

Extending higher-order logic with predicate subtyping : application to PVS

Frédéric Gilbert

▶ **To cite this version:**

Frédéric Gilbert. Extending higher-order logic with predicate subtyping : application to PVS. Logic in Computer Science [cs.LO]. Université Sorbonne Paris Cité, 2018. English. NNT : 2018USPCC009 . tel-02058937

HAL Id: tel-02058937

https://theses.hal.science/tel-02058937

Submitted on 6 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Sorbonne Paris Cité

Thèse de doctorat

de l'Université Sorbonne Paris Cité

Préparée à l'Université Paris Diderot

École doctorale de Sciences Mathématiques de Paris Centre (ED386)

Deducteam (INRIA) et LSL (CEA LIST)

Extending higher-order logic with predicate subtyping

Application to PVS

Par Frédéric Gilbert

Thèse de doctorat de mathématiques
(logique et fondements de l'informatique)

Dirigée par Gilles Dowek et Florent Kirchner

Présentée et soutenue publiquement à Paris le 10 avril 2018

Présidente du jury	Delia Kesner	Professeure, Université Paris Diderot
Rapporteurs	Herman Geuvers	Professor, Radboud University Nijmegen
	Shankar Natarajan	Distinguished Scientist, SRI International
Examinateurs	César A. Muñoz	Research Computer Scientist, NASA
	Jean-Pierre Jouannaud	Professeur émérite, Université Paris-Sud
Directeurs de thèse	Gilles Dowek	Directeur de recherche, INRIA
	Florent Kirchner	Chef de laboratoire, CEA LIST

Abstract

The type system of higher-order logic allows to exclude some unexpected expressions such as the application of a predicate to itself. However, it is not sufficient to verify more complex criteria such as the absence of divisions by zero. This thesis is dedicated to the study of an extension of higher-order logic, named *predicate subtyping*, whose purpose is to make the assignment of types as expressive as the assignment of predicates. Starting from a type A and a predicate $P(x)$ of domain A, predicate subtyping allows to build a subtype of A, denoted $\{x : A \mid P(x)\}$, whose elements are the terms t of type A such that $P(t)$ is provable. Predicate subtyping is at the heart of the proof system PVS.

This work presents the formalization of a minimal system expressing predicate subtyping, named PVS-Core, as well as a system of verifiable certificates for PVS-Core. This second system, named PVS-Cert, is based on the introduction of proof terms and explicit coercions. PVS-Core and PVS-Cert are equipped with a notion of conversion corresponding respectively to equality modulo β and to equality modulo β *and the erasure of coercions*, chosen to establish a simple correspondence between the two systems.

The construction of PVS-Cert is similar to that of PTSs (Pure Type Systems) with dependent pairs and PVS-Cert can be equipped with the notion of $\beta\sigma$-reduction used at the core of these systems. One of the main theorems proved in this work is the strong normalization of both the reduction underlying the conversion and $\beta\sigma$-reduction. This theorem allows, on the one hand, to build a type-checking (and proof-checking) algorithm for PVS-Cert and, on the other hand, to prove a cut elimination result, used in turn to prove important properties of the two studied systems. Furthermore, it is also proved that PVS-Cert is a conservative extension of the PTS λ-HOL and that, as a consequence, PVS-Core is a conservative extension of higher-order logic.

A second part presents the prototype of an instrumentation of PVS to generate proof certificates. A third and final part is dedicated to the study of links between classical and constructive logic, with the definition of a minimal double-negation translation as well as the presentation of an automated proof constructivization algorithm.

Keywords: higher-order logic, predicate subtyping, PVS, type theory, proof theory

Résumé

Le système de types de la logique d'ordre supérieur permet d'exclure certaines expressions indésirables telles que l'application d'un prédicat à lui-même. Cependant, il ne suffit pas pour vérifier des critères plus complexes comme l'absence de divisions par zéro. Cette thèse est consacrée à l'étude d'une extension de la logique d'ordre supérieur appelée sous-typage par prédicats (*predicate subtyping*), dont l'objet est de rendre l'attribution de types aussi expressive que l'attribution de prédicats. A partir d'un type A et d'un prédicat $P(x)$ de domaine A, le sous-typage par prédicats permet de construire un sous-type de A, noté $\{x : A \mid P(x)\}$, dont les éléments sont les termes t de type A tels que $P(t)$ est démontrable. Le sous-typage par prédicats est au cœur du système PVS.

Ce travail présente la formalisation d'un système minimal incluant le sous-typage par prédicats, appelé PVS-Core, ainsi qu'un système de certificats vérifiables pour PVS-Core. Ce deuxième système, appelé PVS-Cert, repose sur l'introduction de termes de preuves et de coercions explicites. PVS-Core et PVS-Cert sont munis d'une notion de conversion correspondant respectivement à l'égalité modulo β et à l'égalité modulo β *et effacement des coercions*, choisi pour établir une correspondance simple entre les deux systèmes.

La construction de PVS-Cert est semblable à celle des PTS (Pure Type Systems) avec paires dépendantes et PVS-Cert peut être muni de la notion de $\beta\sigma$-réduction utilisée au coeur de ces systèmes. L'un des principaux théorèmes démontré dans ce travail est la normalisation forte de la réduction sous-jacente à la conversion et de la $\beta\sigma$-réduction. Ce théorème permet d'une part de construire un algorithme de vérification du typage (et des preuves) pour PVS-Cert et d'autre part de démontrer un résultat d'élimination des coupures, utilisé à son tour pour prouver plusieurs propriétés importantes des deux systèmes étudiés. Par ailleurs, il est également démontré que PVS-Cert est une extension conservative du PTS λ-HOL, et qu'en conséquence PVS-Core est une extension conservative de la logique d'ordre supérieur.

Une deuxième partie présente le prototype d'une instrumentation de PVS pour produire des certificats de preuve. Une troisième et dernière partie est consacrée à l'étude de liens entre logique classique et constructive avec la définition d'une traduction par double négation minimale ainsi que la présentation d'un algorithme de constructivisation automatique des preuves.

Mots-clés : logique d'ordre supérieur, sous-typage par prédicats, PVS, théorie des types, théorie de la démonstration

Acknowledgments

First of all, I would like to thank Gilles Dowek and Florent Kirchner, who gave me the chance to discover the foundations of mathematics. Having two full-time advisors was a true privilege. I'm very grateful to Gilles for helping me face the numerous choices of research orientation and formalization, for providing me hindsight and perspectives, but also for encouraging me to choose trial and error over dithering and hesitation. I also want to express my sincere thanks to Florent for his insightful advice when I was stuck either with theoretical questionings or technical difficulties, as he taught me in both cases how to decompose hard problems into sequences of easier ones.

I warmly thank César Muñoz, who introduced me to PVS and invited me to spend two summers working with him and his team. I learned a lot in each of these visits.

I'm deeply indebted to my reviewers Herman Geuvers and Shankar Natarajan for their careful reading of my dissertation. Their respective works were both great sources of inspiration to me. I'm also much obliged to Delia Kesner and Jean-Pierre Jouannaud for accepting to be part jury, and I'm very honoured by their interest in this work.

I enjoyed a lot working in Deducteam and I'm grateful to all its members for the excellent atmosphere as well as the stimulating discussions we had. Special thanks to Ali Assaf, Raphaël Cauderlier, Simon Cruanes, Pierre Halmagrand, Bruno Bernardo, Simon Martiel, and François Thiré, with whom I've been glad to spend most of my time, and to Olivier Hermant for guiding me when I took my very first steps in logic.

My time at LSL was quantitatively shorter but equally fulfilling. I would like to thank in particular Adel, Michael, and Quentin, it was a great pleasure to share my office with them.

I'll never be able to thank enough the NIA/NASA team for their welcome and their hospitality during my two visits in Hampton. I want to express my deepest gratitude to Andrew, Mariano, Azul, Bernardo, Laura, Marco, Carolyn, and Jennifer for their friendliness and for making my visits so pleasant.

I'd like to thank the institutions without which my PhD work wouldn't have seen the light of day: Inria, ENS Cachan, CEA LIST, École des Ponts, NASA, and NIA.

Preparing a PhD thesis requires less patience and perseverance from the candidates than it does from their relatives. My particular thanks go to my friends for putting up with me all these years, to my family for their encouragements and their help, and especially to Alexia for her constant support in the achievement of this project.

Contents

1 Introduction **13**
- 1.1 Types and predicates in higher-order logic . . . 14
- 1.2 Enriching types with predicate subtyping . . . 15
- 1.3 The practice of predicate subtyping in PVS . . . 16
- 1.4 Contributions . . . 17
  - 1.4.1 First part: theoretical analysis of predicate subtyping . . . 17
  - 1.4.2 Second part: extracting proof certificates from PVS . . . 19
  - 1.4.3 Third part: expressing classical first-order logic in constructive systems . . . 19
- 1.5 Related works . . . 20

I Theoretical analysis of predicate subtyping **23**

2 Predicate subtyping in PVS **25**
- 2.1 Presentation of a subset of the specification language . . . 25
  - 2.1.1 General structure of PVS specifications . . . 26
  - 2.1.2 Higher-order logic types and expressions . . . 27
  - 2.1.3 Addition of predicate subtyping . . . 29
- 2.2 Idealization of typing and provability in PVS . . . 30
  - 2.2.1 Idealization of typing and well-formedness . . . 31
  - 2.2.2 Simplification of the PVS sequent calculus with natural deduction 36

3 PVS-Core: the formalization of predicate subtyping **39**
- 3.1 Terms and judgements . . . 39
- 3.2 Rules and derivability . . . 42
- 3.3 Summary of PVS-Core's main characteristics . . . 44
  - 3.3.1 An extension of higher-order logic . . . 45
  - 3.3.2 Predicate subtyping and conversion . . . 45

4 PVS-Cert: verifiable certificates for PVS-Core **47**
- 4.1 Formal presentation . . . 48
  - 4.1.1 Terms and judgements . . . 48

4.1.2 Typing rules . . . 49
4.2 The choice of PVS-Cert . . . 50
4.2.1 An extension of λ-HOL . . . 50
4.2.2 The expression of predicate subtyping . . . 51

5 Properties of PVS-Cert 53
5.1 Tools for the analysis of derivations . . . 53
5.2 Thinning and substitution . . . 57
5.3 The Church-Rosser property . . . 61
5.4 Stratification in PVS-Cert . . . 62
5.5 A type preserving reduction . . . 65
5.6 Uniqueness of types . . . 71
5.7 Additional observations on PVS-Cert . . . 72
5.7.1 PVS-Cert extends PVS-Cert$^-$ . . . 73
5.7.2 Defining conversion on contexts . . . 73

6 Strong normalization in PVS-Cert 77
6.1 Saturated sets . . . 79
6.2 Closure properties of SN . . . 80
6.3 Properties of saturated sets . . . 84
6.4 Set interpretation of terms . . . 86
6.5 Proof of strong normalization . . . 94
6.6 Normal forms . . . 102
6.7 Defining and using cut elimination in PVS-Cert . . . 105

7 Type-checking in PVS-Cert 109
7.1 Definitions . . . 110
7.2 Soundness . . . 113
7.3 Termination . . . 115
7.4 Completeness . . . 116

8 A conservative extension of higher-order logic 121
8.1 The choice of the translation . . . 124
8.2 The translation of expressions and stratified contexts . . . 127
8.3 Properties of λ-HOL . . . 130
8.3.1 Subderivations, renaming, thinning, substitution . . . 131
8.3.2 Technical properties in λ-HOL . . . 132
8.3.3 Specific properties of λ-HOL . . . 133
8.4 Soundness of the translation . . . 135
8.5 Conservativity of the translation . . . 141

9 Expressing PVS-Core in PVS-Cert **155**
9.1 From PVS-Cert to PVS-Core . . . 155
9.2 Expressing PVS-Core derivations as PVS-Cert judgement . . . 162
9.3 Soundness of the synthesis of certificates . . . 167

10 Transposing PVS-Cert results in PVS-Core **177**
10.1 Using PVS-Cert as a system of verifiable certificates for PVS-Core . . . 177
10.2 Transposing PVS-Cert properties in PVS-Core . . . 178
10.2.1 A provable proposition is well-typed . . . 178
10.2.2 Type preservation in PVS-Core . . . 178
10.2.3 Strong normalization in PVS-Core . . . 181
10.2.4 PVS-Core is a conservative extension of higher-order logic . . . 182
10.3 Using cut elimination in PVS-Cert to study PVS-Core logical properties . 184

11 Conclusion **187**
11.1 Summary of the main contributions . . . 187
11.2 Perspectives for PVS . . . 188
11.2.1 Extending PVS-Cert and PVS-Cert . . . 188
11.2.2 The problem of extracting certificates from PVS . . . 190
11.3 Other perspectives . . . 191

II Proof certificates in PVS **193**

12 Proof certificates in PVS **195**
12.1 Certificates as refinements of the PVS proof traces . . . 196
12.2 Proofs certificates in PVS . . . 197
12.2.1 Expressions and conversion . . . 197
12.2.2 Reasoning . . . 198
12.2.3 Proof objects . . . 201
12.3 Checking PVS proofs using Dedukti and Metitarski . . . 202
12.3.1 Translating proofs to Dedukti . . . 202
12.3.2 Checking assumptions with MetiTarski . . . 203
12.4 Results . . . 203
12.5 Conclusion . . . 204

III Expressing classical first-order logic in constructive systems **207**

13 A lightweight double-negation translation **211**
13.1 Introduction . . . 211
13.2 Syntax-directed double-negation translations . . . 213
13.3 Partial orders among double-negation translations . . . 215
13.4 The minimal translation . . . 217

13.5 Correctness of the minimal translation 220
13.6 Minimality of the translation 223
13.7 Conclusion 226

14 Automated constructivization of proofs **227**
14.1 Introduction 227
14.2 Notations and definitions 228
14.3 State of the art: two constructive fragments of predicate logic 231
14.4 The weakening normalization 232
14.5 A new constructive fragment 235
14.6 The full constructivization algorithm 237
14.7 Experimental results 242
14.8 Conclusion 243

Chapter 1

Introduction

In spite of its relatively short lifetime, the history of the formalization of mathematics from Frege's Begriffsschrift [30] to modern proof systems experienced several significant disruptions. Besides the radical changes brought with striking and sometimes unexpected theorems of inconsistency [65], incompleteness [42], or undecidability [15, 77], a slower yet equally profound revolution arose with the emergence and the enhancement of computers.

Before the development of these essential tools for the manipulation of formal languages, significant progresses were made to define universal languages for mathematics. The culmination of this programme was reached with Hilbert's and Ackermann's formalization of first-order logic [46]. They defined, once and for all, a universal language for mathematics, and reduced in this way any further formalization of mathematics to an exercise of axiomatization in this language. A similar situation was reached one decade later in the field of computability theory, where not one but several provably equivalent universal formalisms emerged at the same time [77, 16, 44].

Offering the possibility to use formal languages in the practice of proving and computing, the development of computers called this situation into question. Instead of the single criterion of universality, the expressivity of formal languages became considered as their ability to convey intuitions in the most concise and practical way. This shift shed a new light on logical systems admitting much simpler presentations when formalized as alternatives to first-order logic than when formalized as particular first-order logic theories. In particular, the idea of assigning types to mathematical expressions, which had been pioneered by Russell and Whitehead [80], was renewed.

Among the wide family of typed languages used in the contemporary proof systems, one of the most concise system – and also one of the oldest – is Church's simple type theory [17], also referred to as higher-order logic. This language is at the core of several proof systems used today, such as Isabelle/HOL [58], HOL Light [45], or HOL4 [69], in which Church's original language is revived with a surprisingly limited amount of

modifications and extensions. A more involved extension of higher-order logic is the language of the proof system PVS [60], which includes one major additional feature: predicate subtyping. This feature, which is at the core of the presented work, alleviates one of the main peculiarities of higher-order logic: the gap between types and predicates as two different kinds of attributes for mathematical expressions.

1.1 Types and predicates in higher-order logic

The formalism of higher-order logic is characterized by the coexistence of *typing judgements* and *proof judgements* and, through this distinction, by the assignment of two radically different kinds of attributes to mathematical expressions: *types* and *predicates*. For instance, the mathematical expression `1+1` can be assigned a type `nat` expressing that it is a natural number, or a predicate `Even` expressing that it is divisible by two. The first assignment corresponds to a typing judgement, whose validity is based on *typing rules*, while the second corresponds to a proof judgement, whose validity is based on *deduction rules*.

Types and predicates have very different roles in higher-order logic. On the one hand, typing is used as an *a priori* condition to reasoning: a valid proof judgement contains only well-typed expressions. For instance, if `TRUE` is an expression of type `bool`, then the ill-typed expression `1+TRUE` cannot occur in any valid proof judgement.

In higher-order logic, the use of types to restrict the domain of reasoning goes together with the rich reasoning capabilities resulting from the formalization of predicates as first-class objects, on which quantification is permitted. In particular, a large amount of set theoretic mathematics can be formulated in higher-order logic through the expression of sets as predicates. For instance, sets of natural numbers (such as the set of even numbers) can be formulated as predicates of domain `nat` (such as the predicate `Even`). The types of higher-order logic ensure that the formulation of Russell's paradox through this sets-as-predicates correspondence is ill-typed, and hence excluded from reasoning. Indeed, following the previous analogy, the belonging of a set to itself is expressed as the application of a predicate to itself: as the type and the domain of a predicate are never equal in higher-order logic, the application of a predicate to itself, including the formulation of Russell's paradox or simpler examples such as `Even(Even)`, cannot be well-typed.

As advocated in [26], this distinction between types and predicates is a conceptual difficulty as both express attributes of mathematical expressions. This difficulty is amplified by the considerable difference of expressivity between the two. Contrary to the assignment of predicates in proof judgements, the assignment of types in typing judgements remains very simple: in particular, well-typedness is decidable. In return, most attributes of mathematical expressions formulated as predicates cannot be formulated as types in higher-order logic. For instance, being a natural number different from `0` is expressible as a predicate, but not as a type. Such a type would be useful to exclude

expressions such as `1/(1-1)` from reasoning in the same way as done for `1+TRUE` or Russell's paradox, but it does not exist natively in higher-order logic.

1.2 Enriching types with predicate subtyping

The main idea of predicate subtyping is to recover a symmetrical situation between the expressivity of types and predicates as assignment of mathematical expressions. Predicate subtyping is defined as the addition of new types, referred to as *predicate subtypes*. Given a predicate `P` defined on a domain `A` (e.g. `Even` defined on the domain `nat`), the predicate subtype {`x : A | P(x)`} is defined. An expression `t` can be assigned this type if and only if it can be assigned the type `A` and `P(t)` is provable. For instance, if `Nonzero` is a predicate of domain `nat` expressing the difference of a natural number from `0`, then the numerical expression `1` can be assigned the type {`x : nat | Nonzero(x)`} as it can be assigned the type `nat` and `Nonzero(1)` is provable.

In the same way as the type {`x : nat | Nonzero(x)`} is defined from the predicate `Nonzero` of domain `nat`, it becomes possible to define a new type {`x : A | P(x)`} from any predicate `P` of domain `A`, and both express the same underlying attribute on mathematical expressions. This augmented expressivity of type assignments permits to exclude many unwanted expressions from reasoning. For instance, defining the denominators domain of Euclidean division as {`x : nat | Nonzero(x)`}, all divisions where the denominator is not provably different from zero become ill-typed. In this setting, the expression `1/(1-1)` becomes excluded from reasoning in the same way as `0 + TRUE` or the expression of Russell's paradox.

As expressions may have several types, predicate subtyping induces a form of subtyping: for instance, as any expression of type {`x : nat | Nonzero(x)`} also admits the type `nat`, the former can be considered as a subtype of the latter.

The main counterpart of this extension of higher-order logic is the fact that typing and provability become entangled. For instance, proving the equality `(1 / 1) = 1` requires that `1` can be assigned the type {`x : nat | Nonzero(x)`}, which, in turn, requires that `Nonzero(1)` is provable. In particular, contrary to the case of higher-order logic, typing is not decidable in the presence of predicate subtyping.

A definition of *well-formed* type becomes necessary as well to exclude ill-formed types. The well-formedness of types is also entangled with the well-typedness of expressions and the provability of formulas. For instance, the well-formedness of {`x : nat | (x/1) = 1`} requires, as in the previous example, that `1` can be assigned the type {`x : nat | Nonzero(x)`}, which requires in turn that `Nonzero(1)` is provable. On the other hand, {`x : nat | (x/0) = 1`} is an example of ill-formed type.

1.3 The practice of predicate subtyping in PVS

Predicate subtyping is not widely used among proof systems, with the notable exception of PVS. PVS (Prototype Verification System) [60] is an integrated environment for the development and the analysis of formal *specifications*. Its specification language is based on the extension of higher-order logic with predicate subtyping as well as many other features, such as recursion and induction. On top of its specification language, PVS is composed of two main tools.

- A *type-checker*, whose purpose is to recognize *well-formed specifications*, i.e. specifications composed of well-formed types and well-typed expressions.
- An interactive *prover*.

The presence of predicate subtyping is at the core of the main specificity of the PVS type-checker: the emission of *TCCs* (type-correctness conditions). Because of predicate subtyping, the well-formedness of types and the well-typedness of expressions may require complex proofs beyond the capabilities of the PVS type-checker: in such situations, the PVS type-checker outputs these requirements as specific formulas, the TCCs. After the execution of the type-checker, all TCCs have to be proved to consider the specification as well-formed.

The specificity of predicate subtyping and the emission of TCCs distinguishes PVS from other proof systems based on higher-order logic such as the systems of the HOL family (such as HOL Light [45] and HOL4 [69] for example), in which the well-formedness of a specification can be ensured once and for all before starting any proof. In this regard, PVS is comparable to more complex extensions of higher-order logic such as Coq [5] as, in both systems, the well-typedness of some expressions may depend on proofs – in Coq, this situation follows from the fact that proofs *are* expressions.

However, this dependency of well-typedness to proofs is formalized in a very different way in Coq. Contrary to PVS – and to the systems of the HOL family –, the specification language of Coq includes explicit proof terms, which are typed by their corresponding formulas, following the Curry-Howard isomorphism. In this setting, proof judgements are based on explicit proofs terms and become special cases of typing judgements: in particular, proof checking becomes a special case of type checking. This approach is not restricted to extensions of higher-order logic: it is also the core of alternative foundations of mathematics, such as Martin-Löf's intuitionistic type theory [56], and alternative proof systems, such as Agda [14]. In these latter systems, proof judgements and typing judgements become merged entirely.

The comparison between PVS and the Curry-Howard approach with explicit proof terms can be illustrated in the attempt to formalize Euclidean division. On the one hand, following the previous illustrations, Euclidean division can be formalized in PVS with two arguments, a numerator and a denominator, in such a way that the well-formedness

of a division `n/m` requires to prove that `m` is different from `0`. On the other hand, Euclidean division can be formalized in Coq or Agda with three arguments: a numerator `n`, a denominator `m`, and a proof term witnessing the fact that `m` is different from `0`.

As a consequence, the situation of PVS is intermediate between systems of the HOL family and Coq. In PVS, as well as in the systems of the HOL family, proofs are not formalized inside specifications: they are recorded as scripts outside specifications and cannot be recognized by the *type-checker*. However, in PVS, as in Coq, the type system is sufficiently expressive to make the well-typedness of some expressions depend on the proof of some formulas.

1.4 Contributions

This work is composed of three parts. The first one, which is also the most important one, is dedicated to the theoretical analysis of predicate subtyping.

1.4.1 First part: theoretical analysis of predicate subtyping

Higher-order logic, as well as its extension with predicate subtyping, can be defined in various ways. The first contribution of this work is the formalization, in Chapter 3, of a minimal system for predicate subtyping, denoted PVS-Core. This formalization is obtained starting from the practice of predicate subtyping in PVS, described in Chapter 2. Besides its minimality, the main design choice for this system is the introduction of a definitional equality, referred to as *conversion*, corresponding to syntactical equality modulo computation. Following Church's original presentation of higher-order logic, the *simple theory of types* [17], PVS as well as PVS-Core are based on λ-calculus. In this setting, computation is defined with a rewriting relation, β-reduction.

Starting from PVS-Core, the second contribution of this work is the formalization, in Chapter 4, of a language of verifiable proofs for PVS-Core. This new language, denoted PVS-Cert, is designed from PVS-Core with the addition of explicit proof terms, formalized as λ-terms, as well as the addition, at the level of expressions, of explicit coercions based on these proof terms. The addition of explicit proof terms follows the Curry-Howard isomorphism in the sense that PVS-Cert proofs terms are typed by their corresponding formulas. On the other hand, the addition of explicit coercions ensures the decidability of type-checking. We present, in Chapter 7, a terminating, sound and complete type-checking algorithm for PVS-Cert.

