{tag} \circ \mathtt{untag} \equiv id^{(0)}_{\mathbb{O}}$.*

2. *For all pure terms $u^{(0)}_1, u^{(0)}_2 : X \to EV$, one has: $u^{(0)}_1 \equiv u^{(0)}_2 \iff \mathtt{tag} \circ u^{(0)}_1 \equiv \mathtt{tag} \circ u^{(0)}_2 \iff \mathtt{untag} \circ \mathtt{tag} \circ u^{(0)}_1 \equiv \mathtt{untag} \circ \mathtt{tag} \circ u^{(0)}_2$.*

3. *For each pair of catchers* $f_1^{(2)}\ f_2^{(2)} \colon EV \to Y$, $f_1^{(2)} \circ \mathtt{untag} \equiv f_2^{(2)} \circ \mathtt{untag} \iff f_1^{(2)} \sim f_2^{(2)}$.

Proof. 1. By rewriting the axiom (eax_1), we get $\mathtt{tag} \circ \mathtt{untag} \circ \mathtt{tag} \sim \mathtt{tag}$; then by rule (elocal-global) $\mathtt{tag} \circ \mathtt{untag} \equiv id_{\mathbb{O}}^{(0)}$.

2. Implications from left to right are clear. Conversely, given $\mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$, we first convert the strong equation into a weak equation, then use rules (eax_1) and (wsubs) so as to get $u_1^{(0)} \sim u_2^{(0)}$. Since u_1 and u_2 are pure, we obtain $u_1^{(0)} \equiv u_2^{(0)}$.

3. Assuming that $f_1^{(2)} \circ \mathtt{untag} \equiv f_2^{(2)} \circ \mathtt{untag}$, we have $f_1^{(2)} \circ \mathtt{untag} \circ \mathtt{tag} \equiv f_2^{(2)} \circ \mathtt{untag} \circ \mathtt{tag}$, thanks to (replsubs). We convert the strong equation into a weak equation and apply (eax_1) on both sides to get $f_1^{(2)} \sim f_2^{(2)}$. Conversely, given $f_1^{(2)} \sim f_2^{(2)}$, we have $f_1^{(2)} \circ id_{EV}^{(0)} \sim f_2^{(2)} \circ id_{EV}^{(0)}$, thanks to (pwsubs). Now, by making use of (eax_1) on both sides, we get $f_1^{(2)} \circ \mathtt{untag} \circ \mathtt{tag} \sim f_2^{(2)} \circ \mathtt{untag} \circ \mathtt{tag}$. Due to (elocal-global), we end up with $f_1^{(2)} \circ \mathtt{untag} \equiv f_2^{(2)} \circ \mathtt{untag}$.

□

Proposition 6.9.3. *1. For each propagator* $a^{(1)} : X \to Y$*, either* a *is pure or there is a pure term* $v^{(0)} : X \to EV$ *and an accessor* $u^{(1)} \colon \mathbb{O} \to Y$ *such that* $a^{(1)} \equiv u^{(1)} \circ \mathtt{tag} \circ v^{(0)}$.

2. *For each catcher* $f^{(2)} : X \to Y$*, either* f *is a propagator or there is a propagator* $a^{(1)} : EV \to Y$ *and a pure term* $u^{(0)} : X \to EV$ *such that* $f^{(2)} \equiv a^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v^{(0)}$.

Proof. 1. The proof proceeds by structural induction. If a is pure, then the result is obvious. If $a = \mathtt{tag}$, then it follows that $\mathtt{tag} \equiv id_{\mathbb{O}}^{(1)} \circ \mathtt{tag} \circ id_{EV}^{(0)}$. Otherwise, a can be written as $a = a_1^{(1)} \circ a_2^{(1)}$ such that $a_1 \colon Z \to Y$ and $a_2 \colon X \to Z$. By induction a_1 and a_2 are either pure or $a_1 \equiv u_1^{(1)} \circ \mathtt{tag} \circ v_1^{(0)}$ and $a_2 \equiv u_2^{(1)} \circ \mathtt{tag} \circ v_2^{(0)}$ for some pure terms $v_1 \colon Z \to EV$ and $v_2 \colon X \to EV$ and some propagators $u_1 \colon \mathbb{O} \to Y$ and $u_2 \colon \mathbb{O} \to Z$. Thus, there are four cases to consider:

(1.1) If both a_1 and a_2 are pure, then a is.

(1.2) If a_1 is pure while a_2 is a propagator, we get $a \equiv (a_1 \circ u_2)^{(1)} \circ \mathtt{tag} \circ v_2^{(0)}$.

(1.3) Symmetrically when a_2 is pure while a_1 is a propagator, we get $a \equiv u_1^{(1)} \circ \mathtt{tag} \circ (v_1 \circ a_2)^{(0)}$.

(1.4) If both are propagators, then $a \equiv u_1^{(1)} \circ \mathtt{tag} \circ v_1^{(0)} \circ u_2^{(1)} \circ \mathtt{tag} \circ v_2^{(0)}$. Thanks to (s-empty), we get $\mathtt{tag} \circ v_1^{(0)} \circ u_2^{(1)} \equiv id_{\mathbb{O}}^{(0)}$. The use of (replsubs) yields $u_1^{(1)} \circ \mathtt{tag} \circ v_1^{(0)} \circ u_2^{(1)} \circ \mathtt{tag} \circ v_2^{(0)} \equiv u_1^{(1)} \circ \mathtt{tag} \circ v_2^{(0)}$. Hence, we obtain $a \equiv u_1^{(1)} \circ \mathtt{tag} \circ v_2^{(0)}$.

2. The proof proceeds by structural induction. If f is a propagator, then the result is obvious. If $f = \mathtt{untag}$, then it follows that $\mathtt{untag} \equiv id_{EV}^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ [\ ]_{EV}$ (notice that $\mathtt{tag} \circ [\ ]_{EV} \equiv id_{\mathbb{O}}^{(0)}$ due to (s-empty)). Otherwise f can be written as $f = f_1^{(2)} \circ f_2^{(2)}$ such that $f_1 \colon Z \to Y$ and $f_2 \colon X \to Z$. By induction f_1 and f_2 are either propagators or $f_1^{(2)} \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)}$ and $f_2^{(2)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)}$ for

some pure terms $v_1 : Z \to EV$ and $v_2 : X \to EV$ and some propagators $a_1 : EV \to Y$ and $a_2 : EV \to Z$. Thus, there are four cases to consider:

(2.1) If both f_1 and f_2 are propagators, so is f by (comp).

(2.2) If f_1 is a catcher while f_2 is a propagator, we obtain $f \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ f_2^{(1)}$. Thanks to Point 1, $f_2 \equiv u \circ \mathtt{tag} \circ w$ for some pure term $w^{(0)} : X \to EV$ and some propagator $u^{(1)} : \mathbb{O} \to Z$. So that the equation expands into $f \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)}$. Thanks to (s-empty), we get $\mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \equiv id_{\mathbb{O}}^{(0)}$. The use of (replsubs) yields $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)} \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ w^{(0)}$. Thus, we end up with $f \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ w^{(0)}$.

(2.3) Symmetrically when f_2 is a catcher while f_1 is a propagator, we obtain $f \equiv (f_1 \circ a_2)^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)}$.

(2.4) If both are catchers, then $f \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)}$ such that $v_1^{(0)} : Z \to EV$, $v_2^{(0)} : X \to EV$, $a_1^{(1)} : EV \to Y$ and $a_2^{(1)} : EV \to Z$. Now, the reasoning proceeds on $a_2^{(1)}$. Thus, we have two sub-cases:

(2.4.1) Let us first consider the case where a_2 is pure. Since, the composition $v_1^{(0)} \circ a_2^{(0)}$ is pure, due to (pwsubs), we have $\mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ a_2^{(0)} \sim v_1^{(0)} \circ a_2^{(0)}$. Now, we apply Point 3 in Lemma 6.9.2 and obtain $\mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ a_2^{(0)} \circ \mathtt{untag} \equiv v_1^{(0)} \circ a_2^{(0)} \circ \mathtt{untag}$. Thanks to (replsubs), we write $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ a_2^{(0)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)} \equiv a_1^{(1)} \circ v_1^{(0)} \circ a_2^{(0)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)}$. Hence, $f \equiv (a_1 \circ v_1 \circ a_2)^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)}$.

(2.4.2) It remains to consider the case where a_2 is not pure (it has tag). Thanks to Point 1, we obtain $a_2^{(1)} \equiv u^{(1)} \circ \mathtt{tag} \circ w^{(0)}$ such that $u^{(1)} : \mathbb{O} \to Z$ and $w^{(0)} : EV \to EV$. Thus, $f \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)}$. Due to (s-empty), we have $\mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \equiv id_{\mathbb{O}}^{(0)}$. By using (replsubs), we get $\mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)} \equiv \mathtt{tag} \circ w^{(0)}$. We first convert the strong equation into a weak equation and then make use of (wrepl) to obtain $\mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)} \sim \mathtt{untag} \circ \mathtt{tag} \circ w^{(0)}$. Since w is pure, we apply (eax_1) on the right to get $\mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)} \sim w^{(0)}$. Now, we apply Point 3 in Lemma 6.9.2 and obtain $\mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)} \circ \mathtt{untag} \equiv w^{(0)} \circ \mathtt{untag}$. Thanks to (replsubs), we obtain $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1^{(0)} \circ u^{(1)} \circ \mathtt{tag} \circ w^{(0)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)} \equiv (a_1 \circ w)^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v_2^{(0)}$.