In order to maintain a simple correspondence between PVS-Core and PVS-Cert, *conversion* in PVS-Cert is not defined from β-reduction only but uses a more distinctive notion: syntactical equality modulo β-reduction *and coercion erasure* (Definition 4.1.3). The extension of β-reduction with coercion erasure is denoted $\rightarrow_{\beta*}$, and the corresponding conversion relation is denoted $\equiv_{\beta*}$. We present in Chapter 9 a translation

from PVS-Cert to PVS-Core and, at the level of derivation trees, a translation from PVS-Core to PVS-Cert. These translations are used in Chapter 10 together with the type-checking algorithm of PVS-Cert to define how to use PVS-Cert as a language of verifiable proofs for PVS-Core (Definition 10.1.1).

Remark 1.4.1. *The order in which the main contributions are presented in this section is chosen to provide an overview of Part I. As a consequence, it does not follow the precise order between the chapter composing this part. The ordering between chapters are chosen, on the one hand, to respect proof dependencies, and, on the other hand, to separate the work related to PVS-Core (Chapters 2 and 3) from the work related to PVS-Cert (Chapters 4 to 8) and the work related to the relations between the two systems (Chapters 9 and 10).*

PVS-Cert is very similar to the formalism of Pure Type Systems (PTSs) – see for instance [4] –, and more precisely to the formalism of PTSs extended with dependent pairs – see for instance the system ECC in [53]. The only characteristic of PVS-Cert distinguishing it from such systems lies in its conversion relation $\equiv_{\beta *}$. Instead of this specific conversion relation, PTSs with dependent pairs are equipped with a more standard but less flexible conversion relation ($\equiv_{\beta\sigma}$, described in Definition 4.2.2), based itself on a more standard definition of reduction ($\rightarrow_{\beta\sigma}$). Instead of the case of the reduction $\rightarrow_{\beta\sigma}$ in PTSs with dependent pairs, $\rightarrow_{\beta *}$ is not a type preserving reduction in PVS-Cert. We prove however, among the main properties of PVS-Cert presented in Chapter 5, that $\rightarrow_{\beta\sigma}$ is a type preserving reduction in PVS-Cert (Theorem 5.5.2). As a consequence, it defines, when applied to proof terms, a notion of *cut elimination*.

The strong normalization of the reductions $\rightarrow_{\beta *}$ and $\rightarrow_{\beta\sigma}$ is proved in Chapter 6 (Theorem 6.5.2) using a notion of *saturated sets* [73] which is adapted to both reductions. One of the main characteristics of this proof relies on the fact that the interpretation of terms as sets is not only defined on well-typed terms, but also on their iterated reducts under the relation $\rightarrow_{\beta *}$. In a more general setting, this proof opens the way to cut elimination theorems in PTS-like systems in which conversion and cut elimination are based on distinct notions of reduction.

While the termination of the reduction $\rightarrow_{\beta *}$ is at the core of the termination of the type-checking algorithm defined for PVS-Cert, the termination of the reduction $\rightarrow_{\beta\sigma}$ provides a cut elimination theorem (Theorem 6.5.3), which is a useful tool to analyze specific properties of PVS-Cert and PVS-Core – and thus predicate subtyping –, from consistency (Theorems 6.7.1 and 10.3.1) to more complex theorems such as the analysis of Leibniz's equality (Theorems 6.7.2 and 10.3.2).

Last, we present in Chapter 8 a translation (Definition 8.2.5) from PVS-Cert to its restriction to higher-order logic with explicit proofs, which corresponds precisely to the PTS λ-HOL defined in [4]. Using this translation and the fact that any PVS-Cert formula in which predicate subtyping is not explicitly used is translated as itself through this

translation, we conclude that PVS-Cert is a conservative extension of λ-HOL (Theorem 8.5.2), and as a consequence that PVS-Core is a conservative extension of higher-order logic (Theorem 10.2.4). These theorems allow to reduce the question of the provability of any proposition using predicate subtyping to the question of the provability of a proposition formulated in pure higher-order logic. Hence, they provide useful tools to study the properties of predicate subtyping, in complement of the cut elimination theorem. As detailed at the end of this introductive chapter, the conservativity of PVS-Cert over λ-HOL also opens perspectives related to the conjecture of conservativity of ECC [53] over higher-order logic.

1.4.2 Second part: extracting proof certificates from PVS

Although the extension of PVS-Cert to a system of proof certificates for the whole PVS proof system is left to future work, we present in Part II a first prototype for the extraction of proofs from PVS that can be verified externally. Following the dichotomy between the *type-checker* and the *prover* in PVS, this prototype is only suited to verify the reasoning steps performed in the prover, and does not contain any typing information at this stage. This proof extraction mechanism is built by instrumenting the PVS proof system itself. More precisely, the PVS prover is modified to record detailed proofs step by step during the proof search process. Proofs can be built for any PVS theory. However, some reasoning steps rely on unverified assumptions. For a restricted fragment of PVS, the proofs are exported to the universal proof checker Dedukti [66], and the unverified assumptions are proved externally using the automated theorem prover MetiTarski [1]. This work is published in [39]. A shared perspective from Part I and Part II is to turn this prototype into a complete system of certificates expressed in some extension of PVS-Cert, as discussed in Section 11.2.2.

1.4.3 Third part: expressing classical first-order logic in constructive systems

In the last part, we present two works from a very different research topic: the investigation of the relationship between classical and constructive logic in the framework of first-order logic. The first one is the definition of a lightweight double-negation translation from classical logic to constructive logic. It is published in [37]. The study of double-negation translation also lead to a contribution in a collaborative work [2] investigating how to express various theories in the proof checker Dedukti [11] – in which the contribution of the author is restricted to the definition of an encoding of classical predicate logic in Dedukti.

The second work of this final part is the definition of a constructivization algorithm, taking as input the classical proof of some formula and generating as output, whenever possible, a constructive proof of the same formula. It is published in [38]. The experiments performed in this work are based on the instrumentation of the first-order classical theorem prover Zenon [12] to output constructive proofs in the Dedukti [11]

format. This instrumentation is adapted from a preceding collaboration [22] dedicated to the extension of Zenon to the formalism of *deduction modulo* – in which the contribution of the author was restricted to the implementation of an output of this new system in the Dedukti format.

1.5 Related works

The introduction of predicate subtyping can be traced back to the first-order language OBJ2 [32] and its *sort constraints*, allowing to restrict some typing relations to the satisfaction of a predicate. This idea was later refined and combined with higher-order logic in the proof system PVS, which is the most important system based on predicate subtyping. Overviews of the PVS specification language and its use of predicate subtyping are given in particular in [60] and [64]. In the presented work, Chapter 2 presents a minimal subset of PVS restricted to predicate subtyping which relies explicitly on these works.

On the theoretical point of view, one of the most important works carried out on predicate subtyping is the presentation of *formal semantics* for PVS in [61]. This works defines, for some fragment of the PVS language including predicate subtyping but also other features such as *parametric theories*, set-theoretical interpretations of types and expressions. These interpretations are limited to *standard* interpretations: the interpretation of a function type is the set of all functions from the interpretation of the domain to the interpretation of the co-domain, and the interpretation of the type of propositions is a set containing exactly two elements, distinguishing *true* propositions from *false* ones. As a minimal extension of higher-order logic with predicate subtyping, the system PVS-Core is smaller than the fragment of PVS presented in [61]. As a consequence, it would seem possible to restrict the work presented in [61] to obtain a notion of *standard model* for PVS-Core. Such an analysis wouldn't be redundant with the present contribution but, on the contrary, complementary:

- Standard models provide very simple definitions of *true* and *false* propositions. They also reveal a very simple and natural way to interpret the extension of higher-order logic with predicate subtyping in set-theoretical terms. In particular, they offer a simple interpretation of predicate subtyping as a particular case of restricted comprehension, which has not been explored in the presented work. In comparison, the only set-theoretical interpretation of predicate subtyping presented in this work, which is the interpretation (Definition 6.4.4) used at the core of the proof of strong normalization and cut elimination in PVS-Cert, relies on a much more complex way of interpreting predicate subtyping in set-theoretical terms.

- On the other hand, a large part of the presented work is dedicated to the construction of tools to discriminate between *provable* and *unprovable* propositions, including a cut elimination theorem and a conservativity theorem over higher-order logic. In the case of predicate subtyping as well as in pure higher-order logic, the

simplicity of standard models admits a counterpart, which is incompleteness: some propositions that are *true* in all standard models may remain *unprovable*. As a result, the use of standard models is very limited whenever one wants to discriminate *provable* from *unprovable* propositions, which is the central question of this work. Although they may be sufficient to answer some simple questions of provability, such as the question of the consistency of predicate subtyping, they cannot be used for more complex questions, such as the analysis of Leibniz's equality developed in Theorem 10.3.2.

Another important related work is [10], in which two systems are presented: ICC_Σ, a type system with *implicit* type constructions, and AICC_Σ, a system obtained from ICC_Σ by adding *explicit* coercions. ICC_Σ contains several advanced features, including a generalization of predicate subtypes named *subset types*. The construction of PVS-Cert from PVS-Core follows the same idea as the construction of AICC_Σ from ICC_Σ in [10]: adding the missing information explicitly in the terms of the language to recover the decidability of type-checking. The main difference between the two approaches lies in the complexity of the respective languages. ICC_Σ is a very rich and complex language, making its analysis difficult – in particular, strong normalization in ICC_Σ is kept as a conjecture, on which the decidability of type-checking itself relies. Conversely, PVS-Core is designed as a minimal language including predicate subtyping, making its analysis much simpler.

A variant of predicate subtyping was also formalized as an extension of the calculus of constructions in [70]. In the same way as in the present work, this presentation contains two systems connected with each other. On the one hand, it includes one system, named Russell, which is comparable to a weakened version of PVS-Core in which a term t of type A admits the type $\{x : A \mid P\}$ even when $P[t/x]$ is not provable. In this variant of predicate subtyping named *subset equivalence*, type-checking is decidable. On the other hand, this work includes a system with explicit coercions which is comparable to PVS-Cert. Contrary to PVS-Core, Russell derivations are not intended to contain all information necessary to build complete terms with explicit coercions: instead, a translation producing incomplete terms in the system with explicit coercions is presented. This system allows to write programs and specifications together in Russell, and to prove their correctness in a second step by filling all proof holes produced through the translation, in a way which is similar to what can be done using PVS and its TCCs.

Contrary to the case of PVS-Core and Russell, PVS-Cert and the counterpart of Russell with explicit coercions have similar characteristics. The theoretical properties of this latter system are not formalized, but it is presented as a simple extension of the proof-irrelevant type theory presented in [79]. There exists indeed a tight connection between proof irrelevance and the explicit version of predicate subtyping formalized as PVS-Cert: if one considers for instance the usual predicate $Even$ on natural numbers expressing divisibility by two, the predicate subtype $even = \{x : nat \mid Even(x)\}$, and two expression with explicit coercions $\langle 2, p\rangle_{even}$ and $\langle 2, q\rangle_{even}$ of this type with p and q two proofs of $Even(2)$, then the hypothesis of proof irrelevance, which ensures that

p and q are convertible, also ensures that the expressions $\langle 2, p\rangle_{even}$ and $\langle 2, p\rangle_{even}$ are convertible, which is the main requirement expected from explicit coercions to reflect predicate subtyping faithfully.

This relation between proof irrelevance and predicate subtyping is explored further in [79]. Besides the fact that this work is based on the calculus of constructions and besides some technical differences in the precise definition of conversion between the system presented in this paper and PVS-Cert, analyzing the strong relation between these two systems appears as a very interesting future work. In particular, it would provide a natural strategy for attempting to prove the strong normalization conjecture presented in the paper.

PVS-Cert is an adaptation of the formalism of Pure Type Systems (PTSs) – sometimes also referred to as Generalized Type Systems (GTSs) –, presented for instance in [4]. The definition of PTSs is itself the result of several successive works, including notably [7, 74, 75, 76, 36, 3]. More specifically, PVS-Cert is derived from the notion of PTSs *with dependent pairs*, which has its roots in the system ECC, presented in [53]. A subsystem of PVS-Cert, named PVS-Cert$^-$ and presented in Chapter 4 (Definition 4.2.3), even appears as a fragment of ECC. As the conjecture of conservativity of ECC over higher-order logic remains an open problem, an interesting extension of the presented work would be to investigate to which extent the proof of conservativity presented for PVS-Cert can be expanded to some larger fragment of ECC than simply PVS-Cert$^-$.

Part I

Theoretical analysis of predicate subtyping

Chapter 2

Predicate subtyping in PVS

In this chapter, we present a minimal fragment of the PVS specification language containing both higher-order logic and predicate subtyping, which will be the starting point of the formalization of the system PVS-Core presented in Chapter 3. We also present, in an idealized way, how typing and deduction can be defined on this subset of the specification language. This idealization is the result of several layers of abstraction and simplifications, starting from the reality of the PVS type-checker and the PVS interactive prover.

The purpose of this introductory chapter is twofold. On the one hand, it provides a brief overview of the system PVS, which can be considered as the most important implementation of predicate subtyping in a proof system. On the other hand, it provides some justifications and perspectives for the system PVS-Core presented in Chapter 3. More precisely, it shows how the formalism of PVS-Core may emerge from a drastic simplification and idealization of PVS. However, the definitions given in this chapter are presented in an informal way, and all assertions occurring in it are kept at the at the state of conjectures: the actual contributions of the presented work begin at Chapter 3, with the presentation of PVS-Core.

2.1 Presentation of a subset of the specification language

This section is dedicated to the presentation of a minimal subset of the PVS specification language containing higher-order logic and predicate subtyping. After the presentation of the structure of PVS specifications on top of types and expressions, we detail this language in two successive layers: first, a minimal subset of PVS containing higher-order logic, and then, a minimal additional layer containing predicate subtyping. In the following of this chapter, we use `verbatim` notations to refer to the PVS specification language, and *italic* notations to refer to definitions of the PVS documentation [62].

2.1.1 General structure of PVS specifications

A PVS *specification* is a collection of *theories*. In this selection of the language, we will considered here only specifications composed of one single theory. This restriction excludes several features of PVS, such as the possibility of importing theories, as well as the mechanism of abstraction through *formal parameters* associated with it – this feature includes in particular a mechanism of abstraction at the level of types, widely used in practice, which is discussed as a possible extension of the present work in Section 11.2.1. The present restriction also excludes the accessibility to the PVS standard library, which is referred to as the *prelude*. The mechanism of *name resolution*, which is particularly useful in the development of specifications involving several theories, is also abstracted.

In this simplified setting, a *theory* consists, in turn, of a list of *declarations*. Hence, in the following of this chapter, the notion of *specification*, *theory*, and list of *declarations* will be considered equivalent. Declarations are themselves composed of *types* and *expressions*. We will first present a fragment of the language of declarations, before detailing a fragment of the language of types and a fragment of the language of expressions. In each case, we select only the minimal subset of the language including predicate subtyping.

In a minimal presentation of higher-order logic with predicate subtyping, we select only the three following kinds of declarations.

- *Uninterpreted type declarations* allow to declare new types. The uninterpreted declaration of a new type `X` is written `X : TYPE` in PVS.
- *Uninterpreted constant declarations* allow to declare new expressions. The uninterpreted declaration of a new constant `x` in a type `A` is written `x : A` in PVS.
- *Formula declarations* allow to declare new axioms or new theorems. The declaration of an axiom `P` of name `h` is written `h : AXIOM P` in PVS. The declaration of a theorem `P` of name `h` can be written in many equivalent ways in PVS, among which `h : THEOREM P`.

On the theoretical point of view, the most important part of PVS excluded from this restriction is its set of features based on induction: *datatypes* (which can be defined in PVS either as a alternatives to a theories or as declarations), *inductive definitions* of predicates, and *recursive definitions*.

However, this restriction also excludes other kinds of declarations which do not have a large impact on the theoretical standpoint, but are very useful in practice: *interpreted declarations*, which allow to define types or constants with existing types or expressions, *variable declarations*, which express universal quantification for all formula declarations in a concise way, *judgements*, which allow to factor the emission of TCCs by the typechecker, *macros*, which consist in automatically unfolded definitions, *conversions*, which

are the core of a system of implicit casts, or *auto-rewrite declarations*, which are commands used by the prover to guide proof-search automatically. Last, this restriction excludes alternative *type declarations*, which also do not have a large impact on the theoretical standpoint but are very useful in practice, such as *subtype declarations*, *nonempty type declarations*, or type declarations with *containing clauses*.

2.1.2 Higher-order logic types and expressions

The languages of types and expressions in PVS is rich: we present here only a minimal selection of them allowing to express higher-order logic. We present first the following type constructions, which defines the language of simple types.

- *Base types.* We distinguish the type of formulas, written `bool` (or `boolean`) in PVS, from all other base types, which are the types declared in *uninterpreted type declarations*.

  The name `bool` highlights the fact that formulas can be used as booleans in PVS. Indeed, as it will be presented in the following, `bool` admits two provably different constants `TRUE` and `FALSE`, and combining classical reasoning with propositional extensionality – as both are permitted by the PVS prover –, the following elimination principle is provable: `FORALL (P : bool) : P = TRUE OR P = FALSE`. More details on these two proving capabilities will be provided in the presentation of an idealized definition of provability for the selected subset of PVS (Section 2.2.2).

- *Function types.* Given two types `A` and `B`, the function type `[A -> B]` represents functions from `A` to `B`. In the special case where `B` is `bool`, such functions correspond to predicates on `A`. This type construction is at the core of higher-order logic. However, in the presence of predicate subtyping presented in the following section, it will be eventually replaced by a more general construction in PVS, the construction of *dependent function types*.

This selection excludes other type constructions exceeding a minimal presentation of higher-order logic: *tuple types*, which correspond to Cartesian products, and *record types*, which correspond to *tuple types* in which the components are referred to by names instead of indexes. The corresponding expressions (*tuple expressions*, *projection expressions*, *record expressions*, and *record accessors*) are also excluded.

We present the following selection of expression constructions.

- *Formulas based on connectives.* Formulas can be based on the constant connectives `TRUE` and `FALSE`, the unary connective `NOT`, and the binary connectives `AND`, `OR`, `IMPLIES`, `WHEN`, and `IFF`. Contrary to all other connectives, `WHEN` is defined: in PVS, the expression `P WHEN Q` is a notation for `Q IMPLIES P`. As a consequence, we first exclude it from the presentation of a minimal presentation of higher-order

logic.

In higher-order logic, and especially classical higher-order logic, there are many different ways to factor connectives further. At first glance, it seems that any such factorization could be applied in the case of PVS, at its interactive prover extends classical higher-order logic. However, as it will appear in the idealized presentation of an idealized definition of typing (Section 2.2.1), the usual equivalences between connectives do not always hold in presence of predicate subtyping. We present here the following factorization, allowing to keep `IMPLIES` as a the only primitive connective, and hence to exclude all other connectives from a minimal presentation of higher-order logic extended with predicate subtyping.

- `FALSE` is defined as `FORALL (P : bool) : P`.
  It uses universal quantification, which will be defined after connectives.
- `TRUE` is defined as `FALSE IMPLIES FALSE`.
- `NOT P` is defined as `P IMPLIES FALSE`.
- `P AND Q` is defined as `NOT (P IMPLIES NOT Q)`.
- `P OR Q` is defined as `(NOT P) IMPLIES Q`.
- `P IFF Q` is defined as `(P IMPLIES Q) AND (Q IMPLIES P)`.

The compatibility of these definitions with the usual deduction rules of classical higher-order logic corresponding to connectives is standard. On the other hand, their compatibility with the typing rules of predicate subtyping will be justified together with the presentation of typing rules in Section 2.2.1.

- *Quantified expressions.* There are two kinds of quantified expressions in PVS: universal quantification, written `FORALL (x : A) : P`, and existential quantification, written `EXISTS (x : A) : P`. In the following, we exclude the existential quantification `EXISTS (x : A) : P`, as, in classical reasoning, it can be expressed as `NOT FORALL (x : A) : NOT P`.

  We also exclude in this selection quantified expressions with several bindings, such as `FORALL (x : A, y : B) : t`.

- *Constants and variables.* Constants are the names declared in *constant declaration*, while variables are introduced in *binding expressions*, such as the quantified expressions. These two notions will be considered as equivalent in the following.

- *λ-expressions.* These expressions are the constructors of function types. A λ-expression is written `LAMBDA (x : A) : t`.

- *Applications.* These expressions are the eliminators of function types. An application is written `f(t)`.

This selection excludes *equalities*. An equality between two terms `t` and `u` is denoted `t = u`. However, once type-checked, it is equipped internally with a type `A`, which is such that both `t` and `u` can be assigned the type `A`. In this setting, the internal recording of such an equality matches Leibniz's definition of equality `FORALL (P : [A -> bool]) : P(t) IMPLIES P(u)`, and hence can be eliminated from a minimal presentation of higher-order logic. We also exclude *disequalities*, which, although primitive in PVS, are equivalent to negations of equalities.

More importantly, this selection also excludes *conditional expressions*, which have a central place in the practice of PVS and admit several forms: *if-then-else expressions*, denoted `IF P THEN t ELSE u ENDIF`, but also more involved conditional expressions such as `COND` expressions, `TABLE` expressions, or *override* expressions (written with the keyword `WITH`).

Last, *numeric expressions* are also excluded from this selection, as they rely in the largest part on the theories described in the PVS *prelude*, which was itself excluded from this presentation. In a similar way, *coercion expressions*, which consist in explicit type annotations, are excluded as well as they rely on the mechanism of *name resolution* which was itself excluded from this presentation.

Remark 2.1.1. *As presented in [61], the addition of equalities together with if-then-else expressions permits to define all connectives, in a way that is consistent with the specific type-checking rules of connectives in the presence of predicate subtyping. Hence, it leads to a system which is comparable to the minimal subset of PVS presented in this work, in the sense that it can be presented in a very economic way. However, beyond the fact that the subset presented in this work contains less expression constructions, its choice is mainly due to the fact that it is much simpler to define a language of formal proofs equipped with a cut elimination theorem in this language.*

2.1.3 Addition of predicate subtyping

We present the extension of the previous language to a minimal definition of predicate subtyping simply with the addition of the following type constructions.

- *Predicate subtypes.* A predicate subtype is written `{x : A | P}`, where `A` is a type and `P` is an expression.
- *Dependent function types.* A dependent function type is written `[x : A -> B]`, where `A` and `B` are types. It replaces the construction of *function types* defines in the previous section, as a function type `[A -> B]` can be written as the dependent function type `[x : A -> B]`.

These two types are binders: in the type `{x : A | P}` (resp. in the type `[x : A -> B]`), the expression variable `x` may occur in `P` (resp. in `B`).

The replacement of function types with dependent function types is kept in this minimal presentation as, in the presence of predicate subtyping, it does not really extend the system: on the contrary, it simplifies the definition of well-typed expressions. More details will be given on this question with the presentation of an idealized definition of typing in Section 2.2.1.

Before detailing idealized definitions of typing and deduction for this selection of the PVS specification language, we can summarize their main distinctive characteristics, due to predicate subtyping, in the following principle: an expression `t` also admits the type {`x : A | P`} if and only if `t` admits the type `A` and `P[t/x]` is provable (where `P[t/x]` denotes the substitution of `x` by `t` in `P`). This principle will be declined in two typing rules, and one deduction rule:

- Whenever an expression `t` admits the type `A` and `P[t/x]` is provable, `t` also admits the type {`x : A | P`}.
- Whenever an expression `t` admits the type {`x : A | P`}, `t` admits the type `A`.
- Whenever an expression `t` admits the type {`x : A | P`}, `P[t/x]` is provable.

As it will be detailed in Section 2.2.1, additional typing rules should be added to capture the complexity of typing in PVS. Yet, these three rules will be kept in the following as the core of the formalization of predicate subtyping.

2.2 Idealization of typing and provability in PVS

This section presents a first step towards the idealization of the notions of typing and provability in the subset PVS presented in the previous section (Section 2.1). In this setting, typing rules (and well-formedness rules) will be presented only in an informal way, as an introduction of the formal system PVS-Core defined in the following chapter (Chapter 3).