□

Corollary 6.9.4. *For each propagator $a^{(1)} \circ X \to Y$, either a is pure or there is a pure term $v^{(0)} : X \to EV$ such that $a^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag} \circ v^{(0)}$.*

Proof. Due to Point 1 in Proposition 6.9.3, a is either pure or it can be written in a unique way as $a = b^{(1)} \circ \mathtt{tag} \circ v^{(0)}$ for some pure term $v^{(0)} : X \to EV$ and some propagator $b^{(1)} : \mathbb{O} \to Y$. Thanks to (s-empty), we have $b^{(1)} \equiv [\,]_Y^{(0)}$, hence the result follows. □

Thanks to Corollary 6.9.4 and Proposition 6.9.3, in order to study equations in the logic $\mathcal{L}_{exc}$, we may restrict our study to pure terms, propagators of the form $[\,]_Y^{(0)} \circ \mathtt{tag} \circ v^{(0)}$ and catchers of the form $a^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u^{(0)}$.

Assumption 6.9.5. For any strict thrower term $a^{(1)} \colon X \to Y$ and pure terms $v_1^{(0)} \colon X \to EV$, $v_2^{(0)} \colon X \to Y$, such that $a^{(1)} = [\,]_Y^{(0)} \circ \mathtt{tag} \circ v^{(0)}$. Then;

$$[\,]_Y^{(0)} \circ \mathtt{tag} \circ v^{(0)} \equiv v_2^{(0)} \iff (\text{for all } f^{(0)}, g^{(0)} \colon X \to Y,\ f^{(0)} \equiv g^{(0)}).$$

Let $\mathscr{C}$ be the category of sets. Thanks to items (2) and (3) in Proposition 4.2.4, $[\,]_Y^{(0)} \circ \mathtt{tag}^{(1)} \circ v^{(0)} \equiv v_2^{(0)} \colon X \to Y$ is interpreted as $T([\,]_Y) \circ \mu_{\mathbb{O}} \circ T(\mathtt{tag}) \circ T(v_1) = T(v_2) \colon X + E \to Y + E$ in $\mathscr{C}$. Given any exceptional argument $e' \in E$, we have $T([\,]_Y) \circ \mu_{\mathbb{O}} \circ T(\mathtt{tag}) \circ T(v_1)(e') = e' = T(v_2)(e')$: both sides propagate the exception e'. Besides, given any ordinary argument $x \in X$, we have $T([\,]_Y) \circ \mu_{\mathbb{O}} \circ T(\mathtt{tag}) \circ T(v_1)(x) = e'$, for some $e' \in E$ and $T(v_2)(x) = y$, for some $y \in Y$. Since, "+" is the disjoint union operator on sets, $T([\,]_Y) \circ \mu_{\mathbb{O}} \circ T(\mathtt{tag}) \circ T(v_1) = T(v_2) \colon X + E \to Y + E$ cannot hold in $\mathscr{C}$: sides agree on exceptional arguments but not on ordinary ones. In other words, with this assumption, we thus mean that if such an equality holds then all pure parallel terms are equal to each other.

Remark 6.9.6. Notice also that Assumption 6.9.5 is the image of Assumption 6.8.4 by the translation given in Section 6.3, so that by Theorem 6.3.4, they are equivalent.

Now, Proposition 6.9.7 shows that

(1) equations between catchers can be reduced to some equations between propagators,

(2) equations between propagators can be reduced to some equations between pure terms.

Proposition 6.9.7. *1. For all $a_1^{(1)}, a_2^{(1)} : EV \to Y$ and $u_1^{(0)}, u_2^{(0)} : X \to EV$, let $f_1^{(2)} = a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} : X \to Y$ and $f_2^{(2)} = a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)} : X \to Y$. Then;*

$$\begin{cases} f_1^{(2)} \sim f_2^{(2)} \iff a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)} \\ f_1^{(2)} \equiv f_2^{(2)} \iff a_1^{(1)} \equiv a_2^{(1)} \ \text{ and } \ a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)}. \end{cases}$$

2. For all $a_1^{(1)} : EV \to Y$, $u_1^{(0)} : X \to EV$ and $a_2^{(1)} : X \to Y$, let $f_1^{(2)} = a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} : X \to Y$. Then;

$$\begin{cases} f_1^{(2)} \sim a_2^{(1)} \iff a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \\ f_1^{(2)} \equiv a_2^{(1)} \iff a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \ \text{ and } \ a_1^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag}. \end{cases}$$

3. Let us assume that $[\,]_Y^{(0)}$ is a monomorphism with respect to propagators. For all $v_1^{(0)}, v_2^{(0)} : X \to EV$, let $a_1^{(1)} = [\,]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)} : X \to Y$ and $a_2^{(1)} = [\,]_Y^{(0)} \circ \mathtt{tag} \circ v_2^{(0)} : X \to Y$. Then;

$$a_1^{(1)} \equiv a_2^{(1)} \iff v_1^{(0)} \equiv v_2^{(0)}.$$

Proof. 1. We have four implications to prove:

(1.1) $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)} \implies a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)}$. Since, u_1 and u_2 are both pure, we apply (eax_1) on both sides. So that we get $a_1^{(1)} \circ u_1^{(0)} \sim a_2^{(1)} \circ u_2^{(0)}$. Due to the lack of catchers, we end up with $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)}$.

(1.2) $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)} \implies a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$. We first convert the strong equation into a weak equation and then using (ids), we get $a_1^{(1)} \circ id_{EV}^{(0)} \circ u_1^{(0)} \sim a_2^{(1)} \circ id_{EV}^{(0)} \circ u_2^{(0)}$. By rewriting ($\text{eax}_1$) on both sides (since u_1 and u_2 are both pure), we end up with $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$.

(1.3) $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)} \implies a_1^{(1)} \equiv a_2^{(1)}$ and $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)}$:

(1.3.1) Given $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$, we get $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \circ [\ ]_X^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)} \circ [\ ]_X^{(0)}$ thanks to (replsubs). Since $\mathtt{tag} \circ u_i^{(0)} \circ [\ ]_X^{(0)} \equiv [\ ]_{\mathrm{O}}^{(0)} \equiv id_{\mathrm{O}}^{(0)}$ due to (s-empty), for each $i \in \{1,2\}$, we obtain $a_1^{(1)} \circ \mathtt{untag} \equiv a_2^{(1)} \circ \mathtt{untag}$. Now, we apply Point 3 in Lemma 6.9.2 to get $a_1^{(1)} \sim a_2^{(1)}$. There is no catchers involved, thus we write $a_1^{(1)} \equiv a_2^{(1)}$.

(1.3.2) First, we convert the strong equation into a weak equation and then apply (eax_1) on both sides (provided that u_1 and u_2 are both pure) so as to get $a_1^{(1)} \circ u_1^{(0)} \sim a_2^{(1)} \circ u_2^{(0)}$. There is no catchers involved so that we conclude with $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)}$.

(1.4) $a_1^{(1)} \equiv a_2^{(1)}$ and $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)} \implies a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$:

(1.4.1) Starting from $a_1^{(1)} \equiv a_2^{(1)}$, thanks to (replsubs), we get $a_1^{(1)} \circ \mathtt{untag} \circ id_{\mathrm{O}}^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ id_{\mathrm{O}}^{(0)}$. Due to (s-empty), we obtain both $\mathtt{tag} \circ u_1^{(0)} \circ [\ ]_X^{(0)} \equiv [\ ]_{\mathrm{O}}^{(0)} \equiv id_{\mathrm{O}}^{(0)}$ and $\mathtt{tag} \circ u_2^{(0)} \circ [\ ]_X^{(0)} \equiv [\ ]_{\mathrm{O}}^{(0)} \equiv id_{\mathrm{O}}^{(0)}$. Hence, $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \circ [\ ]_X^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)} \circ [\ ]_X^{(0)}$.

(1.4.2) Starting from $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)}$, thanks to the use of (ids) on both sides, we get $a_1^{(1)} \circ id_{EV}^{(0)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ id_{EV}^{(0)} \circ u_2^{(0)}$. After converting the strong equation into a weak equation, we apply (eax_1) on both sides, since u_1 and u_2 are pure so as to obtain: $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$.

Now, the (eeffect) rule yields $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$ given both items (1.4.1) and (1.4.2).

2. We again have four cases to prove:

(2.1) $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)} \implies a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)}$: Since, u_1 is pure, we can apply (eax_1) and obtain $a_1^{(1)} \circ u_1^{(0)} \sim a_2^{(1)}$. There is no catchers involved, thus we end up with $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)}$.

(2.2) $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \implies a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)}$: Thanks to (ids), we have $a_1^{(1)} \circ id_{EV}^{(0)} \circ u_1^{(0)} \equiv a_2^{(1)}$. Now, we first convert the strong equation into a weak equation and then apply (eax$_1$), since the term u_1 is pure, so as to get $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)}$.

(2.3) $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)} \implies a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)}$ and $a_1^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}$:

(2.3.1) We first convert the strong equation into a weak equation. Since u_1 is pure, we apply (eax$_1$) and obtain $a_1^{(1)} \circ u_1^{(0)} \sim a_2^{(1)}$. There is no catchers involved, thus we end up with $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)}$.

(2.3.2) Thanks to (replsubs), we get $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \circ [\ ]_X^{(0)} \equiv a_2^{(1)} \circ [\ ]_X^{(0)}$. Due to (s-empty), we have $\mathtt{tag} \circ u_1^{(0)} \circ [\ ]_X^{(0)} \equiv [\ ]_O^{(0)} \equiv id_O^{(0)}$ and $a_2^{(1)} \circ [\ ]_X^{(0)} \equiv [\ ]_Y^{(0)}$. Therefore, $a_1^{(1)} \circ \mathtt{untag} \equiv [\ ]_Y^{(0)}$. Now, the use of (replsubs) gives $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}$. After converting the strong equation into a weak equation, we apply (eax$_1$) and obtain $a_1^{(1)} \sim [\ ]_Y^{(0)} \circ \mathtt{tag}$. The lack of catchers gives $a_1^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}$.

(2.4) $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)}$ and $a_1^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag} \implies a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)}$:

(2.4.1) Starting from $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)}$ first, we have the conversion of the strong equation into a weak equation, then we apply (ids) so as to get $a_1^{(1)} \circ id_{EV}^{(0)} \circ u_1^{(0)} \sim a_2^{(1)}$. Since u_1 is pure, we apply (eax$_1$) on the left and obtain $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \sim a_2^{(1)}$.

(2.4.2) Staring from $a_1^{(1)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag}$, we first obtain $a_1^{(1)} \equiv a_2^{(1)} \circ [\ ]_X^{(0)} \circ \mathtt{tag}$ thanks to (s-empty). By the use of (replsubs), we get $a_1^{(1)} \circ \mathtt{untag} \equiv a_2^{(1)} \circ [\ ]_X^{(0)} \circ \mathtt{tag} \circ \mathtt{untag}$. By using Point 1 in Lemma 6.9.2 we have $a_1^{(1)} \circ \mathtt{untag} \equiv a_2^{(1)} \circ [\ ]_X^{(0)}$. Due to (s-empty) $\mathtt{tag} \circ u_1^{(0)} \circ [\ ]_X^{(0)} \equiv [\ ]_O^{(0)} \equiv id_O^{(0)}$ holds. Therefore, $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \circ [\ ]_X^{(0)} \equiv a_2^{(1)} \circ [\ ]_X^{(0)}$.

Now, the (eeffect) rule yields $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)}$, given above items (2.4.1) and (2.4.2).

3. Recall the statement $[\ ]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)} \implies v_1^{(0)} \equiv v_2^{(0)}$:

(3.1) If $[\ ]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag} \circ v_2^{(0)}$, since $[\ ]_Y^{(0)}$ is a monomorphism with respect to propagators we get $\mathtt{tag} \circ v_1^{(0)} \equiv \mathtt{tag} \circ v_2^{(0)}$. By Point 2 in Lemma 6.9.2, this yields in $v_1^{(0)} \equiv v_2^{(0)}$.

(3.2) Conversely, if $v_1^{(0)} \equiv v_2^{(0)}$ then thanks to (replsubs), we simply get $[\ ]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)} \equiv [\ ]_Y^{(0)} \circ \mathtt{tag} \circ v_2^{(0)}$.

□

Now, Corollary 6.9.8 shows that equations between catchers can be reduced to equations between pure terms. It also makes the proof in Coq easier to read.

Corollary 6.9.8. *Let us assume that* $[\ ]_Y^{(0)}$ *is a monomorphism with respect to propagators. Then:*

1. *For all* $a_1^{(1)}, a_2^{(1)} : X \to Y$, *we have either of below cases:*

(1.1) *when* $X \neq \mathbf{O}$, *we have one of below subcases:*

(1.1.1) $\exists v_1^{(0)}, v_2^{(0)} : X \to EV,\ a_1^{(1)} \equiv a_2^{(1)} \iff v_1^{(0)} \equiv v_2^{(0)}$,

(1.1.2) $\exists v_1^{(0)}, v_2^{(0)} : X \to Y,\ a_1^{(1)} \equiv a_2^{(1)} \iff v_1^{(0)} \equiv v_2^{(0)}$,

(1.1.3) $\exists v^{(0)} : X \to Y,\ \forall i \in \{1, 2\},\ a_i^{(1)} \equiv v^{(0)} \iff$
$\forall f^{(0)}, g^{(0)} : X \to Y\ \ f^{(0)} \equiv g^{(0)}$

(1.2) *when* $X = \mathbf{O}$, *we have:*

$$a_1^{(1)} \equiv a_2^{(1)}.$$

2. *For all* $f_1^{(2)},\ f_2^{(2)} : X \to Y$, *we have either of below subcases:*

(2.1) $\exists a_1^{(1)}, a_2^{(1)} : EV \to Y, \exists b_1^{(1)}, b_2^{(1)} : X \to Y,\ f_1^{(2)} \equiv f_2^{(2)} \iff$
$a_1^{(1)} \equiv a_2^{(1)}$ *and* $b_1^{(1)} \equiv b_2^{(1)}$,

(2.2) $\exists a_1^{(1)}, a_2^{(1)} : X \to Y,\ f_1^{(2)} \equiv f_2^{(2)} \iff a_1^{(1)} \equiv a_2^{(1)}$.

Proof. The proof is immediate from Proposition 6.9.7. See full proof in Appendix B. □

Theorem 6.9.9. *Under Assumption 6.9.5, the theory of exceptions* $\mathcal{T}_{exc}$ *of* $\mathcal{L}_{exc-\oplus}$ *is relatively Hilbert-Post complete with respect to the pure sublogic* $\mathcal{L}_{meq+\mathbf{O}}$.

Proof. Using Corollary 4.5.10, the proof is based upon Corollary 6.9.8. It follows the same lines as the proof of Theorem 6.8.5, except when X is empty: due to catchers, the proof here is slightly more subtle. First, the theory $\mathcal{T}_{exc}$ is consistent: it cannot be proved that $\mathtt{untag}^{(2)} \equiv [\ ]_{EV}^{(0)}$ because the logic $\mathcal{L}_{exc}$ is sound with respect to its intended model and the interpretation of this equation in the intended model is false: indeed, the function $\mathtt{untag} : E \to EV + E$ is such that $\mathtt{untag}(\mathtt{tag}(p)) = p \in EV$ for each $p \in EV$ while $[\ ]_{EV}(e) = e \in E$ for each $e \in E$, which includes $e = \mathtt{tag}(p)$; since $EV + E$ is a disjoint union we have $\mathtt{untag}(e) \neq [\ ]_{EV}(e)$ when $e = \mathtt{tag}(p)$. Now, let us consider an equation between two terms f_1 and f_2 with domain X; we distinguish two cases, whether X is empty or not. When X is non-empty, Corollary 6.9.8 proves that the given equation is equivalent to a finite set of equations between pure terms. When X is empty, then $f_1 \sim [\ ]_Y$ and $f_2 \sim [\ ]_Y$, so that if the equation is weak or if both f_1 and f_2 are propagators, then the given equation is equivalent to the empty set of equations between pure terms. When X is empty and the equation is $f_1 \equiv f_2$ with at least one of f_1 and f_2 a catcher, then by Point 2 of Corollary 6.9.8 the given equation is equivalent to a set of equations between propagators; but we have seen that each equation between propagators (whether X is empty or not) is equivalent to a set of equations between pure terms, so that $f_1 \equiv f_2$ is equivalent to the union of the corresponding sets of pure equations. □

6.10 Chapter summary

In this Chapter;

(1) The logic $\mathcal{L}_{exc}$ has been built as an extension to the logic $\mathcal{L}_{mon}$ and interpreted interpreted via the *coKleisli-on-Kleisli construction* applied to the exceptions monad.

(2) $\mathcal{L}_{exc-pl}$ has been established as an extension to $\mathcal{L}_{mon}$ and interpreted via the *Kleisli adjunction associated to a monad* applied to the exceptions monad.

(3) The logics $\mathcal{L}_{exc}$ and $\mathcal{L}_{exc-pl}$ have been formalized in Coq and these formalizations have been used to prove properties of programs with features to handle exceptions.

(4) The base language of the logic $\mathcal{L}_{exc}$ (with no use of coproducts) and the language of $\mathcal{L}_{exc-pl}$ have been proven to be Hilbert-Post complete (for a single exception name) and these proofs have been checked in Coq.