As mentioned in the introduction (Chapter 1), the most important characteristic of predicate subtyping is the fact typing and provability become entangled. In PVS, where the type-checker is in charge of typing verifications while the interactive prover is in charge of proof verifications, this entanglement is the main reason of the fact that the type-checker's output is not binary: whenever a typing condition requires some specific proof, a new formula declaration, named *TCCs* or *proof obligation*, is generated. A TCC is printed `h : OBLIGATION P`, where `h` is the name of the TCC and `P` the corresponding formula. In such situations, a specification is considered well-formed only when all corresponding TCCs have been proved.

Although TCCs are not printed explicitly inside *theories*, they are part of PVS specifications: internally, they are recorded together with other declarations. More precisely,

each TCC is generated during the type-checking of some specific declaration, and is placed immediately above.

2.2.1 Idealization of typing and well-formedness

As the PVS type-checker emits TCCs, the idealization of typing and well-formedness presented in this section does not correspond to a idealized description of the type-checker, but to a idealized description of well-typed expressions, well-formed types and well-formed specifications *a posteriori*, i.e. once all TCCs have been generated and inserted in specifications. As a consequence, the data of the typing rules presented here is not sufficient to present an idealized version of the type-checker, as it does not describe how TCCs are generated. For instance, the rules presented here do not predict the generation of TCCs ensuring that constants are declared only in types that are provably non-empty, as it can be the case in the specification composed of the following declarations:

- `X : TYPE`
- `x : X`

However, as presented in the introduction (Chapter 1), typing depends on provability in the presence of predicate subtyping. As a consequence, some typing rules presented in this idealization depend on proofs, and in such cases, their correspondence to the reality of PVS appears only through the way *subtype TCCs* (i.e. TTCs related to predicate subtyping) are generated in PVS.

Type-checking some specification consists in the verification that every type appearing in it is well-formed, and that every expression appearing in it is well-typed. These two notions are not absolute: some expression may be well-typed in some context, but not in another. In the PVS type-checker, this notion of context is complex: it relies on the data of several global variables, which are updated all along the type-checking process. The most important part of the idealization of PVS presented in this work is the idealization of type-checking contexts as list of declarations: in this setting, *contexts* and *specifications* are identified as they are both defined as lists of declarations. This identification will be the cornerstone of the formalization of PVS-Core presented in the next chapter (Chapter 3).

Using this notion of *context*, we define the well-formedness of a *specification*, which is an absolute notion, as the well-formedness of each declaration composing it relatively the context formed by all declarations appearing strictly above it. In particular, in this idealized definition of well-formedness, all TCCs that have been generated during the inspection of some declaration by the PVS type-checker appear as *formula declarations* in the context associated to this declaration.

In turn, the well-formedness of declarations is idealized with the following rules. They use the notion of typed expressions and well-formed types, which will be presented after.

- A type declaration `X : TYPE` is well-formed in some context if the type name `X` was not previously declared in this context.
- A constant declaration `x : A` is well-formed in some context if the constant name `x` was not previously declared in this context and `A` is well-formed in this context.
- A formula declaration `h : AXIOM P`, `h : THEOREM P`, or `h : OBLIGATION P` is well-formed in some context if the constant name `h` was not previously declared in this context and `P` admits the type `bool` in this context.

The well-formedness of types is defined as follows.

- The base type `bool` is well-formed in any well-formed context.
- Any base type `X` is well-formed in some context whenever this context is well-formed and contains the declaration `X : TYPE`.
- A predicate subtype {`x : A | P`} is well-formed in some context if `A` is well-formed in this context and `P` admits the type `bool` in this context augmented by the constant declaration `x : A`.
- A dependent function type `[x : A -> B]` is well-formed in some context if `A` is well-formed in this context and `B` is well-formed in this context augmented by the constant declaration `x : A`.

The rule corresponding to predicate subtypes reveals that, as underlined in the introduction (Section 1), the definition of the well-formedness of types depends on the definition of typability of expressions.

Finally, the typability of expressions is defined, on the one hand, by rules that are associated with the different expression constructions, and, on the other hand, by the rules that are specific predicate subtyping. We first present the rules associated to the different expressions constructions. As done in the previous sections, we use the notation `t[u/x]` for substitution.

- Formulas based on connectives. An expression `P IMPLIES Q` admits the type `bool` in some context if `P` admits the type `bool` in this context and `Q` admits the type `bool` in this context augmented by a formula declaration of `P`.

  This rule is more flexible than the usual higher-order logic rule in which the typing context of `Q` is not augmented by a formula declaration of `P`. For instance, in a formalization of arithmetic in PVS in which division by zero is excluded using predicate subtyping (as presented in Section 1.2), the expression `FORALL (n : nat)`

`: n > 0 IMPLIES n/n = 1` is well-typed because of this added flexibility.

In order to justify the compatibility with predicate subtyping of the factorization of PVS connectives presented in Section 2.1.2, we also present the idealized rules corresponding to the type-checking of the other primitive connectives in PVS.

- `TRUE` and `FALSE` admit the type `bool` in any well-formed context, `NOT P` admits the type bool in some context `P` does, and `P IFF Q` admits the type `bool` in some context whenever `P` and `Q` do.
- `P AND Q` admits the type `bool` in some context whenever `P` admits the type `bool` in this context and `Q` admits the type `bool` in this context augmented by a formula declaration of `P`.
- `P OR Q` admits the type `bool` in some context whenever `P` admits the type `bool` in this context and `Q` admits the type `bool` in this context augmented by a formula declaration of `NOT P`.

The rules corresponding to conjunction and disjunction recall, in another setting, the variants *and then* and *or else* of boolean programming, in which the execution of the second expression is conditioned by the result of the execution of the first. In the same way as in the case of implication, they introduce some flexibility in typing. For instance, these rules ensure the well-typedness of the expressions `FORALL (n : nat) : n > 0 AND n/n = 1` and `FORALL (n : nat) : n = 0 OR n/n = 1`.

The definitions of connectives presented in Section 2.1.2 are indeed compatible with these rules:

- In the cases `TRUE`, `FALSE`, `NOT P`, and `P IFF Q`, there is no compatibility constraint related to these rules.
- In the case `P AND Q`, one must verify that `P` is accessible when type-checking `Q`, which is the case in the definition `NOT (P IMPLIES NOT Q)`.
- In the case `P OR Q`, one must verify that `NOT P` is accessible when type-checking `Q`, which is the case in the definition `(NOT P) IMPLIES Q`.

- A quantified expression `FORALL (x : A) : P` admits the type `bool` in some context whenever `P` does in this context augmented by the constant declaration `x : A`.
- A constant or variable (these notions being equivalent in this idealization) `x` admits the type `A` in some context whenever this context is well-formed and contains the declaration `x : A`.
- A λ-expression `LAMBDA (x : A) : t` admits the type `[x : A -> B]` in some context whenever `t` admits the type `B` in this context augmented by the constant declaration `x : A`.

- An application `f(t)` admits a type defined as a substitution `B[t/x]` in some context whenever `f` admits a type of the form `[x : A -> B]` and `t` admits the type `A` in this context.

Through this idealized presentation of typing, it appears, as claimed in Section 2.1.3, that the replacement of function types by dependent function types does not really extend the system, but on the contrary simplifies the definition of well-typed expressions. Indeed, without this generalization, the typing rule corresponding to λ-expressions would be more complex: a λ-expression `LAMBDA (x : A) : t` would admit the type `[A -> B]` in some context not only whenever `t` admits the type `B` in this context augmented by the constant declaration `x : A`, but also when `B` is well-formed in this context (and, as a consequence, does not contain any free occurrence of `x`).

The presentation of the typing rules corresponding to the expression constructions being established, the formalization of typing is completed with the presentation of additional rules corresponding to predicate subtyping. We begin with the two typing rules stated in the general presentation of predicate subtyping (Section 2.1.3), formulated relatively to typing contexts.

- An expression `t` admits the type $\{$`x : A | P`$\}$ in some context whenever `t` admits the type `A` in this context and `P[t/x]` is provable in this context.
- An expression `t` admits the type `A` in some context it admits some type of the form $\{$`x : A | P`$\}$ in this context.

These rules are not sufficient to capture the complexity of typing in PVS, even in the presented restriction of the specification language. We conjecture that the addition of two extra rules is sufficient to fill the most part of this gap. These extra rules require a formal definition of the subtype relations, which can be done from the typing relations as follows: in a given context, a type `A` is a subtype of `B` if, given a new constant `x`, `x` admits the type `B` in the context augmented by the constant declaration `x : A`. Using this notion of subtype relation, these two additional rules are the following.

- An expression `t` admits the type `[x : A -> B]` in some context whenever it admits a type of the form `[x : A -> C]` in this context, and `C` is a subtype of `B` in this context augmented by the constant declaration `x : A`.
- An expression `t` admits the type `[x : A -> B]` in some context whenever it admits a type of the form `[x : C -> B]` in this context, `A` is a subtype of `C` in this context, and `C` is a subtype of `A` in this context.

We present two examples where these rules are useful, in the context formed by the following declarations:

- `nat : TYPE`
- `Even : [x : nat -> bool]`

- `Odd : [x : nat -> bool]`
- `axiom: AXIOM FORALL (x : nat) : Even(x) IFF NOT Odd(x)`
- `double : [x : nat -> {y : nat | Even(y)}]`
- `half : [x : {y : nat | Even(y)} -> nat]`

Using the axiom and the two rules, we can conclude the following statements.

- `double` admits the type `[x : nat -> {y : nat | NOT Odd(y)}]`
- `double` admits the type `[x : nat -> nat]`
- `half` admits the type `[x : {y : nat | NOT Odd(y)} -> nat]`

As illustrated in these examples, the first of these two rules can be interpreted as the covariance of the subtype relation in the ranges of dependent function types, while the second rule can be interpreted as the equivariance of the subtype relation in the domains of dependent function types. This equivariance is more restrictive than a simple contravariant criterion. For instance, one can not conclude in this context that `double` admits the type `[x : {y : nat | Even(y)} -> {y : nat | Even(y)}]`. This corresponds to the fact that, in PVS as in set theory, the domain corresponding to the expression of a function is unique – more precisely, in PVS, it is unique modulo the equivalence induced from the subtype relation.

However, in the following of this work, these two rules will be excluded from the formalization of predicate subtyping, for two reasons. On the one hand, they make the formal analysis of predicate subtyping much more complex, particularly in the perspective of defining a language of formal proofs equipped with a cut elimination theorem. On the other hand, we conjecture that their use can be avoided by replacing some expressions by appropriate η-expansions. For instance, in the previous context, the following statements hold without using these two rules.

- `LAMBDA (x : nat) : double(x)` admits the type `[x : nat -> {y : nat | NOT Odd(y)}]`
- `LAMBDA (x : nat) : double(x)` admits the type `[x : nat -> nat]`
- `LAMBDA (x : {y : nat | NOT Odd(y)}) : half(x)` admits the type `[x : {y : nat | NOT Odd(y)} -> nat]`

Yet, we will keep a much weaker form of conversion in this idealization of predicate subtyping: conversion modulo β-equivalence. This rule is defined as follows: an expression `t` admits some type `B` in some context whenever `B` is well-formed in this context and

`t` is admits some type `A` β-equivalent to `B` in this context.

Using the former context, this allows to conclude that `double` admits the type `[x : nat -> {y : nat | (LAMBDA (z : NAT) : Even(z)) (x)}]`. Unlike the two previous conversion rules, keeping this weak form of conversion won't burden the formalization of proof certificates and the definition cut elimination. On the contrary, it allows to consider expressions and types *modulo β-equivalence*, which will be an important characteristic of the two formal systems PVS-Core and PVS-Cert presented in this work.

2.2.2 Simplification of the PVS sequent calculus with natural deduction

In the restriction of PVS to higher-order logic presented in Section 2.1.2, the proving capabilities of PVS match classical higher-order logic, including both propositional extensionality and functional extensionality. Before presenting an idealized version of deduction in the presence of predicate subtyping, we first describe how proofs are performed in PVS.

The PVS interactive prover is based on a sequent calculus. The details of its usage are described in the PVS prover guide [68]. Sequents are at the core of the interaction between the prover and its user. Sequents are printed as a pair of lists of formulas: a list of *antecedents* and a list of *succedents*. However, internally, a sequent corresponds to a single list of formulas containing the succedents and all negations of antecedents. For instance, a sequent appearing as

```
[-1] NOT P1
[-2] P2
 |-------
[1] Q
```

is recorded internally as some permutation of the list `NOT NOT P1, NOT P2, Q`. Although recorded as lists, sequents are manipulated as multisets: the ordering between formulas has no influence on the deduction rules that can be applied to them. Sequent calculus has many advantages in the practice of PVS, as it is well-suited both in the setting of interactive and automated proving. Moreover, it allows to perform classical reasoning without adding any corresponding axiom.

However, the formalization of deduction rules is complex in the setting of sequent calculus, as such deductions would involve both sequents and contexts – the first one being a multiset of formulas and the latter a list of declarations. Moreover, the final purpose of this work is to present a formal language of proofs equipped with a notion of proof cut. Such definitions for sequent calculus, such as Curien and Herbelin's system [21], are much more complex than there counterpart in natural deduction. In particular, the definition of proof cut defined for classical sequent calculus is not confluent.

As a consequence, we will consider an idealization of PVS definition provability with deduction rules based on natural deduction instead of sequent calculus, and in which a proof judgement is simply the data of a single formula together with a context. We conjecture that, as it is the case in higher-order logic, provability in sequent calculus matches provability in natural deduction in the presence of predicate subtyping, at the condition of augmenting all contexts with an axiom permitting classical reasoning. We suggest for instance the following double negation elimination law in the case of PVS: `FORALL (P : bool) : (NOT NOT P) IMPLIES P`. However, as we seek to present a minimal system allowing the expression of predicate subtyping, this axiom won't be added by default.

As the previous definitions given for the connectives `AND` and `OR` as well as the definition of existential quantification only hold in classical reasoning, they are ensured to apply only if the previous axiom is added to the considered the context. However, it is possible to define alternative definitions which also hold in a constructive setting, taking the usual impredicative definitions:

- `P1 AND P2` is defined as `FORALL (Q : bool) : (P1 IMPLIES (P2 IMPLIES Q)) IMPLIES Q`.
- `P1 OR P2` is defined as `FORALL (Q : bool) : (P1 IMPLIES Q) IMPLIES ((P2 IMPLIES Q) IMPLIES Q)`.
- `EXISTS (x : A) : P` is defined as `FORALL (Q : bool) : (FORALL (x : A) : P IMPLIES Q) IMPLIES Q`.

However, the second definition is not compatible with PVS typing rules, which are intrinsically linked to classical reasoning. Indeed, in this definition, the type-checking context of `P2` does not contain the formula declaration of `NOT P1`, but only the formula declaration of `P1 IMPLIES Q`.

Two other proof capabilities in PVS will be excluded from the presentation of a minimal system for predicate subtyping as, both in the setting of sequent calculus and in the setting of natural deduction, they can be expressed as axioms, and hence do not need to be present by default. The first one is propositional extensionality, and the second one is functional extensionality. These principles can be expressed respectively by adding axioms of the following schemes.

- `(P IFF Q) IMPLIES P = Q`
- `(FORALL (x : A) : f(x) = g(x)) IMPLIES f = g`

The deduction rules selected in the minimal formalization of higher-order logic extended with predicate subtyping can be presented informally as follows.

- An axiom rule allows to prove a formula `P` in a given context as long as this context is well-formed and contains `P`.

- The usual introduction rules and eliminations rules corresponding to the implication connective are defined.
  - A formula `P IMPLIES Q` is provable in some context if `Q` is provable in this context augmented by the addition of `P` as an axiom.
  - A formula `Q` is provable in some context if two formulas of the form `P` and `P IMPLIES Q` are provable in this context.
- The usual introduction rules and eliminations rules corresponding to universal quantification are defined.
  - A formula `FORALL (x : A) : P` is provable in some context if `P` is provable in this context augmented by the addition of the constant declaration `x : A`.
  - Using the notation for substitution presented previously, a formula of the form `P[t/x]` is provable in some if a formula of the form `FORALL (x : A) : P` is provable and the term `t` admits the type `A` in this context.
- The rule corresponding to β-reduction is presented in the following way. A formula `P` is provable in some context if it is well-typed in this context, and if `P` is β-equivalent to some formula `Q` which is provable in this context.

Finally, we add one deduction rule which is specific to predicate subtyping, and corresponds to the third principle given in the presentation of predicate subtyping in Section 2.1.3: using again the notation for substitution presented previously, a formula of the form `P[t/x]` is provable in some context if `t` admits some type of the form {`x : A | P`} in this context.

Chapter 3

PVS-Core: the formalization of predicate subtyping

This chapter is dedicated to the first contribution of this work: the formalization of a minimal system for predicate subtyping. This system, denoted PVS-Core, is designed following the idealization of PVS described in Chapter 2. Besides its correspondence to the fragment of PVS described in this previous chapter, the main design choice for PVS-Core is the introduction of a definitional equality, referred to as *conversion*, corresponding to syntactical equality modulo β-reduction.

3.1 Terms and judgements

We first define PVS-Core **variables** and **terms**.

Definition 3.1.1 (Variables and terms)**.** *We define the set of* ***variables*** $\mathcal{V}$ *as the disjoint union of two infinite countable sets of symbols* $\mathcal{V}_{expressions}$ *and* $\mathcal{V}_{types}$*. We introduce the generic notation* v *or* w *to refer to a variable in general, as well as the following specific notations:*

- *The notation* X *or* Y *refers to variables in* $\mathcal{V}_{types}$*.*
- *The notation* x *or* y *refers to variables in* $\mathcal{V}_{expressions}$*.*

Then, we define a set of ***terms*** *as the disjoint union of the three following sets. The last two are defined together recursively.*

- *The first set contains a unique symbol:* $Type$*.*
- *The second set is the set of* ***types****. It is given with the following grammar:* $A, B := X \mid Prop \mid \Pi x : A.B \mid \{x : A \mid P\}$
- *The last set is the set of* ***expressions****. It is given with the following grammar:* $t, u, P, Q := x \mid \forall x : A.P \mid P \Rightarrow Q \mid \lambda x : A.t \mid tu$

There is no formal distinction between the expressions denoted t or u and the expressions denoted P or Q, as all of them refer to expressions in general. Yet, in the following, the notations P and Q will be often used to refer to expressions admitting the type $Prop$, also referred to as *formulas* or *propositions*.

These definitions of variables and terms represent the PVS syntax in the following way:

- $\mathcal{V}_{types}$ represents non-propositional base types.
- $\mathcal{V}_{expressions}$ represents constant expressions.
- $Type$ represents `TYPE`.
- $Prop$ represents `bool`. As presented in Chapter 2, classical reasoning and propositional extensionality are only considered as optional axioms in the presented idealization of predicate subtyping: in this setting, the notation `bool` is not convenient for the types of formulas in PVS-Core. We choose instead the usual alternative $Prop$, as formulas are also referred to as propositions.
- $\Pi x : A.B$ represents a dependent function type `[x : A -> B]`.
- $\{x : A \mid P\}$ represents a predicate subtype `{x : A | P}]`.
- $\forall x : A.P$ represents a universal quantification `FORALL (x : A) : P`.
- $P \Rightarrow Q$ represents an implication `P IMPLIES Q`.
- $\lambda x : A.t$ represents a λ-expressions `LAMBDA (x : A) : t`.
- tu represents an application `t(u)`.

In a second step, we define **declarations**, **contexts**, and **judgements**.

Definition 3.1.2. *We define successively declarations, contexts, and judgements as follows.*

- *We define three kinds of* ***declarations****:*
  $X : Type \mid x : A \mid P$
- *We define* ***contexts****, denoted* Γ*, as lists of declarations:*
  $\Gamma := \varnothing \mid \Gamma, X : Type \mid \Gamma, x : A \mid \Gamma, P$
- *We define four kinds of* ***judgements****:*
  $\Gamma \vdash WF \mid \Gamma \vdash A : Type \mid \Gamma \vdash t : A \mid \Gamma \vdash P$

These definitions correspond to the following notions in PVS and its idealized presentation in Chapter 2:

- A declaration $X : Type$ represents a PVS uninterpreted type declaration `X : TYPE`.
- A declaration $x : A$ represents a PVS uninterpreted constant declaration `x : A`.
- A declaration P represents a PVS formula declaration `h : AXIOM P`, but also `h : THEOREM P` or `h : OBLIGATION P`.
- A context represent both a PVS specifications, a typing context, and a proof context, as these two notions were considered equivalent in the idealized presentation of PVS.
- A judgement $\Gamma \vdash WF$ represents the well-formedness of a PVS specification.
- A judgement $\Gamma \vdash A : Type$ represents the well-formedness of a type relatively to a context.
- A judgement $\Gamma \vdash t : A$ represents typability of an expression by a type relatively to a context.
- A judgement $\Gamma \vdash P$ represents the provability of a formula relatively to a context.

The following definitions and notations will be useful in the analysis of PVS-Core terms and judgements.

Definition 3.1.3 (Binders, α-conversion)**.** *The terms of the form $\lambda x : A.t$, $\Pi x : A.B$, $\forall x : A.P$, and $\{x : A \mid P\}$ are binders, in which x is bound in t (resp. B, P) only. α-conversion is defined from this notion of binding, with the restriction that a bound variable $X \in \mathcal{V}_{types}$ (resp. $x \in \mathcal{V}_{expressions}$) can be substituted only by a variable $Y \in \mathcal{V}_{types}$ (resp. $y \in \mathcal{V}_{expressions}$). In the following, denote α-conversion by $=$ and we use it implicitly instead of syntactic equality.*

Definition 3.1.4 (Free variables, declared variables, substitution)**.** *The set of free variables of a term is defined as usual from the definition of binders, and we denote it $FV(\cdot)$. It is extended to contexts as follows.*

- $FV(\varnothing) = \varnothing$
- $FV(\Gamma, X : Type) = FV(\Gamma) \cup \{X\}$
- $FV(\Gamma, x : A) = FV(\Gamma) \cup \{x\} \cup FV(A)$
- $FV(\Gamma, P) = FV(\Gamma) \cup FV(P)$

The set of declared variables of a context $DV(\Gamma)$ is defined as follows.

- $DV(\varnothing) = \varnothing$

- $DV(\Gamma, X : Type) = DV(\Gamma) \cup \{X\}$
- $DV(\Gamma, x : A) = DV(\Gamma) \cup \{x\}$
- $DV(\Gamma, P) = DV(\Gamma)$

Substitution on terms and contexts is defined as usual. The substitution of v by N in M is denoted as $M[N/v]$.

Last, we define the following notations for β-reduction and β-equivalence in PVS-Core as follows.

Definition 3.1.5 (Reduction)**.** *We define the relation $\triangleright_\beta$ on terms as usual with $(\lambda x : A.t)u \triangleright_\beta t[u/x]$. Then, we define*

- *$\rightarrow_\beta$ the congruence closure of $\triangleright_\beta$*
- *$\twoheadrightarrow_\beta$ the reflexive transitive closure of $\rightarrow_\beta$*
- *$\equiv_\beta$ the symmetric closure of $\twoheadrightarrow_\beta$*

3.2 Rules and derivability

We define the following rules in PVS-Core, following the idealization of typing and deduction presented in Chapter 2. PVS-Core rules are organized as follows.

Well-formed contexts

- $\dfrac{}{\varnothing \vdash WF}$ Empty
- $\dfrac{\Gamma \vdash WF}{\Gamma, X : Type \vdash WF}$ TypeDecl $X \in \mathcal{V}_{types} \backslash DV(\Gamma)$
- $\dfrac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash WF}$ EltDecl $x \in \mathcal{V}_{expressions} \backslash DV(\Gamma)$
- $\dfrac{\Gamma \vdash P : Prop}{\Gamma, P \vdash WF}$ Assumption

Well-formed types

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ TypeVar $(X : Type) \in \Gamma$
- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash Prop : Type}$ Prop
- $\dfrac{\Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}$ Pi

- $\dfrac{\Gamma, x : A \vdash P : Prop}{\Gamma \vdash \{x : A \mid P\} : Type}$ SUBTYPE

Well-typed expressions

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ ELTVAR $(x : A) \in \Gamma$
- $\dfrac{\Gamma, x : A \vdash P : Prop}{\Gamma \vdash \forall x : A.P : Prop}$ FORALL
- $\dfrac{\Gamma, P \vdash Q : Prop}{\Gamma \vdash P \Rightarrow Q : Prop}$ IMPLY
- $\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}$ LAM
- $\dfrac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}$ APP
- $\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : Type}{\Gamma \vdash t : \{x : A \mid P\}}$ SUBTYPEINTRO
- $\dfrac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash t : A}$ SUBTYPEELIM1
- $\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash B : Type}{\Gamma \vdash t : B}$ TYPECONVERSION $A \equiv_\beta B$

Deductions

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash P}$ AXIOM $P \in \Gamma$
- $\dfrac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q}$ IMPLYINTRO
- $\dfrac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}$ IMPLYELIM
- $\dfrac{\Gamma, x : A \vdash P}{\Gamma \vdash \forall x : A.P}$ FORALLINTRO
- $\dfrac{\Gamma \vdash \forall x : A.P \qquad \Gamma \vdash t : A}{\Gamma \vdash P[t/x]}$ FORALLELIM
- $\dfrac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash P[t/x]}$ SUBTYPEELIM2
- $\dfrac{\Gamma \vdash P \qquad \Gamma \vdash Q : Prop}{\Gamma \vdash Q}$ PROPCONVERSION $P \equiv_\beta Q$

Each rule defines a family of **rule instances** in the following way.