7
Conclusions

7.1 Summary

In this thesis, as extensions to the monadic equational logic ($\mathcal{L}_{meq}$), we have presented the decorated logic for a comonad ($\mathcal{L}_{com}$) and a monad ($\mathcal{L}_{mon}$). The former has been extended into the decorated logic for the state ($\mathcal{L}_{st}$) and the latter into the decorated logic for exceptions ($\mathcal{L}_{exc}$). The logics $\mathcal{L}_{st}$ and $\mathcal{L}_{exc}$ have been used to formalize the state and the exceptions effects, respectively. We have also introduced the decorated logic for the programmers' language of exceptions ($\mathcal{L}_{exc-pl}$). It has been translated into the logic $\mathcal{L}_{exc}$ and the translation has been proven to be correct. We have also given categorical interpretations of all these logics. Besides, the logic $\mathcal{L}_{st}$ (without products) and the logic $\mathcal{L}_{exc}$ (without coproducts) as well as the logic $\mathcal{L}_{exc-pl}$ have been proved to be Hilbert-Post complete, relatively to their pure sub-logics.

We have separately implemented the logics $\mathcal{L}_{st}$ and $\mathcal{L}_{exc}$ in Coq and used it to certify some properties of programs involving the state and the exceptions effects. We have also certified the relative Hilbert-Post completeness proofs of the logics $\mathcal{L}_{st}$ and $\mathcal{L}_{exc}$. Similarly, the logic $\mathcal{L}_{exc-pl}$ has also been implemented in Coq to certify the correctness of its translation into the logic $\mathcal{L}_{exc}$ and its relative Hilbert-Post completeness proof.

7.2 Future directions

Below, we itemize some exciting directions to which this thesis can be extended:

- The relation between Hoare Logic and the logic $\mathcal{L}_{st}$ could be shown.
- We plan to study the combination of the logics $\mathcal{L}_{st}$ and $\mathcal{L}_{exc}$. This may be used to build some sort of equational semantics for a toy imperative language with exceptions. This would enable us to make some equational reasoning between programs involving both the state and the exceptions effects. In fact, we have made the first attempt for combining the logics $\mathcal{L}_{st}$ and $\mathcal{L}_{exc}$ in a very restricted setting where the toy language is chosen as `IMP` (or "`while`") [Mar12] with only one type of exceptions. Following is the link to its implementation in Coq where a few examples of program equivalences involving the mentioned effects could be found: `https://forge.imag.fr/frs/download.php/697/IMPEX-STATES-EXCEPTIONS-THESIS.tar.gz`
- Other sorts of computational effects such as local state and non-termination can first be formalized separately in a decorated logic. Then, their composition with the logics $\mathcal{L}_{st}$ and $\mathcal{L}_{exc}$ could be discussed. This would enhance the decorated treatment of the effects of a given computation.

- The logic $\mathcal{L}_{st}$ with products and similarly the logic $\mathcal{L}_{exc}$ with coproducts could be proven to be Hilbert-Post complete, relative to their sub-logics. To do so, it will be necessary to figure out the canonical forms of terms with decorations (1) and (2) in the presence of pairs and copairs.
- The effect composition could be generalized in a way to compose all effects that can be separately formalized through the decorated logic. This study requires a detailed work in categorical interpretations of decorated logics.
- We have already stated our first understanding on the relation between *effect handlers* and *terms with decoration* (2) in Example 2.1.9. We could study this relation further to get a better understanding.
- We will check whether the theories $\mathcal{T}_{exc}$ and $\mathcal{T}_{st}$ preserve the Hilbert-Post completeness property when the categorical (co)products are considered ($\mathcal{T}_{exc}$ as a theory of $\mathcal{L}_{exc}$ and $\mathcal{T}_{st}$ as a theory of $\mathcal{L}_{st}$) for several exception names and locations with respect to the logics $\mathcal{L}_{meq+\mathbb{O}}$ and $\mathcal{L}_{meq+\mathbb{1}}$.

Bibliography

[Ahr15] Benedikt Ahrens. Initiality for typed syntax and semantics. *J. Formalized Reasoning*, 8(2):1–155, 2015.

[BDS13] Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

[BK01] Nick Benton and Andrew Kennedy. Exceptional syntax. *J. Funct. Program.*, 11(4):395–410, July 2001.

[Bor94] Francis Borceux. *Handbook of Categorical Algebra*. Number 50 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994. Three volumes.

[BP14] Andrej Bauer and Matija Pretnar. An effect system for algebraic effects and handlers. *Logical Methods in Computer Science*, 10(4), 2014.

[BP15] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, 84(1):108–123, 2015.

[Bra13] Edwin Brady. Programming and reasoning with algebraic effects and dependent types. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 133–144, New York, NY, USA, 2013. ACM.

[BVS93] Stephen Brookes and Kathryn Van Stone. Monads and Comonads in Intensional Semantics. Technical Report CMU-CS-93-140, Pittsburgh, PA, USA, 1993.

[CH88] Thierry Coquand and Gerard Huet. The calculus of constructions. *Inf. Comput.*, 76(2-3):95–120, February 1988.

[Coe04] Claudio Sacerdoti Coen. A semi-reflexive tactic for (sub-)equational reasoning. In *Types for Proofs and Programs, International Workshop, TYPES 2004, Jouy-en-Josas, France, December 15-18, 2004, Revised Selected Papers*, pages 98–114, 2004.

[DD10] César Domínguez and Dominique Duval. Diagrammatic logic applied to a parameterisation process. *Mathematical Structures in Computer Science*, 20(4):639–654, 2010.

[DDE+15] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, Damien Pous, and Jean-Claude Reynaud. Relative hilbert-post completeness for exceptions. In Siegfried Rump and Chee Yap, editors, *MACIS 2015, Sixth International Conference on Mathematical Aspects of Computer and Information Sciences*, 2015.

[DDEP14] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, and Damien Pous. Formal verification in Coq of program properties involving the global state effect. In Christine Tasson, editor, *25e Journées Francophones des Langages Applicatifs, Fréjus*, January 2014.

[DDER14] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, and Jean-Claude Reynaud. Certified proofs in programs involving exceptions. In Matthew England, James Davenport, Paul Libbrecht, Andrea Kohlhase, Michael Kohlhase, Walther Neuper, Pedro Quaresma, Josef Urban, Alan Sexton, Petr Sojka, and Stephen Watt, editors, *CICM'2014, Proceedings of the 2014 Conference on Intelligent Computer Mathematics, Coimbra, Portugal*, CEUR Workshop Proceedings, pages 1–16, July 2014.

[DDFR12a] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, and Jean-Claude Reynaud. Decorated proofs for computational effects: States. In *Proceedings Seventh ACCAT Workshop on Applied and Computational Category Theory, ACCAT 2012, Tallinn, Estonia, 1 April 2012.*, pages 45–59, 2012.

[DDFR12b] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, and Jean-Claude Reynaud. A duality between exceptions and states. *Mathematical Structures in Computer Science*, 22(4):719–722, 2012.

[DDR11] Jean-Guillaume Dumas, Dominique Duval, and Jean-Claude Reynaud. Cartesian effect categories are Freyd-categories. *Journal of Symbolic Computation*, 46(3):272–293, March 2011.

[DDR13] Jean-Guillaume Dumas, Dominique Duval, and Jean-Claude Reynaud. A decorated proof system for exceptions. *CoRR*, abs/1310.2338, 2013.

[DDR14] Jean-Guillaume Dumas, Dominique Duval, and Jean-Claude Reynaud. Breaking a monad-comonad symmetry between computational effects. *CoRR*, abs/1402.1051, 2014.

[Del00] David Delahaye. A tactic language for the system coq. In *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, pages 85–95, 2000.

[Dra12] C++ Working Draft. ISO/IEC JTC1/SC22/WG21 standard 14882:2011, 2012.

[Eki15] Burak Ekici. Imp with exceptions over decorated logic. In *Preproceedings of Trends in Functional Programming 2015*, 2015.

[EMS14] Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. The enriched effect calculus: syntax and semantics. *J. Log. Comput.*, 24(3):615–654, 2014.

[Fil96] A. Filinski. *Controlling Effects*. PhD thesis, 1996.

[GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.

[GSR14] Sergey Goncharov, Lutz Schröder, and Christoph Rauch. (co-)algebraic foundations for effect handling and iteration. *CoRR*, abs/1405.0854, 2014.

[HLPP07] Martin Hyland, Paul Blain Levy, Gordon D. Plotkin, and John Power. Combining algebraic effects with continuations. *Theor. Comput. Sci.*, 375(1-3):20–40, 2007.

[HP07] Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electr. Notes Theor. Comput. Sci.*, 172:437–458, 2007.

[HPP06] Martin Hyland, Gordon D. Plotkin, and John Power. Combining effects: Sum and tensor. *Theor. Comput. Sci.*, 357(1-3):70–99, 2006.

[Jac01] Bart Jacobs. A formalisation of java's exception mechanism. In *ESOP'01*, pages 284–301, 2001.

[Jas09] Mauro Jaskelioff. Modular monad transformers. In *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 64–79, 2009.

[JR11] B. Jacobs and J. Rutten. An introduction to (co)algebras and (co)induction. In *In: D. Sangiorgi and J. Rutten (eds), Advanced topics in bisimulation and coinduction*, pages 38–99, 2011.

[Kan58] Daniel Kan. Adjoint functors. *Transactions of the American Mathematical Society*, 87:294–329, 1958.

[Law63] F. W. Lawvere. *Functorial Semantic of Algebraic Theories (Available with commentary as TAC Reprint 5.)*. PhD thesis, 1963.

[Lev99] Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, pages 228–242, 1999.

[Lev06] Paul Blain Levy. Monads and adjunctions for global exceptions. *Electr. Notes Theor. Comput. Sci.*, 158:261–287, 2006.

[LG88] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, pages 47–57, New York, NY, USA, 1988. ACM.

[Lin66] F. E. J. Linton. Some aspects of equational theories. In *Proc. Conf. on Categorical Algebra*, pages 84–95, La Jolla, 1966. Springer-Verlag.

[Lin69] F. Linton. Relative functorial semantics: adjointness results. In *Lecture notes in mathematics*, volume 99, 1969.

[Mar12] Claude Marché. Mpri course 2-36-1: Proof of program. Technical report, 2012.

[Mel10] Paul-André Melliès. Segal condition meets computational effects. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 150–159, 2010.

[ML71] Saunders Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.

[Mog89] E. Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, Piscataway, NJ, USA, 1989. IEEE Press.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, July 1991.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[OBM10] Dominic A. Orchard, Max Bolingbroke, and Alan Mycroft. Ypnos: Declarative, parallel structured grid programming. In *Proceedings of the 5th ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming*, DAMP '10, pages 15–24, New York, NY, USA, 2010. ACM.

[Orc12] Dominic Orchard. Should i use a monad or a comonad? Technical report, 2012.

[POM] Tomas Petricek, Dominic Orchard, and Alan Mycroft. Coeffects: unified static analysis of context-dependence. In *Proceedings of International Conference on Automata, Languages, and Programming - Volume Part II*, ICALP 2013.

[PP01] Gordon D. Plotkin and John Power. Semantics for algebraic operations. *Electr. Notes Theor. Comput. Sci.*, 45:332–345, 2001.

[PP02] Gordon D. Plotkin and John Power. Notions of computation determine monads. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, pages 342–356, 2002.

[PP03] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[PP09] Gordon D. Plotkin and Matija Pretnar. Handlers of algebraic effects. In *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 80–94, 2009.

[PP13] Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013.

[Pre10] Matija Pretnar. *The Logic and Handling of Algebraic Effects*. PhD thesis, 2010.

[Pre14] Matija Pretnar. Inferring algebraic effects. *Logical Methods in Computer Science*, 10(3), 2014.

[SM04] Lutz Schröder and Till Mossakowski. Generic exception handling and the Java monad. In Charles Rattray, Savitri Maharaj, and Carron Shankland, editors, *Algebraic Methodology and Software Technology*, volume 3116 of *Lecture Notes in Computer Science*, pages 443–459. Springer, 2004.

[Smi10] Peter Smith. The galois connection between syntax and semantics. Technical report, Cambridge University, 2010.

[Soz10] Matthieu Sozeau. A new look at generalized rewriting in type theory. *Journal of Formalized Reasoning*, 2(1):41–62, 2010.

[Sta10] Sam Staton. Completeness for algebraic theories of local state. In *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, pages 48–63, 2010.

[Tze08] Nikos Tzevelekos. *Nominal game semantics*. PhD thesis, 2008. CS-RR-09-18.

[UV05] Tarmo Uustalu and Varmo Vene. The essence of dataflow programming. In *Central European Functional Programming School, First Summer School, CEFP 2005, Budapest, Hungary, July 4-15, 2005, Revised Selected Lectures*, pages 135–167, 2005.

[UV08] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. *Electron. Notes Theor. Comput. Sci.*, 203(5):263–284, June 2008.

[Wad92] Philip Wadler. The essence of functional programming. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, pages 1–14, New York, NY, USA, 1992. ACM.

A

Appendix 1

Decorated proofs of Plotkin&Power properties of the global state (continued from Section 5.3).

Note that in the following, we respectively use $\mathtt{lkp}_i^{(1)}$ and $\mathtt{upd}_i^{(2)}$ instead of $\mathtt{lookup}_i^{(1)}$ and $\mathtt{update}_i^{(2)}$, for the sake of simplicity.

Lemma A.0.1. Interaction update-update (IUU). *Storing a value x and then a value y at the same location i is just like storing the value y in the location.*

$$\forall\, i \in Loc,\ \ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \equiv \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} : V_i \times V_i \to \mathbb{1} \tag{A.1}$$

Proof. (1) the location i stores the same value after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\pi_1^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim id_{V_2}^{(0)} \circ \pi_2^{(0)}}\,\text{(w-lprod-eq)}}{\pi_1^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim \pi_2^{(0)}}\,\text{(idt)} \qquad \dfrac{}{\pi_2^{(0)} \circ \langle \pi_2, \pi_1 \rangle^{(0)} \equiv \pi_1^{(0)}}\,\text{(s-lpair-eq)}}{\pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim \pi_2^{(0)}}}{\pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim \pi_2^{(0)}}}{id_{V_i}^{(0)} \circ \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim id_{V_i}^{(0)} \circ \pi_2^{(0)}}\,\text{(pwrepl)}}{\mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim \mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)}}\,\text{(ax}_1\text{)}$$

(2) every location k, such that $k \neq i$, stores the same value after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall\ \mathrm{i}\ \ \mathrm{k} \in \mathrm{Loc\ s.t.\ i} \neq \mathrm{k}}{\mathtt{lkp}_k^{(1)} \circ \mathtt{upd}_i^{(2)} \sim \mathtt{lkp}_k^{(1)} \circ \langle\,\rangle_{V_i}^{(0)}}\,\text{(ax}_2\text{)}}{\mathtt{lkp}_k^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} \sim \mathtt{lkp}_k^{(1)} \circ \langle\,\rangle_{V_i}^{(0)} \circ \pi_2^{(0)}}\,\text{(wsubs)} \qquad \dfrac{\vdots}{\langle\,\rangle_{V_i}^{(0)} \circ \pi_2^{(0)} \equiv \pi_1^{(0)}}\,\text{(s-unit)}}{\mathtt{lkp}_k^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} \sim \mathtt{lkp}_k^{(1)} \circ \pi_1^{(0)}}}{\mathtt{lkp}_k^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim \mathtt{lkp}_k^{(1)} \circ \pi_1^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)})}\,\text{(wsubs)} \qquad [\Pi_1]}{\mathtt{lkp}_k^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \sim \mathtt{lkp}_k^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)}}$$

$$\dfrac{[\Pi_1]}{\pi_1^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \equiv \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)}}\,\text{(s-rprod-eq)}$$

Given above items (1), (2) and the (local-global) rule, we conclude that $\mathtt{upd}_i^{(2)} \circ \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_i}^{(0)}) \equiv \mathtt{upd}_i^{(2)} \circ \pi_2^{(0)}$.

□

Lemma A.0.2. Interaction update-lookup (IUL). *When one stores a value a in a location i and then reads the location i, one gets the value a.*

$$\forall\, i \in Loc,\ \mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \sim id_{V_i}^{(0)} : V_i \to V_i \tag{A.2}$$

Proof. Applying (ax_1) closes the goal.

□

Lemma A.0.3. Commutation lookup-lookup (CLL). *The order of reading two different locations i and j does not matter.*

$$\begin{aligned} \forall\, i \neq j \in Loc,\ (id_{V_i}^{(0)} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_i^{(1)} \equiv \\ permut_{j,i}^{(0)} \circ (id_{V_j}^{(0)} \times_l \mathtt{lkp}_i^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_j^{(1)} : \mathbb{1} \to V_i \times V_j \end{aligned} \tag{A.3}$$