Definition 3.2.1 (Rule instance)**.** *The description of each rule is done with a set of* ***parameters****: for instance, the parameters describing the rule* ForallElim *are* Γ*,* P*,* t*,* x*, and* A*.*

A set of parameters is well-suited for a rule if it satisfies all side conditions provided with this rule. A ***rule instance*** *is defined as the data of a rule together with a set of well-suited parameters.*

In turn, rule instances are used to define **inference steps** and **derivations** as follows.

Definition 3.2.2 (Derivation, inference step)**.** *A* ***derivation*** *is a tree labeled by judgements in which every junction, referred to as a an* ***inference step****, matches some rule instance.*

Following the relation between PVS-Core judgements and PVS, the derivability of a PVS-Core expresses the following properties in PVS.

- The derivability of a judgement $\Gamma \vdash WF$ asserts the well-formedness of a PVS specification represented by Γ.
- The derivability of a judgement $\Gamma \vdash A : Type$ asserts the well-formedness of a the PVS type represented by A relatively to the context represented by Γ.
- The derivability of a judgement $\Gamma \vdash t : A$ asserts the typability of the an expression represented by t by a type represented by A relatively to a context represented by Γ.
- The derivability of a judgement $\Gamma \vdash P$ asserts the provability of a formula represented by P relatively to a context represented by Γ.

Finally, we define the notions of well-formed context, well-typed expression, and inhabited type as follows.

Definition 3.2.3 (Well-formed context, well-typed expression, inhabited type)**.** *We define well-formed contexts, well-typed expression, and inhabited type as follows:*

- *A context* Γ *is well-formed if* $\Gamma \vdash WF$ *is derivable.*
- *An expression* t *is well-typed in a context* Γ *if* $\Gamma \vdash t : A$ *is derivable for some type* A*. An expression is well-typed if it is well-typed in some context* Γ*.*
- *A type* A *is inhabited in a context* Γ *if* $\Gamma \vdash t : A$ *is derivable for some expression* t*. A type is inhabited if it is inhabited in some context* Γ*.*

3.3 Summary of PVS-Core's main characteristics

We end this chapter with a brief overview of the main characteristics of PVS-Core. This section summarize the principles detailed and motivated in Chapter 2.

3.3.1 An extension of higher-order logic

As PVS, PVS-Core is an extension of higher-order logic. More precisely, a formalization of higher-order logic can be defined as the following restriction of PVS-Core.

Definition 3.3.1 (Underlying formalization of higher-order-logic)**.** *The underlying formalization of higher-order-logic in PVS-Core is defined by the following restrictions:*

- *A HOL term (resp. context, judgement) is a PVS-Core term (resp. context, judgement) in which no subterm has the form* $\{x : A \mid P\}$*.*
- *A HOL derivation is a PVS-Core derivation in which no rule* SUBTYPE*,* SUBTYPEINTRO*,* SUBTYPEELIM1*,* SUBTYPEELIM2*, nor* TYPECONVERSION *is used.*

In this fragment of PVS-Core derivable judgements, types belong to the grammar, $A, B := X \mid Prop \mid \Pi x : A.B$, which correspond to simple types. Indeed, for any type $\Pi x : A.B$ in this grammar, the expression variable x cannot occur free in B, hence this type construction corresponds to a non-dependent type, denoted `[A -> B]` in PVS.

3.3.2 Predicate subtyping and conversion

Starting from the previous formalization of higher-order logic as fragment of PVS-Core, the full system is obtained by adding, on the one hand, the predicate subtype construction $\{x : A \mid P\}$, and, on the other hand, the five rules SUBTYPE, SUBTYPEINTRO, SUBTYPEELIM1, SUBTYPEELIM2, and TYPECONVERSION. The four first rules are the essence of predicate subtyping:

- SUBTYPE is the formation rule of predicate subtypes.
- SUBTYPEINTRO describes how predicate subtypes can be inhabited.
- SUBTYPEELIM1 and SUBTYPEELIM2 shows how the inhabitants of predicate subtypes can be used.

On the other hand, the fifth rule TYPECONVERSION, is the counterpart, at the level of typing, of the deduction rule PROPCONVERSION. Together, these rules allow to consider types and expressions modulo β, and pave the way for the formalization of proofs modulo β, which is on of the main characteristics of the system PVS-Cert presented in the following chapter, and intended to be a system of *verifiable certificates* for PVS-Core.

Chapter 4

PVS-Cert: verifiable certificates for PVS-Core

As presented in the introduction (Chapter 1), the main purpose of this work is to define a language of verifiable certificates for predicate subtyping – more specifically, for the minimal formalization of predicate subtyping PVS-Core, presented in the previous chapter. The present chapter is dedicated to the presentation of such a system, which will be referred to as PVS-Cert.

What is meant as here as a language of *verifiable certificates* is a language of proofs equipped with a decision algorithm discriminating, for any formula P and any context Γ, the correct proofs of P in Γ. In the case of PVS-Cert, this question is a special case of the more general question of *type-checking*, solved in Chapter 7 with the definition of a sound, complete and terminating type-checking algorithm.

At first glance, there is no need to introduce any new system to define PVS-Core proofs formally: the language of PVS-Core derivations itself is a language of verifiable proofs for PVS-Core. However, this language is heavy as many parts of PVS-Core derivations contain unnecessary or redundant information. For instance, in the higher-order logic fragment of PVS-Core, only the deduction rules need to be recorded as typing is decidable in simply-typed λ-calculus. Moreover, defining a notion of cut elimination would be also heavy: as a cut would correspond to the stacking of several rules in a derivation, the definition of a single cut would involve distinct several judgements, based on possibly distinct contexts.

The main idea in the definition of PVS-Cert as a language of certificates for predicate subtyping is to formalize proofs as new kinds of terms, in addition to the types and expressions which are already present in PVS-Core. In PVS-Cert, a complete certificate is simply the typing judgement of some proof term with its corresponding theorem. As PVS-Cert is designed as a system in which type-checking is decidable, PVS-Cert derivations do not need to be recorded. Hence, this system of certificates is much lighter

than the language of PVS-Cert derivations as only one single judgement needs to be recorded. Moreover, it will be equipped (in Section 6) with a much lighter definition of cut elimination: in PVS-Cert, a cut is simply a computation rule for a proof terms, and its definition doesn't involve any judgement, nor any context.

We first present the formal definition of PVS-Cert, before providing more details on its underlying design choices.

4.1 Formal presentation

4.1.1 Terms and judgements

As detailed further in Section 4.2.1, the definition of PVS-Cert is comparable to the formalism of PTSs, presented for instance in [4]. The terms of PVS-Cert are defined as follows.

- **Sorts** $\mathcal{S} = \{Prop, Type, Kind\}$
  We use the notation s to refer to a sort.
- **Axioms** $\mathcal{A} = \{(Prop, Type), (Type, Kind)\}$
- **Rules** $\mathcal{R} = \{(Prop, Prop, Prop), (Type, Type, Type), (Type, Prop, Prop)\}$
- **Variables** The set of variables $\mathcal{V}$ is the disjoint union of three infinite countable sets of symbols $\mathcal{V}_{proofs}$, $\mathcal{V}_{expressions}$, and $\mathcal{V}_{types}$. The sets $\mathcal{V}_{expressions}$ and $\mathcal{V}_{types}$ refer to their respective definitions in PVS-Core, while the set $\mathcal{V}_{proofs}$ is new. We use the notation v to refer to a variable and $s(v)$ to refer to the unique sort such that $v \in \mathcal{V}_s$.
- **Terms** $\mathcal{T}$ is given by the following grammar:
  $M, N, T, U := s \mid v \mid \lambda v : T.M \mid MN \mid \Pi v : T.U \mid \{v : T \mid U\} \mid \langle M, N \rangle_T \mid \pi_1(M) \mid \pi_2(M)$

The contexts and judgements of PVS-Cert are defined as follows.

- **Contexts** $\Gamma := \varnothing \mid \Gamma, v : T$
- **Judgements** $\Gamma \vdash WF \mid \Gamma \vdash M : T$

Definition 4.1.1 (Binders, α-conversion)**.** *The terms $\lambda v : T.U$, $\Pi v : T.U$, and $\{v : T \mid U\}$ are binders, in which v is bound in U only. α-conversion is defined from this notion of binding, with the restriction that a bound variable v can be substituted only by a variable $v' \in \mathcal{V}_{s(v)}$. In the following, denote α-conversion by $=$ and we use it implicitly instead of equality.*

Definition 4.1.2 (Free variables, declared variables, substitution)**.** *The set of free variables of a term is defined as usual from the definition of binders, and we denote it*

$FV(\cdot)$. It is extended to contexts with $FV(v_1 : T_1, ..., v_n : T_n) = \{v_1, ..., v_n\} \cup FV(T_1) \cup ... \cup FV(T_n)$. The set of declared variables of a context is defined as $DV(v_1 : T_1, ..., v_n : T_n) = \{v_1, ..., v_n\} \subseteq FV(v_1 : T_1, ..., v_n : T_n)$. Substitution on terms and contexts is defined as usual. The substitution of v by N in M is denoted as $M[N/v]$.

Definition 4.1.3 (Reduction)**.** *We define the following relations.*

- $(\lambda v : T.M)N \triangleright_\beta M[N/v]$
- $\langle M_1, M_2 \rangle_T \triangleright_* M_1$
- $\pi_1(M) \triangleright_* M$

We define $\triangleright_{\beta*}$ as the union of $\triangleright_\beta$ and $\triangleright_*$. Given some relation $\triangleright__$ among $\triangleright_*$, $\triangleright_\beta$, and $\triangleright_{\beta*}$, we define

- $\rightarrow__$ *the congruence closure of* $\triangleright__$
- $\twoheadrightarrow__$ *the reflexive transitive closure of* $\rightarrow__$
- $\equiv__$ *the symmetric closure of* $\twoheadrightarrow__$

The specificity of these definitions is the reduction relation $\triangleright_*$, which can be interpreted as a the elimination of a coercion at the head of a term, and allows the expression of predicate subtyping in PVS-Cert. More detailed motivations and justifications for this definition are given in Section 4.2.2.

4.1.2 Typing rules

The typing rules of PVS-Cert are defined as follows:

- $\dfrac{}{\varnothing \vdash WF}$ Empty
- $\dfrac{\Gamma \vdash T : s}{\Gamma, v : T \vdash WF}$ Decl $v \in \mathcal{V}_s \backslash DV(\Gamma)$
- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash s_1 : s_2}$ Sort $(s_1, s_2) \in \mathcal{A}$
- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash v : T}$ Var $(v : T) \in \Gamma$
- $\dfrac{\Gamma \vdash T : s_1 \qquad \Gamma, v : T \vdash U : s_2}{\Gamma \vdash \Pi v : T.U : s_3}$ Prod $(s_1, s_2, s_3) \in \mathcal{R}$
- $\dfrac{\Gamma, v : T \vdash M : U \qquad \Gamma \vdash \Pi v : T.U : s}{\Gamma \vdash \lambda v : T.M : \Pi v : T.U}$ Lam
- $\dfrac{\Gamma \vdash M : \Pi v : T.U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U[N/v]}$ App
- $\dfrac{\Gamma \vdash T : Type \qquad \Gamma, v : T \vdash U : Prop}{\Gamma \vdash \{v : T \mid U\} : Type}$ Subtype

- $\dfrac{\Gamma \vdash M : T \qquad \Gamma \vdash N : U[M/v] \qquad \Gamma \vdash \{v : T \mid U\} : Type}{\Gamma \vdash \langle M, N \rangle_{\{v:T|U\}} : \{v : T \mid U\}}$ PAIR
- $\dfrac{\Gamma \vdash M : \{v : T \mid U\}}{\Gamma \vdash \pi_1(M) : T}$ PROJ1
- $\dfrac{\Gamma \vdash M : \{v : T \mid U\}}{\Gamma \vdash \pi_2(M) : U[\pi_1(M)/v]}$ PROJ2
- $\dfrac{\Gamma \vdash M : T \qquad \Gamma \vdash U : s}{\Gamma \vdash M : U}$ CONVERSION $T \equiv_{\beta *} U$

The specificity of this system lies in the use of the relation $\equiv_{\beta *}$ in the CONVERSION rule, which is at the core of the expression of predicate subtyping in PVS-Cert. More detailed motivations for such a definition are given in Section 4.2.2.

Like PVS-Core, PVS-Cert is equipped with formal definitions of **rule instance**, **inference step**, and **derivation**. These notions are formalized in the same way as done in PVS-Core (Definitions 3.2.1 and 3.2.2).

Finally, we define the notions of well-formed context, well-typed term, and inhabited term as follows.

Definition 4.1.4 (Well-formed context, well-typed term, inhabited term)**.** *We define well-formed contexts, well-typed terms, and inhabited terms as follows:*

- *A context* Γ *is well-formed if* $\Gamma \vdash WF$ *is derivable.*
- *A term* M *is well-typed in a context* Γ *if* $\Gamma \vdash M : T$ *is derivable for some term* T*. A term is well-typed if it is well-typed in some context* Γ*.*
- *A term* T *is inhabited in a context* Γ *if* $\Gamma \vdash M : T$ *is derivable for some term* M*. A term is inhabited if it is well-typed in some context* Γ*.*

4.2 The choice of PVS-Cert

4.2.1 An extension of λ-HOL

PVS-Cert is an extension of the PTS λ-HOL presented for instance in [4]. In order to highlight the relations between the two systems, we present the following definition of λ-HOL as a subsystem of PVS-Cert:

Definition 4.2.1 (λ-HOL)**.** *λ-HOL is defined from PVS-Cert defined by the following restrictions:*

- *A λ-HOL term (resp. context, judgement) is a PVS-Cert term (resp. context, judgement) in which no subterm has the form* $\{v : T \mid U\}$*,* $\pi_i(M)$*, or* $\langle M, N \rangle_T$*.*

- *A λ-HOL derivation is a PVS-Cert derivation in which no rule* SUBTYPE, PAIR, PROJ1, *or* PROJ2 *is used, and where conversion corresponds to* $\equiv_\beta$ *instead of* $\equiv_{\beta*}$.

The system λ-HOL is an instance of the family of Pure Type Systems (PTSs), also presented in [4]. PTSs, sometimes also referred to as Generalized Type Systems (GTSs), correspond to λ-HOL-like systems where the sets of sorts, axioms, and rules can be chosen freely. The notion of PTS is the result of several successive works, including notably [7, 74, 75, 76, 36, 3].

PTS-like systems are well-suited to describe reasoning modulo β: all steps of β-reduction or β-expansion in reasoning are kept implicit, which allows to keep proofs compact. Different logics can be expressed in different PTSs using variations of the Curry-Howard correspondence. The simplest PTS expressing higher-order logic is λ-HOL. Well-typed terms in λ-HOL are organized with a stratification, composed of separate classes of well-typed terms. One class corresponds to the simple types of higher-order logic, another corresponds to the terms of higher-order logic (propositions, etc.), and a third class corresponds to verifiable proofs of higher-order logic. In λ-HOL, a theorem becomes the type of its proofs, and such proofs can be verified using a type-checking algorithm.

PVS-Cert is designed to extend λ-HOL in the same way as PVS-Core extends higher-order logic. As in λ-HOL, well-typed terms are organized with a corresponding stratification, presented in Section 5.4, which includes a class of types, a class of expressions, and a class of proofs. This stratification allows to establish a simple correspondence with PVS-Core, described in Chapter 9.

4.2.2 The expression of predicate subtyping

The addition of proof terms is not the only difference between PVS-Core and PVS-Cert: in order to ensure the decidability of type-checking, PVS-Cert terms are inherently equipped with explicit coercions. These coercions contain explicitly the proofs which are required to change the types of terms. For instance, in a context Γ, using the notations of stratified terms which will be detailed in Definition 5.4.3: if $B = \{x : A \mid P\}$ is a type, t is an expression of type A, and p is a proof of type $P[t/x]$, then $\langle t, p\rangle_B$ is a coercion of t from A to B. These explicit coercions ensure the decidability of type-checking (proved in Chapter 7). Another difference between PVS-Core and PVS-Cert due to the presence of these coercions is the uniqueness of types in PVS-Cert, proved in Theorem 5.6.1.

The formalism used in PVS-Cert to define this system of coercions is very similar to the formalism of dependent pairs, used for instance in the type system ECC [53]. More precisely, the terms $\{v : T \mid U\}$ are comparable with types of dependent pairs (usually denoted $\Sigma v : T.U$), the terms $\langle M, N\rangle_T$ are comparable with dependent pairs, and the terms $\pi_i(M)$ are comparable with projections. The only difference between PVS-Cert

and the formalism of dependent pairs lies in the choice of conversion $\equiv_{\beta *}$: in the case of a system with dependent pairs, $\equiv_{\beta *}$ is replaced by the more standard conversion $\equiv_{\beta\sigma}$, defined as follows.

Definition 4.2.2. *The relation $\triangleright_\sigma$ is defined with $\pi_i\langle M_1, M_2\rangle_T \triangleright_\sigma M_i$, and the relation $\triangleright_{\beta\sigma}$ is defined as the union of $\triangleright_\beta$ and $\triangleright_\sigma$. As presented in Definition 4.1.3, for any relation $\triangleright__$ (for instance $\triangleright_\sigma$ or $\triangleright_{\beta\sigma}$), we consider*

- $\rightarrow__$ *the congruence closure of* $\triangleright__$
- $\twoheadrightarrow__$ *the reflexive transitive closure of* $\rightarrow__$
- $\equiv__$ *the symmetric closure of* $\twoheadrightarrow__$

Among all PTSs with dependent pairs, one particular system is even more closely related to PVS-Cert. It will be referred to as PVS-Cert$^-$, and defined as follows.

Definition 4.2.3 (PVS-Cert$^-$)**.** *The system PVS-Cert$^-$ is defined as PVS-Cert where the conversion $\equiv_{\beta *}$ is replaced by $\equiv_{\beta\sigma}$. It corresponds to a PTS extended with dependent pairs, where $\{v : T \mid U\}$ corresponds to the type of dependent pairs usually denoted $\Sigma v : T.U$. More precisely, PVS-Cert$^-$ corresponds to the PTS with dependent pairs obtained from λ-HOL by adding the single dependent pair rule $(Type, Prop, Type)$. It is included in the type system ECC presented in [53].*

Although this may not be directly visible from the definitions, PVS-Cert is in fact an extension PVS-Cert$^-$: as proved in Theorem 5.7.1, any judgement which is derivable in the PTS with dependent pairs PVS-Cert$^-$ is also derivable in PVS-Cert.

Using the more flexible conversion $\equiv_{\beta *}$ allows to establish a direct connection between PVS-Core and PVS-Cert. The link from PVS-Cert to PVS-Core (Chapter 9) is defined through the erasure of coercions, i.e. through normalization with respect to $\rightarrow_*$. Using the conversion $\equiv_{\beta *}$, two PVS-Cert types (resp. expressions) are convertible as long as the corresponding types (resp. expressions) in PVS-Core are also convertible, which allows to define a very simple translation from PVS-Core derivations to PVS-Cert derivable judgements (Definition 9.2.2 and Theorem 9.3.1).

The reduction $\twoheadrightarrow_{\beta *}$ does not preserve typing: if $\Gamma \vdash M : T$ is derivable and $M \twoheadrightarrow_{\beta *} N$, $\Gamma \vdash N : T$ is not necessarily derivable. For instance, the judgement $x : Prop, h : x \vdash \langle x, h\rangle_T : T$ with $T = \{y : Prop \mid y\}$ is derivable, but the judgement $x : Prop, h : x \vdash x : T$ is not. Instead, the reduction $\twoheadrightarrow_{\beta\sigma}$ is type preserving, and will be used both in the type-checking algorithm (Chapter 7) and as a definition of cut elimination for PVS-Cert proofs (Chapter 6).

Chapter 5

Properties of PVS-Cert

This chapter is dedicated to the presentation of important properties of PVS-Cert. These properties, together with the strong normalization (and cut elimination) theorem presented in Chapter 6, will be used to define a type-checking algorithm (Chapter 7) for PVS-Cert, to prove that it is a conservative extension of higher-order logic (Chapter 8), and to formalize the correspondence between PVS-Core and PVS-Cert (Chapter 9). The two most important of these properties are the stratification theorem, which allows to discriminate types, expressions, and proofs among well-typed terms, and the type preservation theorem, which allows to define a sound notion of computation and cut elimination – whose termination will be proved in the strong normalization theorem in Chapter 6.

We begin this section with the definition of tools for the analysis of PVS-Cert derivations.

5.1 Tools for the analysis of derivations

We present three theorems easing the analysis of PVS-Cert derivations, which will be referred to as the subderivations theorem, the free variable theorem, and the renaming theorem respectively. The two first theorems are established directly, while the last one requires several lemmas. These definitions correspond to standard properties expected from most types systems. In particular, similar tools could be defined for arbitrary PTSs, arbitrary PTSs extended with dependent pairs, but also for PVS-Core.

Theorem 5.1.1 (Subderivations)**.** *For any derivation of a judgement* $\Gamma \vdash WF$ *or* $\Gamma \vdash M : U$*, the following properties hold:*

- *For any prefix* Γ' *of* Γ*, there exists a subderivation of* $\Gamma' \vdash WF$*.*
- *For any prefix* $(\Gamma', v : T)$ *of* Γ*, there exists a subderivation of* $\Gamma' \vdash T : s(v)$*.*
- *In the case of a judgement of the form* $\Gamma \vdash M : U$*, there exists a subderivation*

of conclusion of the form $\Gamma \vdash M : T$ *where* $T \equiv_{\beta *} U$ *and the last inference step matches some instance of the rule determined by* M *(*App *for an application, etc.).*

Proof. The proof is done straightforwardly by induction on the derivation. □

Theorem 5.1.2 (Free variables)**.** *If* $\Gamma \vdash M : U$ *or* $\Gamma \vdash WF$ *is derivable, with* $\Gamma = v_1 : T_1, ..., v_n : T_n$*, the following statements hold.*

- *In the first case,* $FV(M) \cup FV(N) \subseteq \{v_1, ..., v_n\}$.
- *For all* i $FV(T_i) \subseteq \{v_1, ..., v_{i-1}\}$.
- *The* v_i *are pairwise distinct.*

Proof. The two first statements are proved together by induction on the derivation. The third is proved alone by induction on the derivation. □

The last useful tool for the analysis of derivations is the renaming theorem. It provides some flexibility in the choice of the parameter v occurring in several rule instances, when matching the last inference step of some derivation. In some cases as Proj1, the same inference step can be matched with different choices of v. In other cases as Lam, we need to change the derivation to match different choices of v, but this can be done in a height-preserving way. The theorem is the following:

Theorem 5.1.3 (Renaming)**.** *For any derivation of height* n *and conclusion* $\Gamma \vdash M : N$ *where the last inference step matches some instance of a rule* R*, the following holds:*

1. *If* R *is among* App, Pair, Proj1, *and* Proj2 *and* v *is its variable parameter, for any* $v \in \mathcal{V}_{s(v)} \backslash DV(\Gamma)$*, the last inference step matches some instance of* R *of variable parameter* v*.*
2. *If* R *is among* Prod, Lam, *and* Subtype *and* v *is its variable parameter, for any* $v \in \mathcal{V}_{s(v)} \backslash DV(\Gamma)$*, there exists a derivation of height* n *and conclusion* $\Gamma \vdash M : N$ *such that the last inference step matches some instance of* R *of variable parameter* v*.*

Although this theorem is a natural and basic result, its proof involves several distinct steps, which will be presented in the rest of this section. As mentioned in the beginning of this section, the proof this result is not specific to PVS-Cert: similar proofs could be done for PTSs or PTSs extended with dependent pairs.

We will use the following properties of substitution.

Lemma 5.1.1. *The following properties hold.*

- *For all terms* T*,* M*, and* N*, for any variables* v *and* $v' \in \mathcal{V}_{s(v)} \backslash FV(M)$*,* $M[v'/v][N/v'] = M[N/v]$.

- *For all terms M, N, and N', any variables $v \notin FV(N')$ and $v' \neq v$, $M[N/v][N'/v'] = M[N'/v'][N[N'/v']/v]$.*

Proof. The first statement is proved by induction on M. Substitution being stable by α-conversion, we can suppose without loss of generality that the bound variables in M are neither equal to v or v' nor free in N, which makes the proof straightforward.

The second statement is proved by induction on M. Substitution being stable by α-conversion, we can suppose without loss of generality that the bound variables in M are neither equal to v or v', nor free in N, N', or $N[N'/v']$, which makes the proof straightforward as well. □

The next step is the proof of the first part of the renaming theorem.

Lemma 5.1.2. *The first statement of the renaming theorem holds: for any derivation of conclusion $\Gamma \vdash M : N$, if the last inference step matches some instance of a rule R among* App, Pair, Proj1, *and* Proj2 *and if v refers to its variable parameters, the following holds: for any $v' \in \mathcal{V}_{s(v)} \backslash DV(\Gamma)$, the last inference step matches some instance of R of variable parameter v'.*

Proof. All cases are similar. We present the example of an instance of Pair:

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash N : U[M/v] \qquad \Gamma \vdash \{v : T \mid U\} : Type}{\Gamma \vdash \langle M, N \rangle_{\{v:T|U\}} : \{v : T \mid U\}} \textsc{Pair}$$

We consider $v' \in \mathcal{V}_{s(v)} \backslash DV(\Gamma)$. To prove the expected result, we prove that the instance of Pair defined by the parameters Γ, M, N, v', T and $U[v'/v]$ matches the same inference step. For this, we need to prove $\{v : T \mid U\} = \{v' : T \mid U[v'/v]\}$ and $U[M/v] = U[v'/v][M/v']$, which both hold, by definition of α-conversion and by Lemma 5.1.1 respectively, as long as $v' \in \mathcal{V}_{s(v)} \backslash FV(U)$. By the free variable theorem, this is the case. □

The following statement will be used to establish the second part of the renaming theorem.

Lemma 5.1.3. *Given*

- *a derivation of some judgement of the form $\Gamma, v : T, \Delta \vdash M : U$ (resp. $\Gamma, v : T, \Delta \vdash WF$),*
- *a variable $w \in \mathcal{V}_{s(v)} \backslash DV(\Gamma, \Delta)$,*

the judgement $\Gamma, w : T, \Delta[w/v] \vdash M[w/v] : U[w/v]$ (resp. $\Gamma, w : T, \Delta[w/v] \vdash WF$) admits a derivation of the same length as the previous one

Proof. This property is proved by induction on the height of derivations.

- The case Empty cannot occur.

- The case DECL splits into two subcases, depending on whether Δ is empty or not. Both cases are established straightforwardly using the fact that $w \in \mathcal{V}_{s(v)} \backslash DV(\Gamma, \Delta)$. The induction hypothesis is used in the second subcase.

- The case VAR splits into three subcases. With the notation $\Gamma, v : T, \Delta \vdash v' : T'$ for the conclusion, using the free variable theorem, $(v' : T)$ appears exactly once in the context, either in Γ, in $\{(v : T)\}$, or in Δ. In every case we conclude the expected result from the induction hypothesis, using the free variable theorem to ensure that $(v'[w/v] : T'[w/v]) = (v' : T')$ in the first case and that $T[w/v] = T$ in the second case.

- The case CONVERSION is done by induction hypothesis, using the stability of $\equiv_{\beta *}$ by substitution.

- The rules APP, PAIR, PROJ1, and PROJ2 are similar, and use Lemma 5.1.2. We present the case PAIR as an example. By Lemma 5.1.2, the last inference step matches some rule instance

$$\frac{\Gamma' \vdash M' : T' \qquad \Gamma' \vdash N' : U'[M'/v'] \qquad \Gamma' \vdash \{v' : T' \mid U'\} : Type}{\Gamma' \vdash \langle M', N' \rangle_{\{v':T' \mid U'\}} : \{v' : T' \mid U'\}} \text{ PAIR}$$

  in which $\Gamma' = \Gamma, v : T, \Delta$ and $v' \notin DV(\Gamma') \cup \{w\}$.
  First, we apply the induction hypothesis on premises. In order to apply a PAIR instance on the new judgements, we have to check two conditions: $\{v' : T' \mid U'\}[w/v] = \{v' : T'[w/v] \mid U'[w/v]\}$ and $U'[M'/v'][w/v] = U'[w/v][M'[w/v]/v']$. By definition of substitution and by Lemma 5.1.1 respectively, both are ensured as long as the two inequalities $v' \neq v$ and $v' \neq w$ hold, which is the case given the requirement on v'.

  Last, we check that the conclusion $\langle M'[w/v], N'[w/v] \rangle_{\{v':T'[w/v] \mid U'[w/v]\}} : \{v' : T'[w/v] \mid U'[w/v]\}$ is the expected one, which straightforward using $\{v' : T' \mid U'\}[w/v] = \{v' : T'[w/v] \mid U'[w/v]\}$.

- The rules PROD, LAM, and SUBTYPE are similar. We present the case LAM as an example. The last inference step matches some instance

$$\frac{\Gamma', v' : T' \vdash M' : U' \qquad \Gamma' \vdash \Pi v' : T'.U' : s'}{\Gamma' \vdash \lambda v' : T'.M' : \Pi v' : T'.U'} \text{ LAM}$$

  in which $\Gamma' = \Gamma, v : T, \Delta$. First, we consider some variable w' belonging to $\mathcal{V}_{s(v')} \backslash (DV(\Gamma') \cup \{v', w\})$. We first apply the induction hypothesis on the first premise to get a derivation of $\Gamma, w' : T' \vdash M'[w'/v'] : U'[w'/v']$. Then, as the length of derivations is preserved and $w' \neq w$, we apply the induction hypothesis again to get a derivation of $\Gamma, w : T, \Delta[w/v], w' : T'[w/v] \vdash M'[w'/v'][w/v] : U'[w'/v'][w/v]$.

On the other hand, by the free variable theorem, $w' \notin FV(U')$, hence the second premise is equal to $\Gamma' \vdash \Pi w' : T'.U'[w'/v'] : s'$. Applying the induction hypothesis, we get a derivation of $\Gamma, w : T, \Delta[w/v] \vdash (\Pi w' : T'.U'[w'/v'])[w/v] : s'$. By hypothesis, $w' \neq w$ and $w' \neq v$, hence this judgement is equal to $\Gamma, w : T, \Delta[w/v] \vdash \Pi w' : T'[w/v].U'[w'/v'][w/v] : s'$.

Applying the Lam rule, we obtain a derivation of $\Gamma, w : T, \Delta[w/v] \vdash \lambda w' : T'[w/v].M'[w'/v'][w/v] : \Pi w' : T'[w/v].U'[w'/v'][w/v]$. We need to prove that this conclusion is the expected one, $(\lambda v' : T'.M')[w/v] = \lambda w' : T'[w/v].M'[w'/v'][w/v]$ and similarly with $\Pi v' : T'.U'$. The second equality was proved in the previous paragraph. The first one is proved in the same way, using the fact that $w' \notin FV(M')$, as ensured by the free variable theorem.

□

We conclude the proof of the renaming theorem as follows.

Proof. [Renaming] The first part of the theorem is Lemma 5.1.2. In the second part, the three cases are similar. We present the example of an instance of Lam. In this case, the last inference step matches some instance

$$\frac{\Gamma, v : T \vdash M : U \qquad \Gamma \vdash \Pi v : T.U : s}{\Gamma \vdash \lambda v : T.M : \Pi v : T.U}\ \textsc{Lam}$$

We consider $w \in \mathcal{V}_{s(v)} \backslash DV(\Gamma)$. By Lemma 5.1.3, there exists a derivation of $\Gamma, w : T \vdash M[w/v] : U[w/v]$ of same height as the first subderivation. Moreover, by the free variable theorem, $w \in \mathcal{V}_{s(v)} \backslash FV(U)$, thus the second premise is equal to $\Gamma \vdash \Pi w : T.U[w/v] : s$. We combine both derivations to derive $\Gamma \vdash \lambda w : T.M[w/v] : \Pi w : T.U[w/v]$ using the rule instance

$$\frac{\Gamma, w : T \vdash M[w/v] : U[w/v] \qquad \Gamma \vdash \Pi w : T.U[w/v] : s}{\Gamma \vdash \lambda w : T.M[w/v] : \Pi w : T.U[w/v]}\ \textsc{Lam}$$

By the free variable theorem, $w \in \mathcal{V}_{s(v)} \backslash FV(M)$ and $w \in \mathcal{V}_{s(v)} \backslash FV(U)$, hence this conclusion is α-convertible to the original one. □

5.2 Thinning and substitution

The theorems presented here express the admissibility of two usual rules in deduction systems: thinning and substitution. These two operations on derivable judgements will be at the core of a more advanced analysis, in particular the type preservation of the reduction $\twoheadrightarrow_{\beta\sigma}$ presented in Section 5.5.

These properties are expected for any PTS, or even for any PTS extended with dependent pairs, such as the system PVS-Cert$^-$ presented in Definition 4.2.3. The

choice of conversion doesn't play a strong role in these proofs, which correspond to what could be done for PTSs as well as PTSs extended with dependent pairs.

Theorem 5.2.1 (Thinning)**.** *If* $\Gamma \vdash M : N$ *and* $\Delta \vdash WF$ *are derivable with* $\Gamma \subseteq \Delta$*, then* $\Delta \vdash M : N$ *is derivable.*

Proof. The proof is done by induction on the height of the derivation. The only difficult cases are Prod, Lam and Subtype, in which some premise admits an extended context. We present here the example of Lam, the two other cases are established similarly using the renaming theorem. Discarding the notations of the original statement and using the renaming theorem, we can suppose that the last inference rule matches some instance

$$\frac{\Gamma, v : T \vdash M : U \qquad \Gamma \vdash \Pi v : T.U : s}{\Gamma \vdash \lambda v : T.M : \Pi v : T.U}\ \textsc{Lam}$$

with $v \notin DV(\Delta)$, as the original derivation can be transformed into a derivation of same height and conclusion matching this requirement. By the subderivations theorem, the first premise contains a subderivation of $\Gamma \vdash T : s(v)$. By strong induction hypothesis, $\Delta \vdash T : s(v)$ is derivable, and, as $v \notin DV(\Delta)$, $\Delta, v : T \vdash WF$ is derivable too. Thus, we can use the induction hypothesis on the first premise to obtain a derivation of $\Delta, v : T \vdash M : U$. On the other hand, we apply the induction hypothesis to the second premise. Then, applying Lam, we obtain a derivation of $\Delta \vdash \Pi v : T.M : \Pi v : T.U$. □

The thinning theorem is the main lemma in the proof of the following substitution theorem.

Theorem 5.2.2 (Substitution)**.** *If*

- $\Gamma, v : T, \Delta \vdash M : U$ *(resp.* $\Gamma, v : T, \Delta \vdash WF$*) is derivable,*
- $\Gamma \vdash N : T$ *is derivable,*

then $\Gamma, \Delta[N/v] \vdash M[N/v] : U[N/v]$ *(resp.* $\Gamma, \Delta[N/v] \vdash WF$*) is derivable too.*

Proof. The proof is similar although simpler than the proof of Lemma 5.1.3. It is done by induction on the height of the first derivation:

- The case Empty cannot occur.
- The case Decl splits into two subcases, depending on whether Δ is empty or not. Both cases are established straightforwardly, using the subderivations theorem in the first subcase and the induction hypothesis in the second subcase.
- The case Sort is straightforward by induction hypothesis.
- The case Var splits into three subcases. With the notation $v' : T'$ for the conclusion, using the free variable theorem, $(v' : T')$ appears exactly once in the context, either in Γ, in $\{(v : T)\}$, or in Δ. In the first case we conclude by induction hypothesis, using the free variable theorem to ensure that $(v'[N/v] : T'[N/v]) = (v' : T')$. In

the second case we use the induction hypothesis to derive $\Gamma, v : T, \Delta[N/v] \vdash WF$, and the thinning theorem with this derivation together with the derivation of $\Gamma \vdash N : T$ to derive $\Gamma, v : T, \Delta[N/v] \vdash N : T$. This conclusion is the expect one as, by the free variable theorem, $(v[N/v] : T[N/v]) = (N : T)$. In the third case, we conclude directly by induction hypothesis.

- The case Conversion is done by induction hypothesis, using the stability of $\equiv_{\beta *}$ by substitution.

- The cases of rules App, Pair, Proj1, and Proj2 are similar. We present the case Pair as an example. By the renaming theorem, the last inference step matches some instance of Pair

$$\frac{\Gamma' \vdash M' : T' \qquad \Gamma' \vdash N' : U'[M'/v'] \qquad \Gamma' \vdash \{v' : T' \mid U'\} : Type}{\Gamma' \vdash \langle M', N' \rangle_{\{v':T'|U'\}} : \{v' : T' \mid U'\}} \text{ Pair}$$

in which $\Gamma' = \Gamma, v : T, \Delta$ and $v' \notin DV(\Gamma')$. First, we apply the induction hypothesis on premises. In order to apply a Pair instance on the new judgements, we have to check two conditions: $\{v' : T' \mid U'\}[N/v] = \{v' : T'[N/v] \mid U'[N/v]\}$ and $U'[M'/v'][N/v] = U'[N/v][M'[N/v]/v']$. By definition of substitution and by Lemma 5.1.1 respectively, both are ensured as long as $v' \neq v$ and $v' \notin FV(N)$ which are provable from the requirement on v' and free variable theorem. Last, we have to check that the conclusion $\langle M'[w/v], N'[w/v] \rangle_{\{v':T'[w/v]|U'[w/v]\}} : \{v' : T'[w/v] \mid U'[w/v]\}$ is the expected one, which straightforward using the equality $\{v' : T' \mid U'\}[N/v] = \{v' : T'[N/v] \mid U'[N/v]\}$.

- The rules Prod, Lam, and Subtype are similar. We present the case Lam as an example. The last inference step matches some instance of Lam

$$\frac{\Gamma', v' : T' \vdash M' : U' \qquad \Gamma' \vdash \Pi v' : T'.U' : s'}{\Gamma' \vdash \lambda v' : T'.M' : \Pi v' : T'.U'} \text{ Lam}$$

in which $\Gamma' = \Gamma, v : T, \Delta$.
First, we apply the induction hypothesis on premises. In order to apply a Lam instance on the new judgements, we have to check that $(\lambda v' : T'.M')[N/v] = \lambda v' : T'[N/v].M'[N/v]$ and $(\Pi v' : T'.U')[N/v] = \Pi v' : T'[N/v].U'[N/v]$. Both are ensured by $v' \neq v$ and $v' \notin FV(N)$, which are provable from the requirement on v' and free variable theorem. Last, we have to check that the conclusion is the expected one, which is straightforward using $(\lambda v' : T'.M')[N/v] = \lambda v' : T'[N/v].M'[N/v]$ and $(\Pi v' : T'.U')[N/v] = \Pi v' : T'[N/v].U'[N/v]$ once again.

□

We end this section with the following corollary of the substitution theorem.

Theorem 5.2.3. *If $\Gamma \vdash M : T$ is derivable and $T \neq Kind$, there exists a sort s such that $\Gamma \vdash T : s$.*

Proof. The proof is done by induction on the derivation:

- The cases EMPTY and DECL cannot occur.
- The cases SORT, PROD, LAM, SUBTYPE, PAIR, and CONVERSION are straightforward.
- In the case VAR, we replace the notation $\Gamma \vdash M : T$ of the original statement by $\Gamma, v : T, \Delta \vdash v : T$. The premise is $\Gamma, v : T, \Delta \vdash WF$, and, from the subderivations theorem, $\Gamma \vdash T : s(v)$ is also derivable. Hence, by the thinning theorem, $\Gamma, v : T, \Delta \vdash T : s(v)$ is derivable.
- The case APP is proved in the following way. Discarding the notations of the original statement and using the renaming theorem, the last inference step matches some instance of APP

$$\frac{\Gamma \vdash M_1 : \Pi v : T.U \qquad \Gamma \vdash M_2 : T}{\Gamma \vdash M_1 M_2 : U[M_2/v]}\ \textsc{App}$$

  where $v \notin FV(\Gamma)$. We know by induction hypothesis that $\Gamma \vdash \Pi v : T.U : s$ is derivable for some sort s. Hence, by the subderivations theorem followed by the renaming theorem, $\Gamma, v : T \vdash U : s'$ is derivable for some sort s'. From the substitution theorem, we conclude that $\Gamma \vdash U[M_2/v] : s'$ is derivable.
- The case PROJ1 is proved in the following way. Discarding the notations of the original statement and using the renaming theorem, the last inference step matches some instance of PROJ1

$$\frac{\Gamma \vdash M : \{v : T \mid U\}}{\Gamma \vdash \pi_1(M) : T}\ \textsc{Proj1}$$

  where $v \notin FV(\Gamma)$. We know by induction hypothesis that $\Gamma \vdash \{v : T \mid U\} : s$ is derivable for some s. Hence, using the subderivations theorem followed by the renaming theorem, $\Gamma, v : T \vdash U : \mathit{Prop}$ is derivable. From the subderivations theorem, we conclude that $\Gamma \vdash T : s(v)$ is derivable .
- The case PROJ2 is proved in the following way. Discarding the notations of the original statement and using the renaming theorem, the last inference step matches some instance of PROJ2

$$\frac{\Gamma \vdash M : \{v : T \mid U\}}{\Gamma \vdash \pi_2(M) : U[\pi_1(M)/v]}\ \textsc{Proj2}$$

  where $v \notin FV(\Gamma)$. We know by induction hypothesis that $\Gamma \vdash \{v : T \mid U\} : s$ is derivable for some s. Hence, using the subderivations theorem followed by the renaming theorem, $\Gamma, v : T \vdash U : \mathit{Prop}$ is derivable. On the other hand, we derive

directly from the premise $\Gamma \vdash \pi_1(M) : T$ using PROJ1. Hence, by the substitution theorem, we conclude that $\Gamma \vdash \pi_2(M) : U[\pi_1(M)/v]$ is derivable.

□

5.3 The Church-Rosser property

The standard properties presented in the previous sections do not depend very strongly on the choice of conversion. In the following, the conversion $\equiv_{\beta *}$ and the reduction $\twoheadrightarrow_{\beta *}$ will play a more important role. In particular, we will prove and use the Church-Rosser property for $\rightarrow_{\beta *}$. In the case of a PTS, an important consequence of the Church-Rosser property of $\rightarrow_{\beta}$ is the type preservation of $\twoheadrightarrow_{\beta}$. In the case of PVS-Cert, the first important consequence of the Church-Rosser property of $\rightarrow_{\beta *}$ is the stratification theorem. In a second step, the Church-Rosser property of $\rightarrow_{\beta *}$ will be used again together with the stratification theorem to establish the type preservation of an alternative reduction, $\twoheadrightarrow_{\beta\sigma}$.

Theorem 5.3.1 (Church-Rosser for $\rightarrow_{\beta *}$)**.** *Whenever $M_1 \equiv_{\beta *} M_2$, there exists N such that $M_1 \twoheadrightarrow_{\beta *} N$ and $M_2 \twoheadrightarrow_{\beta *} N$.*

Proof. $\mathcal{T}$ equipped with $\rightarrow_{\beta *}$ is an orthogonal combinatory reduction system (as defined in [48]), as rules are left-linear and non-overlapping. As proved in [48], such a system admits the Church-Rosser property. □

The following corollary of the Church-Rosser theorem will be useful in the proof of several results, such as the type preservation of the reduction $\twoheadrightarrow_{\beta\sigma}$ presented in Section 5.5.

Theorem 5.3.2. *For all terms $M_1 \equiv_{\beta *} M_2$ such that for all $i \in \{1, 2\}$, M_i has one of the forms s, v, $\Pi v : T.U$, or $\{v : T \mid M\}$, one of the following holds.*

- *There exists a sort s such that $M_1 = M_2 = s$.*
- *There exists a variable v such that $M_1 = M_2 = v$.*
- *M_1 has the form $\Pi v : T_1.U_1$ and M_2 has the form $\Pi v : T_2.U_2$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$.*
- *M_1 has the form $\{v : T_1 \mid U_1\}$ and M_2 has the form $\{v : T_2 \mid U_2\}$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$.*

Proof. We first prove that for any terms M_1 and M_2 such that $M_1 \twoheadrightarrow_{\beta *} M_2$ and M_1, the following statements hold:

- If M_1 is a sort, $M_1 = M_2$.
- If M_1 is a variable, $M_1 = M_2$.

- If M_1 has the form $\Pi v : T_1.U_1$, then M_2 has the form $\Pi v : T_2.U_2$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$.
- If M_1 has the form $\{v : T_1 \mid U_1\}$, then M_2 has the form $\{v : T_2 \mid U_2\}$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$.

The proof is straightforward by induction on the length of the reduction $M_1 \twoheadrightarrow_{\beta *} M_2$. In a second step, the expected result is proved as follows. If two terms $M_1 \equiv_{\beta *} M_2$ are such that for all $i \in \{1, 2\}$, M_i has one of the forms s, v, $\Pi v : T.U$, or $\{v : T \mid M\}$, by the Church-Rosser theorem, there exists a term N such that $M_1 \twoheadrightarrow_{\beta *} N$ and $M_2 \twoheadrightarrow_{\beta *} N$. We conclude the expected result by splitting the 16 different cases for (M_1, M_2). All of them are straightforward. □

5.4 Stratification in PVS-Cert

The stratification of terms in PVS-Cert reveals a strong link between PVS-Cert and PVS-Core (defined and used in Chapter 9), in the same way as the stratification of terms in λ-HOL reveals its link with higher-order logic. The property of stratification holds for several other systems, such as the injective GTSs presented in [36] – in this paper, this result is referred to as classification.

The main lemma used to establish such a result is the fact that, whenever the rule of conversion is used some derivation, the two convertible terms belong to the same class of terms. The simplest way to prove this result is to choose classes of terms that are stable under reduction and to conclude using the Church-Rosser theorem. In the case of injective GTSs, these classes only contain well-typed terms, and the stability under reduction follows from the subject reduction property.

However, as mentioned in Section 4.2.2, type preservation does not hold for $\twoheadrightarrow_{\beta *}$ in PVS-Cert. For this reason, we will choose a relaxed definition of stratified terms, where the different classes are not restricted to well-typed terms. Using this relaxed definition, it will be possible to prove, in the absence of type preservation for $\twoheadrightarrow_{\beta *}$, that most classes of stratified terms are stable by reduction with $\twoheadrightarrow_{\beta *}$.

We will first present three classes of terms defined inductively: the classes of types, expressions, and proofs. The expected property of stability by reduction will only be proved for types and expressions, which is not problematic as the conversion $\equiv_{\beta *}$ is never directly applied to a proof in any derivation.

These classes of terms will play a significant role in the proof of type preservation of the reduction $\twoheadrightarrow_{\beta\sigma}$ presented in Section 5.5, in the strong normalization theorem presented in Chapter 6, as well as in the construction of the type-checking algorithm presented in Chapter 7. These classes will be presented with the following notations for variables.

Definition 5.4.1 (Variables classification)**.** *We use the following notations:*

- X, Y, Z *for variables in* $\mathcal{V}_{types}$
- x, y, z *for variables in* $\mathcal{V}_{expressions}$
- h *for variables in* $\mathcal{V}_{proofs}$

Stratified terms are defined as follows.

Definition 5.4.2 (Stratified terms)**.** *We define stratified terms among terms as follows.*

- ***Types*** $A, B := X \mid Prop \mid \Pi x : A.B \mid \{x : A \mid P\}$
- ***Expressions*** $t, u, P, Q := x \mid \Pi x : A.P \mid \Pi h : P.Q \mid \lambda x : A.t \mid t\ u \mid \langle t, M\rangle_A \mid \pi_1(t)$
- ***Proofs*** $p, q := h \mid \lambda h : P.p \mid \lambda x : A.p \mid p\ q \mid p\ t \mid \pi_2(t)$

In the same way as mentioned in the case of PVS-Core (Definition 3.1.1), there is no formal distinction between the notations t, u, P, and Q, although we will prefer, in the following, the notations of expressions P, Q for expressions of type $Prop$, and the notations t, u in other cases.