Proof. (1) result agreement on the first argument of the returned pair after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{id_{\mathbb{1}}^{(0)} \equiv \langle\,\rangle_{V_j}^{(0)} \circ \mathtt{lkp}_j^{(1)}}\,(\text{s-unit})}{\mathtt{lkp}_i^{(1)} \equiv \mathtt{lkp}_i^{(1)} \circ \langle\,\rangle_{V_j}^{(0)} \circ \mathtt{lkp}_j^{(1)}}\,(\text{replsubs})}{\mathtt{lkp}_i^{(1)} \sim \mathtt{lkp}_i^{(1)} \circ \langle\,\rangle_{V_j}^{(0)} \circ \mathtt{lkp}_j^{(1)}}\,(\text{stow}) \qquad \dfrac{}{\pi_2^{(0)} \circ \underbrace{\langle id_{V_j}, \langle\,\rangle_{V_j}\rangle^{(0)}}_{\pi_1^{-1(0)}} \equiv \langle\,\rangle_{V_j}^{(0)}}\,(\text{s-lpair-eq})}{\mathtt{lkp}_i^{(1)} \sim \mathtt{lkp}_i^{(1)} \circ \pi_2^{(0)} \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_j^{(1)}}}{\mathtt{lkp}_i^{(1)} \sim \pi_2^{(0)} \circ (id_{V_j}^{(0)} \times_l \mathtt{lkp}_i^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_j^{(1)}}\,[\Pi_1]}{\mathtt{lkp}_i^{(1)} \sim \pi_1^{(0)} \circ \underbrace{\langle \pi_2, \pi_1 \rangle^{(0)}}_{permut_{j,i}} \circ (id_{V_j}^{(0)} \times_l \mathtt{lkp}_i^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_j^{(1)}}\,[\Pi_2]}{id_{V_i}^{(0)} \circ \mathtt{lkp}_i^{(1)} \sim \pi_1^{(0)} \circ permut_{j,i} \circ (id_{V_j}^{(0)} \times_l \mathtt{lkp}_i^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_j^{(1)}}\,(\text{idt})}{\pi_1^{(0)} \circ (id_{V_i}^{(0)} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_i^{(1)} \sim \pi_1^{(0)} \circ permut_{j,i} \circ (id_{V_j}^{(0)} \times_l \mathtt{lkp}_i^{(1)}) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}_j^{(1)}}\,[\Pi_3]$$

$$\dfrac{[\Pi_1]}{\pi_2^{(0)} \circ (id_{V_j}^{(0)} \times_l \mathtt{lkp}_i^{(1)}) \equiv \mathtt{lkp}_i^{(1)} \circ \pi_2^{(0)}}\,(\text{s-lprod-eq}) \qquad \dfrac{\dfrac{[\Pi_2]}{\pi_1^{(0)} \circ \langle \pi_2, \pi_1 \rangle^{(0)} \sim \pi_2^{(0)}}\,(\text{l-lpair-eq})}{\pi_1^{(0)} \circ \langle \pi_2, \pi_1 \rangle^{(0)} \equiv \pi_2^{(0)}}\,(\text{wtos})$$

$$\dfrac{\dfrac{\dfrac{[\Pi_3]}{\pi_1^{(0)} \circ (id_{V_i}^{(0)} \times_l \mathtt{lkp}_j^{(1)}) \sim id_{V_i}^{(0)} \circ \pi_1^{(0)}}\,(\text{w-lprod-eq})}{\pi_1^{(0)} \circ (id_{V_i}^{(0)} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)} \sim id_{V_i}^{(0)} \circ \pi_1^{(0)} \circ \pi_1^{-1(0)}}\,(\text{wsubs})}{\pi_1^{(0)} \circ (id_{V_i}^{(0)} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)} \sim id_{V_i}^{(0)}}\,(\text{wtos})$$

(2) result and effect agreement on the second argument of the returned pair after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{id^{(0)}_{V_j} \circ \pi^{(0)}_1 \sim \pi^{(0)}_1 \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i)}\,(\text{w-lprod-eq})}{\pi^{(0)}_1 \circ \pi_1^{-1(0)} \sim \pi^{(0)}_1 \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i) \circ \pi_1^{-1(0)}}\,(\text{wsubs})}{id^{(0)}_{V_j} \sim \pi^{(0)}_1 \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i) \circ \pi_1^{-1(0)}}\,(\text{w-lpair-eq}) \qquad \dfrac{}{\pi^{(0)}_1 \equiv \pi^{(0)}_2 \circ permut_{j,i}}\,(\text{s-pair-eq})}{\dfrac{id^{(0)}_{V_j} \sim \pi^{(0)}_2 \circ permut_{j,i} \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_j}{\mathtt{lkp}^{(1)}_j \sim \pi^{(0)}_2 \circ permut_{j,i} \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_j}\,(\text{wsubs})}{\mathtt{lkp}^{(1)}_j \equiv \pi^{(0)}_2 \circ permut_{j,i} \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_j}\,(\text{wtos}) \qquad [\Pi_1]}{\pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_i \equiv \pi^{(0)}_2 \circ permut^{(0)}_{j,i} \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_j}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[\Pi_1]}{\pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \equiv \mathtt{lkp}^{(1)}_j \circ \pi^{(0)}_2}\,(\text{s-lprod-eq})}{\pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)} \equiv \mathtt{lkp}^{(1)}_j \circ \pi^{(0)}_2 \circ \pi_1^{-1(0)}}\,(\text{replsubs}) \qquad \dfrac{}{\pi^{(0)}_2 \circ \pi_1^{-1(0)} \equiv \langle\,\rangle^{(0)}_{V_i}}\,(\text{s-pair-eq})}{\dfrac{\pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)} \equiv \mathtt{lkp}^{(1)}_j \circ \langle\,\rangle^{(0)}_{V_i}}{\pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_i \equiv \mathtt{lkp}^{(1)}_j \circ \langle\,\rangle^{(0)}_{V_i} \circ \mathtt{lkp}^{(1)}_i}\,(\text{replsubs}) \qquad [\Pi_2]}{\pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_i \equiv \mathtt{lkp}^{(1)}_j}$$

$$\dfrac{[\Pi_2]}{\langle\,\rangle^{(0)}_{V_i} \circ \mathtt{lkp}^{(1)}_i \equiv id^{(0)}_{\mathbb{1}}}\,(\text{s-unit})$$

Given above items (1), (2) and (lpair-u), we conclude that $(id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_i \equiv permut^{(0)}_{j,i} \circ (id^{(0)}_{V_j} \times_l \mathtt{lkp}^{(1)}_i) \circ \pi_1^{-1(0)} \circ \mathtt{lkp}^{(1)}_j$.

□

Lemma A.0.4. Commutation update-update (CUU). *The order of storing in two different locations i and j does not matter.*

$$\begin{aligned} &\forall\, i \neq j \in Loc,\ \mathtt{upd}^{(2)}_j \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \equiv \\ &\qquad \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j) : V_i \times V_j \to \mathbb{1} \end{aligned} \tag{A.4}$$

Proof. (1) the location i stores the same value after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{id^{(0)}_{V_i} \circ \pi^{(0)}_1 \sim \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{w-lprod-eq})}{id^{(0)}_{V_i} \circ \pi^{(0)}_1 \sim id^{(0)}_{V_i} \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{idt})}{\mathtt{lkp}^{(1)}_i \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \sim \mathtt{lkp}^{(1)}_i \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{ax}_1)}{\mathtt{lkp}^{(1)}_i \circ \pi^{(0)}_1 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \sim \mathtt{lkp}^{(1)}_i \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{s-rprod-eq})}{\mathtt{lkp}^{(1)}_i \circ \langle\,\rangle_{V_j} \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \sim \mathtt{lkp}^{(1)}_i \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{s-unit})}{\mathtt{lkp}^{(1)}_i \circ \mathtt{upd}^{(2)}_j \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \sim \mathtt{lkp}^{(1)}_i \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{ax}_2)$$

(2) the location j stores the same value after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall\, \mathrm{i} \in \mathrm{Loc}}{id^{(0)}_{V_j} \sim \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_j}\,(\text{ax}_1)}{id^{(0)}_{V_j} \circ \pi^{(0)}_2 \sim \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_j \circ \pi^{(0)}_2}\,(\text{wsubs})}{id^{(0)}_{V_j} \circ \pi^{(0)}_2 \sim \mathtt{lkp}^{(1)}_j \circ \pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{s-lpair-eq})}{id^{(0)}_{V_j} \circ \pi^{(0)}_2 \sim \mathtt{lkp}^{(1)}_j \circ \langle\,\rangle_{V_i} \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{s-unit})}{id^{(0)}_{V_j} \circ \pi^{(0)}_2 \sim \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{ax}_2)}{\pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \sim \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{w-rpair-eq})}{id^{(0)}_{V_j} \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \sim \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{idt})}{\mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_j \circ \pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j) \sim \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{ax}_1)$$