The most important remark on the definition of stratified terms is the fact that any pair $\langle t, M\rangle_A$ (where t is an expression and A is a type) is accepted as a correct expression: the term M used in it can be arbitrary, and in particular it is not required to be a proof. This choice is due to the fact that proofs are not stable by $\rightarrow_{\beta *}$: for instance, $(\lambda h : x.h)y$ is a proof, but y is not. Hence, compared to the alternative of restricting pairs to terms of the form $\langle t, p\rangle_A$, the present relaxed definition is necessary to ensure the stability of types and expressions under $\rightarrow_{\beta *}$, proved in Proposition 5.4.2 and used as an important step towards the proof of the stratification theorem 5.4.1.

It would be possible, although more cumbersome, to prove another version of the stratification theorem 5.4.1 with this more restrictive alternative definition. However, beyond the proof of the stratification theorem itself, the stability of types and expressions under $\rightarrow_{\beta *}$ will be useful in the following of the study of PVS-Cert. In particular, it eases the proof of the strong normalization theorem presented in Chapter 6. This proof is based on the interpretations of expressions and types (as well as $Type$ and $Kind$) as sets of terms, presented in Definition 6.4.4. The fact that this interpretation is stable under $\equiv_{\beta *}$, proved in Lemma 6.4.2, is necessary to prove the strong normalization theorem, and the stability of expressions and types under $\rightarrow_{\beta *}$ plays a key role in this lemma.

We complete the definition of stratified terms with a definition of stratified contexts and stratified judgements.

Definition 5.4.3 (Stratified contexts, stratified judgements)**.** *We define successively stratified contexts and stratified judgements as follows.*

- *A **stratified context** is a context in which all declarations have the form $X : Type$, $x : A$ (for some type A), or $h : P$ (for some expression P).*
- *A **stratified judgement** is a judgement of one of the following form, in which Γ is a stratified context:*
  - $\Gamma \vdash WF$
  - $\Gamma \vdash Type : Kind$
  - $\Gamma \vdash A : Type$
  - $\Gamma \vdash t : A$
  - $\Gamma \vdash p : P$

Before showing that terms and expressions are stable under $\rightarrow_{\beta *}$, we first show that they are stable under some specific substitutions.

Proposition 5.4.1. *For any term M, the following hold.*

- *For any expression t and any expression variable x, if M is a type (resp. an expression), so is $M[t/x]$.*
- *For any term N and any proof variable h, if M is a type (resp. an expression), so is $M[N/h]$.*

Proof. The proofs are straightforward by induction on types and expressions. □

We establish the stability of terms and expressions under $\rightarrow_{\beta *}$ as follows.

Proposition 5.4.2. *For any term M, the following hold.*

- *If $M \rightarrow_{\beta *} N$ and M is a type (resp. an expression), so is N.*
- *If $M \twoheadrightarrow_{\beta *} N$ and M is a type (resp. an expression), so is N.*

Proof. The proof of the first statement is done by induction on M (as a type or an expression), using Proposition 5.4.1. We conclude the second one by induction on the length of the reduction. □

We conclude the following proposition, which is the main lemma of the stratification theorem.

Proposition 5.4.3. *If $M_1 \equiv_{\beta *} M_2$ and, for all $i \in \{1, 2\}$, M_i is either a type, an expression, $Type$, or $Kind$, then the M_i are either both $Kind$, both $Type$, both types or both expressions.*

Proof. We prove that all other situations lead to a contradiction. The most difficult case appears when one term is a type and the other an expression. In this case, by the Church-Rosser theorem, there exists a term M such that $M_1 \twoheadrightarrow_{\beta *} M$ and $M_2 \twoheadrightarrow_{\beta *} M$. Using Proposition 5.4.2, M is both a type and a expression. We prove that such a situation is impossible by straightforward induction on terms.

All other cases are straightforward using the Church-Rosser theorem. □

Using this result, we prove the stratification theorem as follows.

Theorem 5.4.1 (Stratification)**.** *Any derivable judgement is a stratified judgement: if some judgement* $\Gamma \vdash M : T$ *or* $\Gamma \vdash WF$ *is derivable with* $\Gamma = v_1 : T_1, ..., v_n : T_n$*, the following statements hold.*

- *In the first case,* $M : T$ *has one of the following forms:* $p : P$*,* $t : A$*,* $A : Type$*, or* $Type : Kind$*.*
- *For all* i*,* $v_i : T_i$ *has one of the following forms:* $h : P$*,* $x : A$*, or* $X : Type$

Proof. The proof is done by induction the derivation.

- The cases Empty, Decl, Sort, Var, Subtype, Pair, and Proj1 are straightforward by induction hypothesis.
- The case Proj2 is proved straightforwardly using the induction hypothesis and Proposition 5.4.1.
- In the case Prod, we separate the three possibilities for (s_1, s_2, s_3). Each subcase is straightforward.
- The cases Lam and App are similar. Using the notations given in these rules, we apply the induction hypotheses and separate the three possibilities for $\Pi v : T.U$. In the case Lam, each subcase is straightforward using the induction hypotheses. In the case App, each subcase is straightforward using the induction hypotheses and Proposition 5.4.1.
- The case Conversion follows from Proposition 5.4.3.

□

5.5 A type preserving reduction

In the case of PTSs (resp. PTSs with dependent pairs), the property of subject reduction, i.e. the type preservation of $\twoheadrightarrow_\beta$ (resp. $\twoheadrightarrow_{\beta\sigma}$), always holds. However, because of its conversion rule based on the relation $\equiv_{\beta *}$, PVS-Cert is not a PTS with dependent pairs and, as mentioned in Section 4.2, the relation $\twoheadrightarrow_{\beta *}$ is not a type preserving reduction in PVS-Cert. However, we will prove that the reduction $\twoheadrightarrow_{\beta\sigma}$ is type preserving. This reduction will be used both in the type-checking algorithm (Chapter 7) and as a definition of cut elimination for PVS-Cert proofs (Chapter 6), which will be itself at the core of the study of the logical properties of this system.

The specificity of this proof of type preservation compared to similar results for PTSs lies in the fact that $M \twoheadrightarrow_{\beta\sigma} N$ does not imply $M \equiv_{\beta *} N$ in general. However, we will prove that it is the case if M is either a type or an expression. This theorem will be used

as the most important lemma of the type preservation theorem for $\twoheadrightarrow_{\beta\sigma}$. It is presented together with the stability of types and expressions under $M \twoheadrightarrow_{\beta\sigma} N$.

Theorem 5.5.1. *For any term M, the following hold:*

- *If $M \to_{\beta\sigma} N$ and M is a type (resp. an expression), so is N, and $M \equiv_{\beta *} N$.*
- *If $M \twoheadrightarrow_{\beta\sigma} N$ and M is a type (resp. an expression), so is N, and $M \equiv_{\beta *} N$.*

Proof. The proof of the first statement is done by induction on M (as a type or an expression), using Proposition 5.4.1. We conclude the second one by induction on the length of the reduction. □

We will also use the following lemma, which simplifies the proof of type preservation.

Lemma 5.5.1. *If*

- *$\Gamma, v : T, \Delta \vdash M : N$ (resp. $\Gamma, v : T, \Delta \vdash WF$) is derivable,*
- *$\Gamma \vdash U : s$ with $T \equiv_{\beta *} U$ is derivable, then*

$\Gamma, v : U, \Delta \vdash M : N$ (resp. $\Gamma, v : U, \Delta \vdash WF$) is derivable

Proof. The proof is done by induction on the first derivation.

- The case Empty cannot occur.
- The case Decl splits into two subcases, depending on whether Δ is empty or not. If Δ is empty, we use the stratification theorem and Proposition 5.4.3 to establish $s = s(v)$. Then, we conclude applying Decl to the second derivation. If Δ is not empty, we conclude directly by induction hypothesis.
- The case Var splits into two subcases. With the notation $v' : T'$ for the conclusion, using the free variable theorem, $(v' : T')$ appears exactly once in the context, either in Γ, Δ or in $\{(v : T)\}$. In the first case we conclude directly by induction hypothesis. In the second case we first use the induction hypothesis to derive $\Gamma, v : U, \Delta \vdash WF$. From the subderivations theorem, $\Gamma \vdash T : s'$ is derivable for some s', hence we conclude from the thinning theorem that $\Gamma, v : U, \Delta \vdash T : s'$ if derivable. On the other hand, $\Gamma, v : U, \Delta \vdash v : U$ is derivable using Var. As $T \equiv_{\beta *} U$, we conclude using Conversion.
- All other cases are straightforward by induction hypothesis.

□

The main lemma of the type preservation for $\twoheadrightarrow_{\beta\sigma}$ is the following.

Proposition 5.5.1. *Given a derivable judgement $\Gamma \vdash M : T$, and N such that $M \to_{\beta\sigma} N$, the judgement $\Gamma \vdash N : T$ is derivable.*

Proof. The proof is done by induction on the derivation. We first consider all cases where $M \not\triangleright_{\beta\sigma} N$.

- The cases EMPTY, DECL, SORT, and VAR cannot occur.
- The cases PROJ1 and CONVERSION are straightforward by induction hypothesis.
- The cases PROD and SUBTYPE are similar. We present the proof for the case PROD. Discarding the notations of the original statement, the last inference step matches some rule instance

$$\frac{\Gamma \vdash T : s_1 \qquad \Gamma, v : T \vdash U : s_2}{\Gamma \vdash \Pi v : T.U : s_3}\ \textsc{Prod}\ (s_1, s_2, s_3) \in \mathcal{R}$$

  If the reduction occurs in U, we conclude directly by induction hypothesis. If the reduction hypothesis occurs in T, we write $T \to_{\beta\sigma} T'$. By induction hypothesis, $\Gamma \vdash T' : s_1$ is derivable. By the stratification theorem and Theorem 5.5.1, $T \equiv_{\beta *} T'$. Hence, using Lemma 5.5.1, we conclude from the second premise that $\Gamma, v : T' \vdash U : s_2$ is derivable. Finally, using PROD, $\Gamma \vdash \Pi v : T'.U : s_3$ is derivable.

- The case LAM is proved in the following way. Discarding the notations of the original statement, the last inference step matches some instance

$$\frac{\Gamma, v : T \vdash M : U \qquad \Gamma \vdash \Pi v : T.U : s}{\Gamma \vdash \lambda v : T.M : \Pi v : T.U}\ \textsc{Lam}$$

  If the reduction occurs in M, we conclude directly by induction hypothesis. If the reduction hypothesis occurs in T, we write $\lambda v : T.M \to_{\beta\sigma} \lambda v : T'.M$. By induction hypothesis, $\Gamma \vdash \Pi v : T'.U : s$ is derivable. By the stratification theorem and Theorem 5.5.1, $T \equiv_{\beta *} T'$. On the other hand, by the subderivations theorem, $\Gamma \vdash T' : s'$ is derivable for some sort s'. Hence, using Lemma 5.5.1, we conclude from the first premise that $\Gamma, v : T' \vdash M : U$ is derivable. Finally, using LAM, $\lambda v : T'.M : \Pi v : T'.U$ is derivable.

- The case APP is proved in the following way. Discarding the notations of the original statement and using the renaming theorem, the last inference step matches some instance of APP

$$\frac{\Gamma \vdash M_1 : \Pi v : T.U \qquad \Gamma \vdash M_2 : T}{\Gamma \vdash M_1 M_2 : U[M_2/v]}\ \textsc{App}$$

  with $v \notin DV(\Gamma)$. If the reduction occurs in M_1, we conclude directly by induction hypothesis. Else, we write $M_1 M_2 \to_{\beta\sigma} M_1 N_2$. By induction hypothesis followed by APP, $\Gamma \vdash M_1 N_2 : U[N_2/v]$ is derivable. By the stratification theorem, $U[M_2/v] \neq Kind$, hence, by Theorem 5.2.3, $\Gamma \vdash U[M_2/v] : s$ is derivable for some sort s. On the other hand, as $M_2 \to_{\beta\sigma} N_2$, $U[M_2/v] \twoheadrightarrow_{\beta\sigma} U[N_2/v]$. Hence,

by the stratification theorem followed by Theorem 5.5.1, $U[M_2/v] \equiv_{\beta *} U[N_2/v]$. Therefore, applying conversion, $\Gamma \vdash M_1 N_2 : U[M_2/v]$ is derivable.

- The case PAIR is proved in the following way. Discarding the notations of the original statement and using the renaming theorem, the last inference step matches some instance of PAIR

$$\frac{\Gamma \vdash M_1 : T \qquad \Gamma \vdash M_2 : U[M_1/v] \qquad \Gamma \vdash \{v : T \mid U\} : Type}{\Gamma \vdash \langle M_1, M_2 \rangle_{\{v:T|U\}} : \{v : T \mid U\}} \text{ PAIR}$$

with $v \notin DV(\Gamma)$. There are four cases:

  - If the reduction occurs in M_1, we write $M_1 \rightarrow_{\beta\sigma} N_1$. By induction hypothesis, $\Gamma \vdash N_1 : T$ is derivable. On the other hand, using the subderivation theorem on the third premise followed by the renaming theorem, $\Gamma, v : T \vdash U : Prop$ is derivable. Hence, by the substitution theorem, $\Gamma \vdash U[N_1/v] : Prop$ is derivable. By the stratification theorem followed by Theorem 5.5.1, $M_1 \equiv_{\beta *} N_1$, hence $U[M_1/v] \equiv_{\beta *} U[N_1/v]$. Therefore, applying conversion, $\Gamma \vdash M_2 : U[N_1/v]$ is derivable, and we conclude the expected result applying PAIR.
  - If the reduction occurs in M_2, we conclude directly by induction hypothesis.
  - If the reduction occurs in T, we write $T \rightarrow_{\beta\sigma} T'$. By induction hypothesis, $\Gamma \vdash \{v : T' \mid U\} : Type$ is derivable. By the subderivations theorem, $\Gamma \vdash T' : Type$ is derivable too. Hence, by the stratification theorem followed by Theorem 5.5.1, $T \equiv_{\beta *} T'$. Therefore, applying conversion, $\Gamma \vdash M_1 : T'$ is derivable, and we conclude the expected result applying PAIR.
  - Else, the reduction occurs in U. We write $U \rightarrow_{\beta\sigma} U'$. By induction hypothesis, $\Gamma \vdash \{v : T \mid U'\} : Type$ is derivable. By the subderivation theorem, $\Gamma, v : T \vdash U : Prop$ and $\Gamma, v : T \vdash U' : Prop$ are derivable. Hence, by the substitution theorem, $\Gamma, v : T \vdash U[M_1/v] : Prop$ and $\Gamma, v : T \vdash U'[M_1/v] : Prop$ are derivable. By the stratification theorem followed by Theorem 5.5.1, $U \equiv_{\beta *} U'$, hence $U[M_1/v] \equiv_{\beta *} U'[M_1/v]$. Therefore, applying conversion, $\Gamma \vdash M_2 : U'[N_1/v]$ is derivable, and we conclude the expected result applying PAIR.

- The case PROJ2 is proved in the following way. Discarding the notations of the original statement and using the renaming theorem, the last inference step matches some instance of PROJ2

$$\frac{\Gamma \vdash M : \{v : T \mid U\}}{\Gamma \vdash \pi_2(M) : U[\pi_1(M)/v]} \text{ PROJ2}$$

with $v \notin DV(\Gamma)$. As the reduction occurs in M, we write $\pi_2(M) \rightarrow_{\beta\sigma} \pi_2(N)$. By induction hypothesis followed by PROJ2, $\Gamma \vdash \pi_2(N) : U[\pi_1(N)/v]$ is derivable. By the stratification theorem, $U[\pi_1(M)/v] \neq Kind$, hence, by Theorem 5.2.3,

$\Gamma \vdash U[\pi_1(M)/v] : s$ is derivable for some sort s. By the stratification theorem and Theorem 5.5.1, $U[\pi_1(M)/v] \equiv_{\beta *} U[\pi_1(N)/v]$. Therefore, applying conversion, $\Gamma \vdash \pi_2(N) : U[\pi_1(M)/v]$ is derivable.

Last, we consider all cases where $M \triangleright_{\beta\sigma} N$.

- The cases Empty, Decl, Sort, Var, Prod, Lam, Subtype, and Pair cannot occur.
- The case Conversion is straightforward by induction hypothesis.
- The case App is proved in the following way. As M is an application, we replace the notations of the original statement by $(\lambda v : T.M)N \triangleright_{\beta\sigma} M[N/v]$. Matching a rule instance is stable by α-conversion, as well as $\rightarrow_{\beta\sigma}$, hence we can suppose $v \notin DV(\Gamma)$ without loss of generality. Using the renaming theorem, the last inference step matches some instance of App

$$\frac{\Gamma \vdash \lambda v : T.M : \Pi v : T'.U' \qquad \Gamma \vdash N : T'}{\Gamma \vdash (\lambda v : T.M)N : U'[N/v]}\ \textsc{App}$$

Using the subderivations theorem on the first premise, the hypothesis $v \notin DV(\Gamma)$, and the renaming theorem, we conclude that there exists some term $\Pi v : T.U \equiv_{\beta *} \Pi v : T'.U'$ such that $\Gamma \vdash \lambda v : T.M : \Pi v : T.U$ admits a derivation ending with an inference step matched by some rule instance

$$\frac{\Gamma, v : T \vdash M : U \qquad \Gamma \vdash \Pi v : T.U : s}{\Gamma \vdash \lambda v : T.M : \Pi v : T.U}\ \textsc{Lam}$$

By the substitution theorem, we conclude that $\Gamma \vdash M[N/v] : U'[N/v]$ is derivable from the second App premise $\Gamma \vdash N : T'$ as long as $\Gamma, v : T' \vdash M : U'$ is also derivable. To prove this, we first use Theorem 5.3.2 to establish $T \equiv_{\beta *} T'$ and $U \equiv_{\beta *} U'$ from $\Pi v : T.U \equiv_{\beta *} \Pi v : T'.U'$, and we use these two equations to derive $\Gamma, v : T' \vdash M : U'$ from $\Gamma, v : T \vdash M : U$ in the two following steps.
First, we first prove $\Gamma, v : T' \vdash M : U$ using Lemma 5.5.1. This requires $T \equiv_{\beta *} T'$, which is already established, and that $\Gamma \vdash T' : s'$ is derivable for some sort s', which is the case from Theorem 5.2.3 applied to the second App premise $\Gamma \vdash N : T'$ (using the fact that, by the stratification theorem, $T' \neq Kind$).
Last, we conclude using conversion. This requires $U \equiv_{\beta *} U'$, which is already established, and that $\Gamma, v : T' \vdash U' : s'$ is derivable for some s'. To prove this latter requirement, we use Theorem 5.2.3 on the first App premise $\Gamma \vdash \lambda v : T.M : \Pi v : T'.U'$ to derive $\Gamma \vdash \Pi v : T'.U' : s''$ for some sort s'', and we conclude that $\Gamma, v : T' \vdash U' : s'$ is derivable for some sort s' from the subderivations theorem followed by the renaming theorem, using the fact that $v \notin DV(\Gamma)$.
- The case Proj1 is proved in the following way. As M is a first projection, we replace the notations of the original statement by $\pi_1\langle M, N\rangle_T \triangleright_{\beta\sigma} M$. Using the

renaming theorem, the last inference step matches some instance of PROJ1

$$\frac{\Gamma \vdash \langle M, N\rangle_T : \{v : T' \mid U'\}}{\Gamma \vdash \pi_1\langle M, N\rangle_T : T'}\ \textsc{Proj1}$$

with $v \notin DV(\Gamma)$. Using the subderivations theorem followed by the renaming theorem, T has the form $\{v : T'' \mid U''\}$ where $\{v : T' \mid U'\} \equiv_{\beta *} \{v : T'' \mid U''\}$ and $\Gamma \vdash \langle M, N\rangle_T : \{v : T'' \mid U''\}$ admits a derivation ending with an inference step matched by some rule instance

$$\frac{\Gamma \vdash M : T'' \qquad \Gamma \vdash N : U''[M/v] \qquad \Gamma \vdash \{v : T'' \mid U''\} : Type}{\Gamma \vdash \langle M, N\rangle_T : \{v : T'' \mid U''\}}\ \textsc{Pair}$$

We derive the expected judgement $\Gamma \vdash M : T'$ from the first premise of this latter derivation using conversion. For this, we need to prove $T'' \equiv_{\beta *} T'$ and to derive $\Gamma \vdash T' : s$ for some s. These two requirement are proved as follows. On the one hand, we establish $T'' \equiv_{\beta *} T'$ from $\{v : T'' \mid U''\} \equiv_{\beta *} \{v : T' \mid U'\}$ using Theorem 5.3.2. On the other hand, by the stratification theorem, $T' \neq Kind$, hence we can use Theorem 5.2.3 on the original conclusion to establish that $\Gamma \vdash T' : s$ is derivable from some sort s.

- The case PROJ2 is proved in the following way. As M is a second projection, we replace the notations of the original statement by $\pi_2\langle M, N\rangle_T \triangleright_{\beta\sigma} N$. Using the renaming theorem, the last inference step matches some instance of PROJ2

$$\frac{\Gamma \vdash \langle M, N\rangle_T : \{v : T' \mid U'\}}{\Gamma \vdash \pi_2\langle M, N\rangle_T : U'[\pi_1\langle M, N\rangle_T/v]}\ \textsc{Proj2}$$

with $v \notin DV(\Gamma)$. Using the subderivations theorem followed by the renaming theorem, T has the form $\{v : T'' \mid U''\}$ where $\{v : T' \mid U'\} \equiv_{\beta *} \{v : T'' \mid U''\}$ and $\Gamma \vdash \langle M, N\rangle_T : \{v : T'' \mid U''\}$ admits a derivation ending with an inference step matched by some rule instance

$$\frac{\Gamma \vdash M : T'' \qquad \Gamma \vdash N : U''[M/v] \qquad \Gamma \vdash \{v : T'' \mid U''\} : Type}{\Gamma \vdash \langle M, N\rangle_T : \{v : T'' \mid U''\}}\ \textsc{Pair}$$

We derive the expected judgement $\Gamma \vdash N : U'[\pi_1\langle M, N\rangle_T/v]$ from the second premise of this latter derivation using conversion. For this, we need to prove $U''[M/v] \equiv_{\beta *} U'[\pi_1\langle M, N\rangle_T/v]$ and to derive $\Gamma \vdash U'[\pi_1\langle M, N\rangle_T/v] : s$ for some sort s. These two requirement are proved as follows. On the one hand, we establish $U' \equiv_{\beta *} U''$ from $\{v : T'' \mid U''\} \equiv_{\beta *} \{v : T' \mid U'\}$ using Theorem 5.3.2, and, as $M \equiv_{\beta *} \pi_1\langle M, N\rangle_T$, we conclude $U''[M/v] \equiv_{\beta *} U'[\pi_1\langle M, N\rangle_T/v]$. On the other hand, by the stratification theorem, $U'[\pi_1\langle M, N\rangle_T/v] \neq Kind$, hence we can use Theorem 5.2.3 on the original conclusion to establish that $\Gamma \vdash U'[\pi_1\langle M, N\rangle_T/v] : s$ is derivable from some sort s.

□

The type preservation property for $\twoheadrightarrow_{\beta\sigma}$ is presented in the following theorem. More precisely, type preservation corresponds to the first statement, and the second statement is a direct consequence of this property.

Theorem 5.5.2 (Type preservation for $\twoheadrightarrow_{\beta\sigma}$)**.** *Given any derivable judgement* $\Gamma \vdash M : T$ *the following statements hold.*

- *If* $M \twoheadrightarrow_{\beta\sigma} N$*, then* $\Gamma \vdash N : T$ *is derivable.*
- *If* $T \twoheadrightarrow_{\beta\sigma} U$*, then* $\Gamma \vdash M : U$ *is derivable.*

Proof. The first statement is obtained directly by induction on the length of the reduction $M \twoheadrightarrow_{\beta\sigma} N$, using Proposition 5.5.1.

The second statement is proved as follows. If $T = U$, the result is straightforward. Else, the length of the reduction $T \twoheadrightarrow_{\beta\sigma} U$ is strictly positive, hence $T \neq Kind$. By the Theorem 5.2.3, $\Gamma \vdash T : s$ is derivable for some sort s. Hence, using the first statement, $\Gamma \vdash U : s$ is derivable. On the other hand, by the stratification theorem and the fact that the length of the reduction $T \twoheadrightarrow_{\beta\sigma} U$ is strictly positive, T is either a type or an expression. Hence, using Theorem 5.5.1, $T \equiv_{\beta *} U$. Therefore, we can apply the conversion rule to obtain a derivation of $\Gamma \vdash M : U$ as expected. □

The type preservation theorem shows that $\twoheadrightarrow_{\beta\sigma}$ transforms the proof of a given theorem into another one. Hence, it defines one step of cut elimination for the proofs of PVS-Cert. In the Chapter 6, we prove that both $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta *}$ terminates on well-typed terms. This will lead to establish several properties of the certificate system, including the decidability of typechecking (proved in Chapter 7).

5.6 Uniqueness of types

The addition of coercions is PVS-Cert was made to ensure the decidability of type-checking, which will be proved in Chapter 7. Besides the type preservation theorem proved in the previous section, one of the most important properties of PVS-Cert used in this proof and its underlying type-checking algorithm is the uniqueness of types (modulo conversion). This property also underlines that, even though PVS-Cert is designed to reflect predicate subtyping, it doesn't admit any subtyping itself.

Theorem 5.6.1 (Uniqueness of types)**.** *If two judgements* $\Gamma \vdash M : T_0$ *and* $\Gamma \vdash M : T_1$ *are derivable, then* $T_0 \equiv_{\beta *} T_1$*.*

Proof. The proof is done by induction on the sum of the height of the two derivations. The possible cases are the following:

- There exists $i \in \{0, 1\}$ such that the derivation of $\Gamma \vdash M : T_i$ matches some rule instance

$$\frac{\Gamma \vdash M : T_i' \qquad \Gamma \vdash T_i : s}{\Gamma \vdash M : T_i}\ \textsc{Conversion}\ T_i' \equiv_{\beta *} T_i$$

By induction hypothesis, $T_i' \equiv_{\beta *} T_{1-i}$, hence $T_0 \equiv_{\beta *} T_1$.

- For all $i \in \{0, 1\}$, the derivation of $\Gamma \vdash M : T_i$ matches some rule instance of Sort: this case is straightforward, using the fact that an axiom $(s, s') \in \mathcal{A}$ is determined by s.

- For all $i \in \{0, 1\}$, the derivation of $\Gamma \vdash M : T_i$ matches some rule instance of Prod: this case is straightforward by induction hypothesis, using the fact that a rule $(s, s', s'') \in \mathcal{R}$ is determined by (s, s').

- For all $i \in \{0, 1\}$, the derivation of $\Gamma \vdash M : T_i$ matches some rule instance of Var, Subtype, or Pair: these three cases are straightforward, as T_i is determined by Γ and M.

- For all $i \in \{0, 1\}$, the derivation of $\Gamma \vdash M : T_i$ matches some rule instance of Lam, App, Proj1, or Proj2: these four cases are straightforward by induction hypothesis. We present the case Lam as an example. In this case, the two derivations match, respectively, some rules

  $$\frac{\Gamma, v : U \vdash N : U_i \qquad \Gamma \vdash \Pi v : U.U_i : s_i}{\Gamma \vdash \lambda v : U.N : \Pi v : U.U_i}\ \textsc{Lam}$$

  By induction hypothesis, $U_0 \equiv_{\beta *} U_1$, hence $T_0 = \Pi v : U.U_0 \equiv_{\beta *} \Pi v : U.U_1 = T_1$ as expected.

□

We end this section with the following corollary of Theorem 5.6.1, which will be useful to prove the completeness of the type-checking algorithm for PVS-Cert presented in Chapter 7.

Corollary 5.6.1. *If two judgements $\Gamma \vdash M : s$ and $\Gamma \vdash M : T$ are derivable, $T = s$.*

Proof. By Theorem 5.6.1, $T \equiv_{\beta *} s$. By the stratification theorem, T is either a type, *Type*, or *Kind*. In all cases, Theorem 5.3.2 applies, hence $T = s$. □

5.7 Additional observations on PVS-Cert

This chapter is ended with the presentation of two additional observations in PVS-Cert: the fact that it is an extension of the system PVS-Cert$^-$ defined in Definition 4.2.3, and the possibility to equip it with admissible rules of context conversion.

5.7.1 PVS-Cert extends PVS-Cert$^-$

We consider the PTS with dependent pairs PVS-Cert$^-$ defined in Definition 4.2.3. As mentioned in Section 4.2, PVS-Cert$^-$ can be considered as a more standard system: it is a PTS extended with dependent pairs, and a subsystem of the type system ECC [53]. We will prove here that PVS-Cert is an extension of PVS-Cert$^-$: any judgement derivable in PVS-Cert$^-$ is derivable in PVS-Cert.

We begin with the following lemma.

Lemma 5.7.1. *If two terms M_0 and M_1 are either types, expressions, Type, or Kind, and if $M_0 \equiv_{\beta\sigma} M_1$, then $M_0 \equiv_{\beta *} M_1$.*

Proof. $\mathcal{T}$ equipped with $\rightarrow_{\beta\sigma}$ is an orthogonal combinatory reduction system (as defined in [48]), as rules are left-linear and non-overlapping. As proved in the same paper, such a system admits the Church-Rosser property. Hence, there exists a term N such that $M_i \twoheadrightarrow_{\beta\sigma} N$ for all $i \in \{0,1\}$. If M_0 and M_1 are either types or expressions, we conclude $M_0 \equiv_{\beta *} M_1$ by Theorem 5.5.1. If there exists $i \in \{0,1\}$ such that M_i is $Type$ (resp. $Kind$), then $N = Type$ (resp. $N = Kind$). As M_{1-i} is either a type, an expression, $Type$, or $Kind$, we conclude $M_{1-i} = Type$ (resp. $M_{1-i} = Kind$), hence $M_0 \equiv_{\beta *} M_1$. □

We conclude that PVS-Cert is an extension of PVS-Cert$^-$ as follows.

Theorem 5.7.1. *PVS-Cert is an extension of PVS-Cert$^-$: for any judgement $\Gamma \vdash M : T$ derivable in PVS-Cert$^-$, the same judgement is derivable in PVS-Cert.*

Proof. The proof is done by induction on the derivation. The only problematic case is conversion, in which we can conclude using the stratification theorem followed by Lemma 5.7.1. □

5.7.2 Defining conversion on contexts

We present a notion of conversion for contexts, and prove the admissibility of some rules of conversion for context. The conversion of contexts is defined as follows.

Definition 5.7.1. *Two contexts Γ and Δ, we define $\Gamma \equiv_{\beta *} \Delta$ as the existence of variables $v_1, ..., v_n$ and terms $T_1 \equiv_{\beta *} U_1$, ..., $T_n \equiv_{\beta *} U_n$ such that $\Gamma = v_1 : T_1, ..., v_n : T_n$ and $\Delta = v_1 : U_1, ..., v_n : U_n$.*

The conversion of contexts is admissible in PVS-Cert in the following ways.

Proposition 5.7.1. *The two following statements hold.*

- *For any derivable PVS-Cert judgements $\Gamma \vdash M : T$ and $\Delta \vdash WF$ such that $\Gamma \equiv_{\beta *} \Delta$, the judgement $\Delta \vdash M : T$ is derivable in PVS-Cert.*
- *For any derivable PVS-Cert judgement $\Gamma \vdash M : T$ and $\Delta \vdash U : s$ such that $\Gamma \equiv_{\beta *} \Delta$ and $T \equiv_{\beta *} U$, the judgement $\Delta \vdash M : U$ is derivable in PVS-Cert.*

Proof. The first statement is proved by strong induction on the height of the PVS-Cert derivation of $\Gamma \vdash M : T$. The possible cases are the following.

- The cases Empty and Decl cannot occur.
- The case Sort is straightforward, applying the corresponding Sort rule instance to the derivation of $\Delta \vdash WF$.
- The case Var is proved as follows. The last inference step matches some instance of the following form.

  $$\frac{\Gamma \vdash WF}{\Gamma \vdash v : T}\ \textsc{Var}\ (v : T) \in \Gamma$$

  As $(v : T) \in \Gamma$, there exists some declaration $(v : U) \in \Delta$ such that $T \equiv_{\beta *} U$. We write $\Gamma = \Gamma_1, v : T, \Gamma_2$ and $\Delta = \Delta_1, v : U, \Delta_2$ with $\Gamma_1 \equiv_{\beta *} \Gamma_2$ and $\Delta_1 \equiv_{\beta *} \Delta_2$. By the subderivations theorem, there exists some subderivation of the derivation of $\Gamma \vdash WF$ with conclusion $\Gamma_1 \vdash T : s(v)$. By the subderivations theorem again, $\Delta_1 \vdash WF$ is also derivable. Hence, by induction hypothesis, $\Delta_1 \vdash T : s(v)$ is derivable. Thus, by the thinning theorem, $\Delta \vdash T : s(v)$ is derivable as well. On the other hand, applying the rule Var, $\Delta \vdash v : U$ is derivable in PVS-Cert. Therefore, applying conversion, $\Delta \vdash v : T$ is derivable in PVS-Cert.
- The case Prod is proved as follows. The last inference step matches some instance of the following form.

  $$\frac{\Gamma \vdash T : s_1 \qquad \Gamma, v : T \vdash U : s_2}{\Gamma \vdash \Pi v : T.U : s_3}\ \textsc{Prod}\ (s_1, s_2, s_3) \in \mathcal{R}$$

  By the subderivations theorem, there exists some subderivation of the derivation of the second premise with conclusion $\Gamma \vdash T : s(v)$. Hence, by the stratification theorem, as $\Gamma \vdash T : s_1$ is derivable as well, $s(v) = s_1$. Moreover, by induction hypothesis, $\Delta \vdash T : s(v)$ is derivable. By the free variable theorem, $v \notin DV(\Gamma)$. As $\Gamma \equiv_{\beta *} \Delta$, $v \notin DV(\Delta) = DV(\Gamma)$. Therefore, the rule Decl can be applied to conclude that $\Delta, v : T \vdash WF$ is derivable. By induction hypothesis on the second premise, we conclude that $\Delta, v : T \vdash U : s_2$ is derivable. Applying the rule Prod, we conclude that $\Delta \vdash \Pi v : T.U : s_3$ is derivable in PVS-Cert.
- The case Lam is proved as follows. The last inference step matches some instance of the following form.

  $$\frac{\Gamma, v : T \vdash M : U \qquad \Gamma \vdash \Pi v : T.U : s}{\Gamma \vdash \lambda v : T.M : \Pi v : T.U}\ \textsc{Lam}$$

  By induction hypothesis, $\Delta \vdash \Pi v : T.U : s$ is derivable. On the other hand, by the subderivations theorem, there exists some subderivation of the derivation of

the first premise with conclusion $\Gamma \vdash T : s(v)$. Hence, by induction hypothesis, $\Delta \vdash T : s(v)$ is derivable. By the free variable theorem, $v \notin DV(\Gamma)$. As $\Gamma \equiv_{\beta*} \Delta$, $v \notin DV(\Delta) = DV(\Gamma)$. Therefore, the rule DECL can be applied to conclude that $\Delta, v : T \vdash WF$ is derivable. In this setting, by induction hypothesis on the first premise, $\Delta, v : T \vdash M : U$ is derivable. Applying the rule LAM, we conclude that $\Delta \vdash \lambda v : T.M : \Pi v : T.U$ is derivable in PVS-Cert.

- The case APP is straightforward by induction hypothesis.
- The case SUBTYPE is proved as follows. The last inference step matches some instance of the following form.

$$\frac{\Gamma \vdash T : Type \qquad \Gamma, v : T \vdash U : Prop}{\Gamma \vdash \{v : T \mid U\} : Type}\ \textsc{Subtype}$$

  By the subderivations theorem, there exists some subderivation of the derivation of the second premise with conclusion $\Gamma \vdash T : s(v)$. Hence, by the stratification theorem, as $\Gamma \vdash T : s_1$ is derivable as well, $s(v) = Type$. Moreover, by induction hypothesis, $\Delta \vdash T : s(v)$ is derivable. By the free variable theorem, $v \notin DV(\Gamma)$. As $\Gamma \equiv_{\beta*} \Delta$, $v \notin DV(\Delta) = DV(\Gamma)$. Therefore, the rule DECL can be applied to conclude that $\Delta, v : T \vdash WF$ is derivable. By induction hypothesis on the second premise, we conclude that $\Delta, v : T \vdash U : Prop$ is derivable. Applying the rule SUBTYPE, we conclude that $\Delta \vdash \{v : T \mid U\} : Type$ is derivable in PVS-Cert.
- The cases PAIR, PROJ1, PROJ2, and CONVERSION are straightforward by induction hypothesis.

The second statement is proved as followed, using the first statement. We consider two PVS-Cert judgements $\Gamma \vdash M : T$ and $\Delta \vdash U : s$ such that $\Gamma \equiv_{\beta*} \Delta$ and $T \equiv_{\beta*} U$. By the subderivations theorem, $\Delta \vdash WF$ is derivable, hence, using the first statement, $\Delta \vdash M : T$ is derivable. Hence, applying conversion, $\Delta \vdash M : U$ is derivable in PVS-Cert. □

Chapter 6

Strong normalization in PVS-Cert

In this chapter, we prove that any reduction from a well-typed term using $\rightarrow_{\beta\sigma}$ (resp. $\rightarrow_{\beta *}$) terminates. These two reductions will be used separately in Chapter 7 to define a type-checking algorithm for PVS-Cert: more precisely, the reduction $\rightarrow_{\beta *}$ is used to decide whether two well-typed terms are convertible with $\equiv_{\beta *}$, while the reduction $\rightarrow_{\beta\sigma}$ will be used in the type-checking of applications. Moreover, the strong normalization of $\rightarrow_{\beta\sigma}$ combined with the type preservation theorem defines a cut elimination for the system PVS-Cert, which is a powerful tool to delineate the provable propositions from the unprovable ones, as illustrated in a proof of consistency of PVS-Cert presented in Theorem 6.7.1 and a characterization of Leibniz's equality presented in Theorem 6.7.2.

A direct approach to prove the strong normalization of both $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta *}$ for well-typed terms would be to prove the strong normalization for well-typed terms of their union, referred to as $\rightarrow_{\beta\sigma *}$. Unfortunately, this reduction is not strongly terminating on well-typed terms, as shown in the following proposition.

Proposition 6.0.1. *There exists a well-typed term admitting an infinite reduction using* $\rightarrow_{\beta\sigma *}$.

Proof. We find a well-typed term admitting an infinite reduction using $\rightarrow_{\beta\sigma *}$ in the following way. As PVS-Cert is an extension of System F [40], it is possible to define two well-typed terms M and N such that MN admits an infinite reduction. We consider two such terms together with some other terms M' and N' such that the application $M\ \pi_2\langle\langle M', N\rangle_T, N'\rangle_U$ is well-typed. As $M\ \pi_2\langle\langle M', N\rangle_T, N'\rangle_U \rightarrow_{\beta\sigma *} M\ \pi_2\langle M', N\rangle_T \rightarrow_{\beta\sigma *} MN$, it admits an infinite reduction.

More precisely, a well-typed term admitting an infinite reduction is built as follows.

- We define $1 = \Pi P : Prop.\Pi h : P.P$ together with $M = \lambda h : 1.h\ 1\ h$ and $N = \lambda h' : 1.\lambda h : 1.h\ 1\ h$
- $\vdash M : (\Pi h : 1.1)$ and $\vdash N : (\Pi h' : 1.\Pi h : 1.1)$ are derivable.

- $MN \to_{\beta\sigma *} (\lambda h' : 1.\lambda h : 1.h\ 1\ h)\ 1\ (\lambda h' : 1.\lambda h : 1.h\ 1\ h) \to_{\beta\sigma *} MN$, hence MN admits an infinite reduction
- We define $M' = 1$, $N' = \lambda P : Prop.\lambda h : P.h$,
  $T = \{x : Prop \mid \Pi h' : 1.\Pi h : 1.1\}$, and $U = \{y : T \mid 1\}$.
- $\vdash \langle\langle M', N\rangle_T, N'\rangle_U : U$ is derivable,
  hence $\vdash M\ \pi_2\langle\langle M', N\rangle_T, N'\rangle_U : 1$ is derivable.
- $M\ \pi_2\langle\langle M', N\rangle_T, N'\rangle_U \twoheadrightarrow_{\beta\sigma *} MN$, hence it admits an infinite reduction.

□

As a consequence, the reductions $\to_{\beta\sigma}$ and $\to_{\beta *}$ have to be kept separated. A second approach could be to prove strong normalization under $\to_{\beta\sigma}$ first – as this reduction is standard and used in several other type systems which strongly normalize – and to try to conclude the case $\to_{\beta *}$ from the strong normalization of $\to_\beta$ and $\to_*$. Some results allowing to prove the strong normalization of the union of two strongly normalizing reductions are investigated e.g. in [23] with the notion of jump and in [25] with the notion of stable classes of terms. Unfortunately, these notions do not apply to the current case. In the first approach, neither the $\to_\beta$ jumps over $\to_*$ nor $\to_*$ jumps over $\to_\beta$. In the second approach, neither well-typed terms nor stratified terms are stable, hence there is no straightforward way to apply the corresponding theorem. As a consequence, our selected approach is to prove strong normalization under $\to_{\beta\sigma}$ and $\to_{\beta *}$ together instead of one after the other.

Proofs of strong normalization can be performed either directly, or through an encoding into some well-known reduction system such as the syntax of pure lambda calculus, as done e.g. in [19] in the case of the calculus of constructions. In this work, we prove strong normalization directly on terms. This avoids to split the two cases $\to_{\beta\sigma}$ and $\to_{\beta *}$ with two separate encodings, and allows to factor the most important part of the proof. In this setting, we introduce the set of strongly normalizing terms SN as follows.

Definition 6.0.1 (SN)**.**

- *We define* $\mathrm{SN} \subseteq \mathcal{T}$ *the set of terms which are both strongly normalizing under* $\to_{\beta\sigma}$ *and strongly normalizing under* $\to_{\beta *}$*.*
- *We refer to a term* $M \in \mathrm{SN}$ *as a strongly normalizing term.*

A naive strategy to prove strong normalization would be to attempt to prove by induction on derivations of the form $\Gamma \vdash M : T$ that $M \in \mathrm{SN}$. However, as illustrated in Proposition 6.0.1, $M \in \mathrm{SN}$ is too weak as an induction hypothesis to conclude in the case of an application. For this reason, the induction hypothesis will be strengthened in several ways. On the one hand, its definition will be based on a family of subsets of SN, allowing to consider more specific properties than simply belonging to SN. On the other hand, the induction hypothesis will be parametrized. The parameters, referred to as

valuations, will be presented in Definitions 6.4.3 and 6.5.1. In the following section, we focus first on the presentation of a family of subsets of SN. Several well-known families of subsets of SN can be used in such proofs, e.g. reducibility candidates as defined in [40] or saturated sets as defined in [73]. The current work is based on saturated sets.

Several ideas and notations are inspired from [35], which presents, among other things, a proof of strong normalization of an extension of the calculus of constructions to dependent pairs. However, the presence of the reduction $\rightarrow_{\beta *}$ leads to the following additions and changes:

- As justified before the definition of SN, we use sets of terms directly instead of encoding of terms into the syntax of pure lambda calculus as in [35].
- The proof of the expected properties of saturated sets are adapted with the new reduction $\rightarrow_{\beta *}$.
- The interpretation of terms as saturated sets are expected to be stable under $\rightarrow_{\beta *}$, hence they cannot be defined on well-typed terms only, as well-typed terms are not stable under $\rightarrow_{\beta *}$. In this setting, we extend the interpretations to types and expressions, which are stable under $\rightarrow_{\beta *}$, and we prove that they are well-defined not only on well-typed terms, but also on their reducts under $\rightarrow_{\beta *}$.

6.1 Saturated sets

We first give the definition of saturated sets together with some operations on them before providing some motivations for these definitions. The usual definition of saturated sets is based on the notions of base terms and key redexes. In the current work, we suggest a factorization of these notions through the following definition.

Definition 6.1.1 (Elimination contexts)**.**
We define the set of elimination contexts $\mathcal{E}$ *with the grammar* $e := \bullet \mid \pi_i(e) \mid e\ M$.
For any term N *we define the instantiation* $e[N]$ *by*

- $\bullet[N] = N$
- $\pi_i(e)[N] = \pi_i(e[N])$
- $(eM)[N] = (e[N])M$

In the following, the usual notion of *base term* will be replaced by *term of the form* $e[v]$ and the usual notion of *key redex* will be replaced by *term of the form either* $e[(\lambda v : T.M)N]$ *or* $e[\pi_i\langle M, N\rangle_T]$. The definition of saturated sets is the following.

Definition 6.1.2 (Saturated sets)**.**
We define $\mathrm{SAT} \subseteq \mathcal{P}(\mathcal{T})$ *in the following way.* $S \in \mathrm{SAT}$ *if and only if the following hold.*

1. $S \subseteq \mathrm{SN}$

2. $e[v] \in S$ *whenever* $e[v] \in \mathrm{SN}$

3. $e[(\lambda v : T.M)N] \in S$ *whenever* $e[(\lambda v : T.M)N] \in \mathrm{SN}$ *and* $e[M[N/v]] \in S$

4. $e[\pi_i\langle M_1, M_2\rangle_T] \in S$ *whenever* $e[\pi_i\langle M_1, M_2\rangle_T] \in \mathrm{SN}$ *and* $e[M_i] \in S$

Finally, we define the following operations on sets of terms, under which saturated sets will be proved to be stable.

Definition 6.1.3 (Operations)**.** *For* $S_1, S_2 \subseteq \mathcal{T}$*, we define*

- $S_1 \tilde{\to} S_2 = \{M \in \mathcal{T} \mid \forall N \in S_1, (MN) \in S_2\}$
- $S_1 \tilde{\times} S_2 = \{M \in \mathcal{T} \mid \pi_1(M) \in S_1 \wedge \pi_2(M) \in S_2\}$

The motivations for these definitions will be given informally, as their precise use depends on the definition of valuations, which will be presented later in Definitions 6.4.3 and 6.5.1. For every derivable judgement $\Gamma \vdash M : T$, the term T will be interpreted as a saturated sets, allowing to replace the naive induction hypothesis $M \in \mathrm{SN}$ – which is too weak in the case of an application – by a stronger induction hypothesis, which is the fact that M belongs to this saturated set. In this setting, the purpose of $\tilde{\to}$ (resp. $\tilde{\times}$) is to define the saturated sets associated with terms of the form $\Pi v : T.U$ (resp. $\{v : T \mid U\}$) from the saturated sets associated with T and U. The definitions of $\tilde{\to}$ and $\tilde{\times}$ are designed to make the induction hypothesis sufficiently strong in the case of applications and projections. On the other hand, the conditions 2, 3, and 4 of the definition of saturated sets, using $e = \bullet$, allow to conclude in all other cases. They are generalized from the case $e = \bullet$ to the case of an arbitrary elimination context e in order to ensure the stability of saturated sets under $\tilde{\to}$ and $\tilde{\times}$.

In the following section, several closure properties of SN will be established. They will be useful in the proof of stability of saturated sets under $\tilde{\to}$ and $\tilde{\times}$ stated in Theorem 6.3.1, but also in the main proof of strong normalization.

6.2 Closure properties of SN

In this section, we show three closure properties of SN, presented in Proposition 6.2.1, Proposition 6.2.2, and Proposition 6.2.3 respectively. These three properties are standard for the set terms which are strongly normalizing under $\to_{\beta\sigma}$. However, as SN is defined as the set of terms which are both strongly normalizing under $\to_{\beta\sigma}$ and strongly normalizing under $\to_{\beta *}$, the proofs of these properties are distinctive in the presence of the unusual reduction $\to_{\beta *}$. The three closure properties are based on the following lemma.