(3) every location k, such that $k \neq j \wedge k \neq i$, stores the same value after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\langle\,\rangle^{(0)}_{V_i} \circ \pi^{(0)}_1 \equiv \langle\,\rangle^{(0)}_{V_j} \circ \pi^{(0)}_2}\,(\text{s-unit})}{\mathtt{lkp}^{(1)}_k \circ \langle\,\rangle^{(0)}_{V_i} \circ \pi^{(0)}_1 \equiv \mathtt{lkp}^{(1)}_k \circ \langle\,\rangle^{(0)}_{V_j} \circ \pi^{(0)}_2}\,(\text{replsubs})}{\mathtt{lkp}^{(1)}_k \circ \langle\,\rangle^{(0)}_{V_i} \circ \pi^{(0)}_1 \sim \mathtt{lkp}^{(1)}_k \circ \langle\,\rangle^{(0)}_{V_j} \circ \pi^{(0)}_2}\,(\text{stow})}{\mathtt{lkp}^{(1)}_k \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \sim \mathtt{lkp}^{(1)}_k \circ \mathtt{upd}^{(2)}_j \circ \pi^{(0)}_2}\,(\text{ax}_2)}{\mathtt{lkp}^{(1)}_k \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \sim \mathtt{lkp}^{(1)}_k \circ \pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{s-lpair-eq})}{\mathtt{lkp}^{(1)}_k \circ \pi^{(0)}_1 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \sim \mathtt{lkp}^{(1)}_k \circ \langle\,\rangle^{(0)}_{V_i} \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{s-rpair-eq})}{\mathtt{lkp}^{(1)}_k \circ \langle\,\rangle^{(0)}_{V_j} \circ \pi^{(0)}_2 \circ \langle (id_{V_j} \circ \pi_2), (\mathtt{upd}_i \circ \pi_1)\rangle^{(2)} \sim \mathtt{lkp}^{(1)}_k \circ \langle\,\rangle^{(0)}_{V_i} \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{s-unit})}{\mathtt{lkp}^{(1)}_k \circ \mathtt{upd}^{(2)}_j \circ \pi^{(0)}_2 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j) \sim \mathtt{lkp}^{(1)}_k \circ \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)}\,(\text{ax}_2)$$

Given above items (1), (2), (3) and the (local-global) rule, we conclude with $\mathtt{upd}^{(2)}_j \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \equiv \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{upd}^{(2)}_j)$. □

Lemma A.0.5. Commutation update-lookup (CUL). *The order of storing in a location i and reading another location j does not matter.*

$$\begin{aligned} &\forall\, i \neq j \in Loc,\ \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_i \equiv \\ &\pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)} : V_i \to V_j \end{aligned} \tag{A.5}$$

Proof. (1) effect agreement after executing operations on both sides:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\pi^{(0)}_1 \sim \pi^{(0)}_1 \circ \underbrace{\langle id_{V_i}, \langle\,\rangle_{V_i}\rangle^{(0)}}_{\pi_1^{-1(0)}}}\,(\text{w-lpair-eq})}{\pi^{(0)}_1 \equiv \pi^{(0)}_1 \circ \pi_1^{-1(0)}}\,(\text{wtos})}{\mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \equiv \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ \pi_1^{-1(0)}}\,(\text{replsubs}) \qquad \dfrac{\dfrac{\dfrac{}{\pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \sim id^{(0)}_{V_i} \circ \pi^{(0)}_1}\,(\text{w-lpair-eq})}{\pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \equiv id^{(0)}_{V_i} \circ \pi^{(0)}_1}\,(\text{wtos})}{\pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \equiv \pi^{(0)}_1}\,(\text{idt})}{\mathtt{upd}^{(2)}_i \equiv \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{upd}^{(2)}_i \equiv \mathtt{upd}^{(2)}_i \circ \pi^{(0)}_1 \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)}}{\mathtt{upd}^{(2)}_i \equiv \pi^{(0)}_1 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)}}\,(\text{s-rpair-eq})}{\mathtt{upd}^{(2)}_i \equiv \langle\,\rangle^{(0)}_{V_j} \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)}}\,(\text{s-unit})}{id^{(0)}_{\mathbb{1}} \circ \mathtt{upd}^{(2)}_i \equiv \langle\,\rangle^{(0)}_{V_j} \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)}}\,(\text{idt})}{\langle\,\rangle^{(0)}_{V_j} \circ \mathtt{lkp}^{(1)}_j \circ \mathtt{upd}^{(2)}_i \equiv \langle\,\rangle_{V_j} \circ \pi^{(0)}_2 \circ (\mathtt{upd}^{(2)}_i \times_r id^{(0)}_{V_j}) \circ (id^{(0)}_{V_i} \times_l \mathtt{lkp}^{(1)}_j) \circ \pi_1^{-1(0)}}\,[\Pi_1]$$

$$\text{(s-unit)}\ \frac{[\Pi_1]}{\langle\,\rangle_{V_j}^{(0)} \circ \mathtt{lkp}_j^{(1)} \equiv id_{\mathbb{1}}^{(0)}}$$

(2) result agreement after executing operations on both sides:

$$\text{(ax}_2\text{)}\ \frac{\forall\ \mathrm{i, j} \in \mathrm{Loc}}{\mathtt{lkp}_j^{(1)} \circ \mathtt{upd}_i^{(2)} \sim \mathtt{lkp}_j^{(1)} \circ \langle\,\rangle_{V_i}}$$
$$\text{(s-lpair-eq)}\ \frac{\mathtt{lkp}_j^{(1)} \circ \mathtt{upd}_i^{(2)} \sim \mathtt{lkp}_j^{(1)} \circ \pi_2^{(0)} \circ \pi_1^{-1(0)}}{\mathtt{lkp}_j^{(1)} \circ \mathtt{upd}_i^{(2)} \sim \mathtt{lkp}_j^{(1)} \circ \pi_2^{(0)} \circ (id_{V_i} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)}}$$
$$\text{(idt)}\ \frac{\mathtt{lkp}_j^{(1)} \circ \mathtt{upd}_i^{(2)} \sim id_{V_j}^{(0)} \circ \pi_2^{(0)} \circ (id_{V_i} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)}}{\mathtt{lkp}_j^{(1)} \circ \mathtt{upd}_i^{(2)} \sim \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_j}^{(0)}) \circ (id_{V_i} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)}}\ \text{(w-rpair-eq)}$$

Given above items (1), (2) and the (effect) rule, we conclude that $\mathtt{lkp}_j^{(1)} \circ \mathtt{upd}_i^{(2)} \equiv \pi_2^{(0)} \circ (\mathtt{upd}_i^{(2)} \times_r id_{V_j}^{(0)}) \circ (id_{V_i}^{(0)} \times_l \mathtt{lkp}_j^{(1)}) \circ \pi_1^{-1(0)}$.

□

Lemma A.0.6. Commutation lookup-constant (CLC). *Just after storing a constant c in a location i, observing the content of i is the same as regenerating the constant c.*

$$\begin{aligned} \forall\, i \in Loc,\ \forall\, c \in V_i;\ \mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \equiv \\ \mathtt{const}\, c^{(0)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} : \mathbb{1} \to V_i \end{aligned} \tag{A.6}$$

Proof. (1) effect agreement after executing operations on both sides:

$$\text{(replsubs)}\ \frac{\text{(s-unit)}\ \dfrac{\vdots}{\langle\,\rangle_{V_i}^{(0)} \circ \mathtt{lkp}_i^{(1)} \equiv id_{\mathbb{1}}^{(0)}}}{\langle\,\rangle_{V_i}^{(0)} \circ \mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \equiv id_{\mathbb{1}}^{(0)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)}} \qquad \text{(s-unit)}\ \frac{\vdots}{\langle\,\rangle_{V_i}^{(0)} \circ \mathtt{const}\, c^{(0)} \equiv id_{\mathbb{1}}^{(0)}}$$
$$\langle\,\rangle_{V_i}^{(0)} \circ \mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \equiv \langle\,\rangle_{V_i}^{(0)} \circ \mathtt{const}\, c^{(0)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)}$$

(2) result agreement after executing operations on both sides:

$$\text{(ax}_1\text{)}\ \frac{\forall\ \mathrm{i} \in \mathrm{Loc}}{\mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \sim id_{V_i}^{(0)}}$$
$$\text{(wsubs)}\ \frac{\mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \sim \mathtt{const}\, c^{(0)}}{\mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \sim \mathtt{const}\, c^{(0)} \circ id_{\mathbb{1}}^{(0)}}\ \text{(ids)} \qquad \text{(w-unit)}\ \frac{}{\mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \sim id_{\mathbb{1}}^{(0)}}$$
$$\mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \sim \mathtt{const}\, c^{(0)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)}$$

Given above items (1), (2) and the (effect) rule, we conclude that $\mathtt{lkp}_i^{(1)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)} \equiv \mathtt{const}\, c^{(0)} \circ \mathtt{upd}_i^{(2)} \circ \mathtt{const}\, c^{(0)}$.

□

B

Appendix 2

Full proof of Corollary 5.4.8:

1. We have four cases to consider:

   (1.1) When both f_1 and f_2 are pure, we set $a_1 = f_2^{(0)}$ and $a_2 = f_2^{(0)}$ so as to trivially obtain $f_1^{(0)} \equiv f_2^{(0)} \iff f_1^{(0)} \equiv f_2^{(0)}$.

   (1.2) When f_1 is an accessor while $f_2^{(0)}$ is pure, we only get $f_1^{(1)} \equiv v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)}$ for some pure term $v_1^{(0)} : V \to Y$. In this case, we set $a_1 = v_1^{(0)}$, $a_2 = f_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_V^{(0)}$, $b_1 = f_2^{(0)}$ and $b_2 = f_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_X^{(0)}$ and get $f_1^{(1)} \equiv f_2^{(0)} \iff v_1^{(0)} \equiv f_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_V^{(0)}$ and $f_2^{(0)} \equiv f_2^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_X^{(0)}$ from Point 4 in Proposition 5.4.5.

   (1.3) Symmetrically when f_2 is an accessor while f_1 is pure, we only get $f_2^{(1)} \equiv v_2^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)}$ for some pure term $v_2^{(0)} : V \to Y$. In this case, we set $a_1 = v_2^{(0)}$, $a_2 = f_1^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_V^{(0)}$, $b_1 = f_1^{(0)}$ and $b_2 = f_1^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_X^{(0)}$ and get $f_2^{(1)} \equiv f_1^{(0)} \iff v_2^{(0)} \equiv f_1^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_V^{(0)}$ and $f_1^{(0)} \equiv f_1^{(0)} \circ k_X^{(0)} \circ \langle\,\rangle_X^{(0)}$ also from Point 4 in Proposition 5.4.5 to close the goal.

   (1.4) When both f_1 and f_2 are accessors, thanks to Corollary 5.4.4, we have $f_1^{(1)} \equiv v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)}$ and $f_2(1) \equiv v_2^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)}$ for some pure terms $v_1^{(0)}, v_2^{(0)} : V \to Y$. Setting $a_1 = v_1$ and $a_2 = v_2$, we obtain $v_1^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \equiv v_2^{(0)} \circ \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \iff v_1^{(0)} \equiv v_2^{(0)}$ from Point 3 in Proposition 5.4.5.

2. We again have four cases to show:

   (2.1) In the case where both are accessors, we set $a_1 = f_1^{(1)}$ and $a_2 = f_2^{(1)}$ so as to trivially obtain $f_1^{(1)} \equiv f_2^{(1)} \iff f_1^{(1)} \equiv f_2^{(1)}$ which is trivial.

   (2.2) When f_1 is a modifier whilst f_2 is an accessor, we only get $f_1^{(2)} \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)}$. We set $a_1 = a_1^{(1)}$, $a_2 = \texttt{lookup} \circ \langle\,\rangle_X^{(0)}$, $b_1 = f_2^{(1)}$ and $b_2 = u_1^{(0)} \circ a_1^{(1)}$, then we get $f_1^{(2)} \equiv f_2^{(1)} \iff \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \equiv a_1^{(1)}$ and $f_2^{(1)} \equiv u_1^{(0)} \circ a_1^{(1)}$. We apply Point 2 in Proposition 5.4.5.

   (2.3) Symmetrically when f_2 is a modifier whilst f_1 is an accessor, we only get $f_2^{(2)} \equiv u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$. We set $a_1 = \texttt{lookup} \circ \langle\,\rangle_X^{(0)}$, $a_2 = a_2^{(1)}$, $b_1 = f_1^{(1)}$ and $b_2 = u_2^{(0)} \circ a_2^{(1)}$, then we get $f_2^{(2)} \equiv f_1^{(1)} \iff \texttt{lookup} \circ \langle\,\rangle_X^{(0)} \equiv a_2^{(1)}$ and $f_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)}$. Similarly, we apply Point 2 in Proposition 5.4.5.

   (2.4) When both f_1 and f_2 are modifiers, due to Point 2 in Proposition 5.4.3, we get $f_1^{(2)} \equiv u_1^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_1^{(1)}$ and $f_2^{(2)} \equiv u_2^{(0)} \circ \texttt{lookup} \circ \texttt{update} \circ a_2^{(1)}$ for some pure terms $u_1^{(0)}, u_2^{(0)} : V \to Y$ and accessors $a_1^{(1)}, a_2^{(1)} : X \to V$. By

setting $a_1 = a_1^{(1)}$, $a_2 = a_2^{(1)}$, $b_1 = u_1^{(0)} \circ a_1^{(1)}$ and $b_2 = u_2^{(0)} \circ a_2^{(1)}$, we obtain $f_1^{(2)} \equiv f_2^{(2)} \iff a_1^{(1)} \equiv a_2^{(1)}$ and $u_1^{(0)} \circ a_1^{(1)} \equiv u_2^{(0)} \circ a_2^{(1)}$. Now, we apply Point 1 in Proposition 5.4.5 to close the goal. □

Full proof of Corollary 6.9.8:

1. The proof proceeds on the distinction whether X is inhabited or not:

   (1.1) when $X \neq \mathbb{O}$, we have four cases to consider:

   (1.1.1) In the case where both a_1 and a_2, we set $v_1 = a_1$ and $v_2 = a_2$ so as to trivially obtain $a_1^{(0)} \equiv a_2^{(0)} \iff a_1^{(0)} \equiv a_2^{(0)}$.

   (1.1.2) When a_1 is a propagator while a_2 is pure, we can only get $a_1^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)}$ for some pure term $v_1^{(0)} : X \to EV$. We set $v = a_2^{(0)}$ so that the goal looks like $[\,]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)} \equiv a_2^{(0)} \iff \forall f^{(0)}, g^{(0)} : X \to Y \; s.t. \; f^{(0)} \equiv g^{(0)}$. This is solved by applying Assumption 6.9.5.

   (1.1.3) Symmetrically when a_2 is a propagator while a_1 is pure, we can only get $a_2^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag} \circ v_2^{(0)}$ for some pure term $v_2^{(0)} : X \to EV$. We set $v = a_1^{(0)}$ so that the goal looks like $a_1^{(0)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag} \circ v_2^{(0)} \iff \forall f^{(0)}, g^{(0)} : X \to Y \; s.t. \; f^{(0)} \equiv g^{(0)}$. This is solved by applying Assumption 6.9.5.

   (1.1.4) When both a_1 and a_2 are propagators, then thanks to Corollary 6.9.4, we have $a_1^{(1)} \equiv [\,]_Y \circ \mathtt{tag} \circ v_1^{(0)}$ and similarly $a_2^{(1)} \equiv [\,]_Y \circ \mathtt{tag} \circ v_2^{(0)}$ for some pure terms $v_1^{(0)}, v_2^{(0)} : X \to EV$. We set $v_1 = v_1^{(0)}$ and $v_2 = v_2^{(0)}$. So that the goal looks like $[\,]_Y^{(0)} \circ \mathtt{tag} \circ v_1^{(0)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag} \circ v_2^{(0)} \iff v_1^{(0)} \equiv v_2^{(0)}$. This is solved by the application of Point 3 in Proposition 6.9.7.

   (1.2) Thanks to (s-empty), we have $[\,]_Y \equiv a_i$ for each $i \in \{1, 2\}$. Thus $a_1 \equiv a_2$.

2. We have four cases to prove:

   (2.1) When both f_1 and f_2 are propagators, we set $a_1 = f_1^{(1)}$ and $a_2 = f_2^{(1)}$ so as to trivially obtain $f_1^{(1)} \equiv f_2^{(1)} \iff f_1^{(1)} \equiv f_2^{(1)}$.

   (2.2) When f_1 is a catcher while f_2 is a propagator, we only get $f_1^{(2)} \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)}$. Here, we set $a_1 = a_1^{(1)}$, $a_2 = [\,]_Y^{(0)} \circ \mathtt{tag}$, $b_1 = f_2^{(1)}$ and $b_2 = a_1^{(1)} \circ u_1^{(0)}$. Now, the goal looks like $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv f_2^{(1)} \iff a_1^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag}$ and $f_2^{(1)} \equiv a_1^{(1)} \circ u_1^{(0)}$. This comes from Point 2 in Proposition 6.9.7.

   (2.3) Symmetrically when f_2 is a catcher while f_1 is a propagator, we only get $f_2^{(2)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$. Here, we set $a_1 = [\,]_Y^{(0)} \circ \mathtt{tag}$, $a_2 = a_2^{(1)}$, $b_1 = f_1^{(1)}$ and $b_2 = a_2^{(1)} \circ u_2^{(0)}$. Now, the goal looks like $f_1^{(1)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)} \iff a_2^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag}$ and $f_1^{(1)} \equiv a_2^{(1)} \circ u_2^{(0)}$. This comes also from Point 2 in Proposition 6.9.7.

   (2.4) When both f_1 and f_2 are catchers, due to Point 2 in Proposition 6.9.3, we get $f_1^{(2)} \equiv a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)}$ and $f_2^{(2)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2^{(0)}$ for some pure terms $u_1^{(0)}, u_2^{(0)} : X \to EV$ and propagators $a_1^{(1)}, a_2^{(1)} : EV \to Y$. We can simply set $a_1 = a_1^{(1)}$, $a_2 = a_2^{(1)}$, $b_1 = a_1^{(1)} \circ u_1^{(0)}$ and $b_2 = a_2^{(1)} \circ u_2^{(0)}$. Therefore, we obtain a goal which looks like $a_1^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1^{(0)} \equiv a_2^{(1)} \circ \mathtt{untag} \circ$

$\texttt{tag} \circ u_2^{(0)} \iff a_1^{(1)} \equiv a_2^{(1)}$ and $a_1^{(1)} \circ u_1^{(0)} \equiv a_2^{(1)} \circ u_2^{(0)}$. We apply Point 1 in Proposition 6.9.7 to solve it. □