Lemma 6.2.1. *If* $e[M] \to_{_} N$ *where* $\to_{_}$ *is either* $\to_\beta$*,* $\to_\sigma$*, or* $\to_*$*, then one of the following holds:*

- N *has the form* $e'[M]$*, where for any term* M'*,* $e[M'] \to_{_} e'[M']$

- *N has the form $e[M']$ where $M \to_- M'$*
- *$\to_-$ is $\to_\beta$, M has the form $\lambda v : T.M_1$, e has the form $e'[\bullet M_2]$, and N has the form $e'[M_1[M_2/v]]$*
- *$\to_-$ is $\to_\sigma$, M has the form $\langle M_1, M_2\rangle_T$, e has the form $e'[\pi_i(\bullet)]$, and N has the form $e'[M_i]$ for some $i \in \{1, 2\}$*

Proof. The proof is straightforward by induction on e. □

Proposition 6.2.1. *The following statements hold.*

- *If $e[v]$ and N belong to* SN*, so does $e[v]N$*
- *If $e[v]$ belongs to* SN*, so does $\pi_i(e[v])$ for all $i \in \{1, 2\}$*

Proof. The proofs of the two statements are similar. We present the first one as an example. We prove that $e[v]N$ is strongly normalizing under $\to_{\beta\sigma}$ and $\to_{\beta *}$ separately. Again, the two cases being similar, we present the first one as an example.

We prove that $e[v]N$ is strongly normalizing under $\to_{\beta\sigma}$ by strong induction on the sum of the maximum length of reduction from the terms $e[v]$ and N respectively. We suppose that the property holds for whenever this number is strictly below n, and we prove it for n by proving that any one-step reduction of $e[v]N$ is strongly normalizing under $\to_{\beta\sigma}$. Using Lemma 6.2.1, such a reduct has the form $e'[v]$, where for any term M, $e[M] \to_{\beta\sigma} e'[M]$. In this setting, $e[v] \to_{\beta\sigma} e'[v]$, and we conclude by induction hypothesis. □

Proposition 6.2.2. *If T, M_2, and $e[M_1[M_2/v]]$ belong to* SN*, so does $e[(\lambda v : T.M_1)M_2]$*

Proof. We first prove that if $e[M_1[M_2/v]] \in$ SN, then $M_1 \in$ SN in the following way. As $M_1 \to_{\beta\sigma} M_1'$ implies $M_1[M_2/v] \to_{\beta\sigma} M_1'[M_2/v]$, is also implies $e[M_1[M_2/v]] \to_{\beta\sigma} e[M_1'[M_2/v]]$. Hence, whenever $e[M_1[M_2/v]]$ is strongly normalizing under $\to_{\beta\sigma}$, so is M_1. We conclude the same property with $\to_{\beta *}$ in the same way. Hence, $M_1 \in$ SN whenever $e[M_1[M_2/v]] \in$ SN.

Using this result, it is sufficient to prove that if T, M_1, M_2, and $e[M_1[M_2/v]]$ belong to SN, so does $e[(\lambda v : T.M_1)M_2]$, i.e. $e[(\lambda v : T.M_1)M_2]$ is strongly normalizing under $\to_{\beta\sigma}$ and $\to_{\beta *}$. The two cases being similar, we present the second one as an example.

We prove that $e[(\lambda v : T.M_1)M_2]$ is strongly normalizing under $\to_{\beta *}$ by strong induction the sum of the maximum length of reduction from the terms T, M_1, M_2, and $e[M_1[M_2/v]]$ respectively.

We suppose that the property holds for whenever this number is strictly below n, and we prove it for n by proving that any one-step reduction of $e[(\lambda v : T.M_1)M_2]$ is strongly normalizing under $\to_{\beta *}$. Using Lemma 6.2.1, the possible cases are the following.

- This reduct has the form $e[N]$ where $(\lambda v : T.M_1)M_2 \rightarrow_{\beta *} N$. We split this case into the following subcases.
  - $e[N] = e[(\lambda v : T'.M_1)M_2]$ where $T \rightarrow_{\beta *} T'$. In this subcase, we conclude by induction hypothesis on T', M_1, M_2, and $e[M_1[M_2/v]]$.
  - $e[N] = e[(\lambda v : T.M_1')M_2]$ where $M_1 \rightarrow_{\beta *} M_1'$. We have $M_1[M_2/v] \rightarrow_{\beta *} M_1'[M_2/v]$, hence $e[M_1[M_2/v]] \rightarrow_{\beta *} e[M_1'[M_2/v]]$, and we can conclude by induction hypothesis on T, M_1', M_2, and $e[M_1'[M_2/v]]$.
  - $e[N] = e[(\lambda v : T.M_1)M_2']$ where $M_2 \rightarrow_{\beta *} M_2'$. We have $M_1[M_2/v] \twoheadrightarrow_{\beta *} M_1[M_2'/v]$, hence $e[M_1[M_2/v]] \twoheadrightarrow_{\beta *} e[M_1[M_2'/v]]$, and we can conclude by induction hypothesis on T, M_1, M_2', and $e[M_1'[M_2/v]]$.
  - $e[N] = e[M_1[M_2/v]]$, we conclude by hypothesis on $e[M_1[M_2/v]]$.
- This reduct has the form $e'[(\lambda v : T.M_1)M_2]$ where for all M', $e[M'] \rightarrow_{\beta *} e'[M']$. As $e[M_1[M_2/v]] \rightarrow_{\beta *} e'[M_1[M_2/v]]$, we conclude by induction hypothesis on T, M_1, M_2, and $e'[M_1[M_2/v]]$.

□

The last closure property, which is presented in Proposition 6.2.3, is the most complex one. Contrary to the two previous propositions, the analysis of strong normalization under $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta *}$ rely on different arguments. In the second case, the following lemma will be used.

Lemma 6.2.2. *The following statements hold.*

- *If $e[M_1]$, M_2, and T are strongly normalizing under $\rightarrow_{\beta *}$, so is $e[\langle M_1, M_2\rangle_T]$.*
- *If $e[M_1]$ is strongly normalizing under $\rightarrow_{\beta *}$, so is $e[\pi_1(M_1)]$.*
- *If $e[M_1]$ and M_2 are strongly normalizing under $\rightarrow_{\beta *}$, so is $e[\pi_2(M_2)]$.*

Proof. The proof is done by induction on the sum of the maximum length of reduction from $e[M_1]$, M_2, and T in the first case, by induction on the maximum length of reduction from $e[M_1]$ in the second case, and by induction on the sum of the maximum length of reduction from $e[M_1]$ and M_2 in the third case. The three proofs are similar. We present the first one as an example.

We suppose that the property holds for whenever the sum of the maximum length of reduction from $e[M_1]$, M_2, and T is strictly below n, and we prove it when it equals n by proving that any one-step reduction of $e[\langle M_1, M_2\rangle_T]$ is strongly normalizing under $\rightarrow_{\beta *}$. Using Lemma 6.2.1, the possible cases are the following.

- This reduct has the form $e[N]$ where $\langle M_1, M_2\rangle_T \rightarrow_{\beta *} N$. We split this case into the following subcases.

  - $e[N] = e[\langle M_1', M_2\rangle_T]$ where $M_1 \to_{\beta *} M_1'$. As $e[M_1] \to_{\beta *} e[M_1']$, we conclude by induction hypothesis with $e[M_1']$, M_2, and T.
  - $e[N] = e[\langle M_1, M_2'\rangle_T]$ where $M_2 \to_{\beta *} M_2'$. We conclude by induction hypothesis with $e[M_1]$, M_2', and T.
  - $e[N] = e[\langle M_1, M_2\rangle_{T'}]$ where $T \to_{\beta *} T'$. We conclude by induction hypothesis with $e[M_1]$, M_2, and T'.
  - $e[N] = e[M_1]$. We conclude by hypothesis on $e[M_1]$.
- This reduct has the form $e'[\langle M_1, M_2\rangle_T]$ where for all M_1', $e[M_1'] \to_{\beta *} e'[M_1']$. As $e[M_1] \to_{\beta *} e'[M_1]$, we conclude by induction hypothesis with $e'[M_1]$, M_2 and T.

□

We state and prove the last important closure property of SN as follows.

Proposition 6.2.3. *If M_1, M_2, T, and $e[M_i]$ belong to* SN*, so does $e[\pi_i\langle M_1, M_2\rangle_T]$.*

Proof. For M_1, M_2, T, and $e[M_i]$ belong to SN, we prove $e[\pi_i\langle M_1, M_2\rangle_T] \in$ SN by proving that $e[\pi_i\langle M_1, M_2\rangle_T]$ is strongly normalizing under $\to_{\beta\sigma}$ and $\to_{\beta *}$ separately.

On the one hand, we prove that $e[\pi_i\langle M_1, M_2\rangle_T]$ is strongly normalizing under $\to_{\beta\sigma}$ by strong induction the sum of the maximum length of reduction from the terms M_1, M_2, T, and $e[M_i]$ respectively.

We suppose that the property holds for whenever this number is strictly below n, and we prove it when it equals n by proving that any one-step reduction of $e[\pi_i\langle M_1, M_2\rangle_T]$ is strongly normalizing under $\to_{\beta\sigma}$. Using Lemma 6.2.1, the possible cases are the following.

- This reduct has the form $e[N]$ where $\pi_i\langle M_1, M_2\rangle_T \to_{\beta\sigma} N$. We split this case into the following subcases.
  - $e[N] = e[\pi_i\langle M_1', M_2'\rangle_T]$ where $M_j \to_{\beta\sigma} M_j'$ and $M_{2-j} = M_{2-j}'$ for some $j \in \{1, 2\}$. As either $e[M_i] \to_{\beta\sigma} e[M_i']$ or $e[M_i] = e[M_i']$, we conclude in both cases by induction hypothesis on M_1', M_2', T, and $e[M_i']$.
  - $e[N] = e[\pi_i\langle M_1, M_2\rangle_{T'}]$ where $T \to_{\beta\sigma} T'$. We conclude by induction hypothesis on M_1, M_2, T', and $e[M_i]$.
  - $e[N] = e[M_i]$. We conclude by hypothesis on $e[M_i]$.
- This reduct has the form $e'[\pi_i\langle M_1, M_2\rangle_T]$ where for all M', $e[M'] \to_{\beta\sigma} e'[M']$. As $e[M_i] \to_{\beta\sigma} e'[M_i]$, we conclude by induction hypothesis on M_1, M_2, T, and $e'[M_i]$.

On the other hand, we prove that $e[\pi_i\langle M_1, M_2\rangle_T]$ is strongly normalizing under $\to_{\beta *}$ by strong induction the sum of the maximum length of reduction from the terms M_1, M_2, T, and $e[M_i]$ respectively.

We suppose that the property holds for whenever this number is strictly below n, and we prove it when it equals n by proving that any one-step reduction of $e[\pi_i\langle M_1, M_2\rangle_T]$ is strongly normalizing under $\rightarrow_{\beta *}$. Using Lemma 6.2.1, the possible cases are the following.

- This reduct has the form $e[N]$ where $\pi_i\langle M_1, M_2\rangle_T \rightarrow_{\beta *} N$. We split this case into the following subcases.
  - $e[N] = e[\pi_i\langle M_1', M_2'\rangle_T]$ where $M_j \rightarrow_{\beta *} M_j'$ and $M_{2-j} = M_{2-j}'$ for some $j \in \{1, 2\}$. As either $e[M_i] \rightarrow_{\beta *} e[M_i']$ or $e[M_i] = e[M_i']$, we conclude in both cases by induction hypothesis on M_1', M_2', T, and $e[M_i']$.
  - $e[N] = e[\pi_i\langle M_1, M_2\rangle_{T'}]$ where $T \rightarrow_{\beta\sigma} T'$. We conclude by induction hypothesis on M_1, M_2, T', and $e[M_i]$.
  - $i = 1$ and $e[N] = e[\langle M_1, M_2\rangle_T]$. As $e[M_1]$, M_2, and T are strongly normalizing under $\rightarrow_{\beta *}$, we conclude by Lemma 6.2.2.
  - $i = 1$ and $e[N] = e[\pi_1(M_1)]$. As $e[M_1]$ is strongly normalizing under $\rightarrow_{\beta *}$, we conclude by Lemma 6.2.2.
  - $i = 2$ and $e[N] = e[\pi_2(M_1)]$. As $e[M_2]$ and M_1 are both strongly normalizing under $\rightarrow_{\beta *}$, we conclude by Lemma 6.2.2.
- This reduct has the form $e'[\pi_i\langle M_1, M_2\rangle_T]$ where for all M', $e[M'] \rightarrow_{\beta *} e'[M']$. As $e[M_i] \rightarrow_{\beta *} e'[M_i]$, we conclude by induction hypothesis with M_1, M_2, T, and $e'[M_i]$.

□

The following section is dedicated to the proof of the main properties expected from saturated sets, presented in Theorem 6.3.1.

6.3 Properties of saturated sets

Theorem 6.3.1. *The following properties hold:*

- $\mathrm{SN} \in \mathrm{SAT}$
- $S_1 \mathrel{\tilde{\rightarrow}} S_2 \in \mathrm{SAT}$ *whenever* $S_1 \in \mathrm{SAT}$ *and* $S_2 \in \mathrm{SAT}$
- $S_1 \mathrel{\tilde{\times}} S_2 \in \mathrm{SAT}$ *whenever* $S_1 \in \mathrm{SAT}$ *and* $S_2 \in \mathrm{SAT}$
- $\bigcap X \in \mathrm{SAT}$ *whenever* $X \subseteq \mathrm{SAT}$ *and* $X \neq \varnothing$

Proof. The four statements are proved separately.

- $\mathrm{SN} \in \mathrm{SAT}$ is straightforward
- Given $S_1 \in \mathrm{SAT}$ and $S_2 \in \mathrm{SAT}$, we prove $S_1 \mathrel{\tilde{\rightarrow}} S_2 \in \mathrm{SAT}$ by proving all requirements separately.

1. If $M \in S_1 \tilde{\to} S_2$, we consider v an arbitrary variable. Using the second property of saturated sets, $v \in S_1$, hence $Mv \in S_2 \subseteq \mathrm{SN}$, from which we conclude $M \in \mathrm{SN}$.

2. In order to prove $e[v] \in S_1 \tilde{\to} S_2$ whenever $e[v] \in \mathrm{SN}$, we consider $N \in S_1$ and we prove $e[v]N \in S_2$ in the following way: by definition of saturated sets, it is sufficient to prove $e[v]N \in \mathrm{SN}$, which holds by Proposition 6.2.1.

3. In order to prove $e[(\lambda v : T.M_1)M_2] \in S_1 \tilde{\to} S_2$ whenever $e[(\lambda v : T.M_1)M_2] \in \mathrm{SN}$ and $e[M_1[M_2/v]] \in S_1 \tilde{\to} S_2$, we consider $N \in S_1$ and we prove $e[(\lambda v : T.M_1)M_2]N \in S_2$ in the following way. By definition of saturated sets, it is sufficient to prove $e[M_1[M_2/v]]N \in S_2$ and $e[(\lambda v : T.M_1)M_2]N \in \mathrm{SN}$. The first requirement holds by hypothesis. On the other hand, as $e[(\lambda v : T.M_1)M_2] \in \mathrm{SN}$, $T, M_1, M_2 \in \mathrm{SN}$. As $e[M_1[M_2/v]]N \in S_2 \subseteq \mathrm{SN}$, we can apply Proposition 6.2.2 with the elimination context eN to conclude $e[(\lambda v : T.M_1)M_2]N \in \mathrm{SN}$ as expected.

4. In order to prove $e[\pi_i\langle M_1, M_2\rangle_T] \in S_1 \tilde{\to} S_2$ whenever $e[\pi_i\langle M_1, M_2\rangle_T] \in \mathrm{SN}$ and $e[M_i] \in S_1 \tilde{\to} S_2$, we consider $N \in S_1$ and we prove $e[\pi_i\langle M_1, M_2\rangle_T]N \in S_2$ in the following way. By definition of saturated sets, it is sufficient to prove $e[M_i]N \in S_2$ and $e[\pi_i\langle M_1, M_2\rangle_T]N \in \mathrm{SN}$. The first requirement holds by hypothesis. On the other hand, as $e[\pi_i\langle M_1, M_2\rangle_T] \in \mathrm{SN}$, $M_1, M_2, T \in \mathrm{SN}$. As $e[M_i]N \in S_2 \subseteq \mathrm{SN}$, we can apply Proposition 6.2.3 with the elimination context eN to conclude $e[\pi_i\langle M_1, M_2\rangle_T]N \in \mathrm{SN}$ as expected.

- Given $S_1 \in \mathrm{SAT}$ and $S_2 \in \mathrm{SAT}$, we prove $S_1 \tilde{\times} S_2 \in \mathrm{SAT}$ by proving all requirements separately.

  1. If $M \in S_1 \tilde{\times} S_2$, $\pi_1(M) \in S_1$, hence $\pi_1(M) \in \mathrm{SN}$, and $M \in \mathrm{SN}$.

  2. In order to prove $e[v] \in S_1 \tilde{\times} S_2$ whenever $e[v] \in \mathrm{SN}$, we prove $\pi_i(e[v]) \in S_2$ for all $i \in \{1, 2\}$ in the following way: by definition of saturated sets, it is sufficient to prove $\pi_i(e[v]) \in \mathrm{SN}$, which holds by Proposition 6.2.1.

  3. In order to prove $e[(\lambda v : T.M_1)M_2] \in S_1 \tilde{\times} S_2$ whenever $e[(\lambda v : T.M_1)M_2] \in \mathrm{SN}$ and $e[M_1[M_2/v]] \in S_1 \tilde{\times} S_2$, we prove $\pi_i(e[(\lambda v : T.M_1)M_2]) \in S_i$ for all $i \in \{1, 2\}$ in the following way. By definition of saturated sets, it is sufficient to prove $\pi_i(e[M_1[M_2/v]]) \in S_i$ and $\pi_i(e[(\lambda v : T.M_1)M_2]) \in \mathrm{SN}$ for all $i \in \{1, 2\}$. The first requirement holds for all $i \in \{1, 2\}$ by hypothesis. On the other hand, as $e[(\lambda v : T.M_1)M_2] \in \mathrm{SN}$, $T, M_1, M_2 \in \mathrm{SN}$. For all $i \in \{1, 2\}$, as $\pi_i(e[M_1[M_2/v]]) \in S_i \subseteq \mathrm{SN}$, we can apply Proposition 6.2.2 with the elimi-

nation context $\pi_i(e)$ to conclude $\pi_i(e[(\lambda v : T.M_1)M_2]) \in \mathrm{SN}$ as expected.

4. In order to prove $e[\pi_i\langle M_1, M_2\rangle_T] \in S_1 \tilde{\times} S_2$ whenever $e[\pi_i\langle M_1, M_2\rangle_T] \in \mathrm{SN}$ and $e[M_i] \in S_1 \tilde{\times} S_2$, we prove $\pi_j(e[\pi_i\langle M_1, M_2\rangle_T]) \in S_j$ for all $j \in \{1, 2\}$ in the following way. By definition of saturated sets, it is sufficient to prove $\pi_j(e[M_i]) \in S_j$ and $\pi_j(e[\pi_i\langle M_1, M_2\rangle_T]) \in \mathrm{SN}$ for all $j \in \{1, 2\}$. The first requirement holds for all $j \in \{1, 2\}$ by hypothesis. On the other hand, as $e[\pi_i\langle M_1, M_2\rangle_T] \in \mathrm{SN}$, $M_1, M_2, T \in \mathrm{SN}$. For all $j \in \{1, 2\}$, as $\pi_j(e[M_i]) \in S_j \subseteq \mathrm{SN}$, we can apply Proposition 6.2.3 with the elimination context $\pi_j(e)$ to conclude $\pi_j(e[\pi_i\langle M_1, M_2\rangle_T]) \in \mathrm{SN}$ as expected.

- Given $X \subseteq \mathrm{SAT}$ such that $X \neq \varnothing$, $\bigcap X \in \mathrm{SAT}$ is straightforward using the fact that a term is in $\bigcap X$ if and only if it is in S for any $S \in X$.

□

6.4 Set interpretation of terms

The next step is to interpret every inhabited term T as a saturated set – more precisely, as a family of saturated sets, because of the presence of additional parameters presented in Definitions 6.4.3 and 6.5.1, referred to *valuations*. These set interpretations will be defined recursively on T. Such a recursive definition leads to define set interpretations on some terms which are not inhabited. For instance, as $x : \mathit{Prop}, h : x \vdash h : (\lambda y : \mathit{Prop}.y)x$ is derivable, the term $(\lambda y : \mathit{Prop}.y)x$ is inhabited, while the term $\lambda y : \mathit{Prop}.y$ is not, and we will need to define set interpretations for $\lambda y : \mathit{Prop}.y$ in order to define set interpretations for $(\lambda y : \mathit{Prop}.y)x$.

This issue is shared by all extensions of System F^ω [40]. A possible solution, presented first in [40], is to define set interpretations as elements of a hierarchy of function spaces built on top of the set of saturated sets SAT. For instance, in the previous example, the set interpretations of $\lambda y : \mathit{Prop}.y$ will be expected to be functions from SAT to SAT, in order to define the set interpretations of $(\lambda y : \mathit{Prop}.y)x$ as applications of these functions to the set interpretations of x. This hierarchy of function spaces is defined as follows.

Definition 6.4.1. *We define the set hierarchy $\mathcal{U}_i$ inductively with*

- $\mathcal{U}_0 = \{\mathrm{SAT}\}$
- $\mathcal{U}_{i+1}$ *is the union of $\mathcal{U}_i$ with the set of all sets of functions from u_1 to u_2 for $u_1, u_2 \in \mathcal{U}_i$*

We define $\mathcal{U} = \bigcup\{\mathcal{U}_i \mid i \in \mathbb{N}\}$. $\mathcal{U}$ contains SAT*, the set of functions from* SAT *to* SAT*, etc.*

Whenever a term is typed by either $Kind$, $Type$, or a type, we will ensure that its set interpretations belong to a unique element of $\mathcal{U}$ defined as follows.

Definition 6.4.2 (Domains)**.** *We define the domain* $[\cdot]$ *from* $Kind$, $Type$ *and types to* $\mathcal{U}$ *as follows:*

- $[Kind] = \text{SAT}$
- $[Type] = \text{SAT}$
- $[X] = \text{SAT}$
- $[Prop] = \text{SAT}$
- $[\Pi x : A.B]$ *is the set of functions from* $[A]$ *to* $[B]$
- $[\{x : A \mid P\}] = [A]$

Remark 6.4.1. *This definition is stable under* α*-conversion.*

Remark 6.4.2. *By Theorem 5.2.3, all inhabited terms except* $Kind$ *admit some sort* s *as a type. In this case, as* $[s] = \text{SAT}$*, the associated set interpretations will be saturated sets, as expected.*

The family of set interpretations corresponding to some term is parametrized by set-valuations, which are defined as follows.

Definition 6.4.3 (Set-valuations)**.** *A set-valuation is a function from a finite set of expression variables* $x_1, ..., x_n$ *to* $\bigcup \mathcal{U}$*. If* ξ *is a set-valuation, we use the notation* $\xi, x = a$ *for the set-valuation taking the value* a *on* x *and defined as* ξ *otherwise.*

For any stratified context Γ*, we define set-valuations adapted to* Γ *as follows: a set-valuation* ξ *is adapted to* Γ *if for any declaration of the form* $(x : A)$ *in* Γ*,* ξ *is defined at* x *and* $\xi(x) \in [A]$*.*

As any well-formed context is stratified (by the stratification theorem), set-valuations adapted to well-formed contexts are special cases of this more general definition.

Remark 6.4.3. *Being adapted to a stratified context* Γ *is stable under the* α*-conversion of* Γ*.*

The next step is the definition of set interpretations themselves. Contrary to the usual definitions of set interpretations for systems without dependent pairs (e.g. the definitions of [4] for system F^ω), the set interpretations of a type A will be functions of domain $[A]$ and codomain SAT, i.e. families of saturated sets indexed by $[A]$. This idea is directly adapted from [35]. Informal justifications will be given after the definition of set interpretations.

Definition 6.4.4 (Set interpretations)**.** *For ξ a set-valuation and M either a type, an expression, $Type$, or $Kind$, we define partially the set interpretation $[\![M]\!]_\xi \in \bigcup \mathcal{U}$ recursively on M:*

- $[\![Kind]\!]_\xi = \mathrm{SN}$
- $[\![Type]\!]_\xi = \mathrm{SN}$
- *The possible cases for types are the following:*
  - $[\![X]\!]_\xi$ *is the function mapping* $a \in [X]$ *to* SN
  - $[\![Prop]\!]_\xi$ *is the function mapping* $a \in [Prop]$ *to* SN
  - $[\![\Pi x : A.B]\!]_\xi$ *is the function mapping* $f \in [\Pi x : A.B]$ *to* $\bigcap\{[\![A]\!]_\xi(a) \tilde{\to} [\![B]\!]_{\xi,x=a}(f(a)) \mid a \in [A]\}$ *whenever for all* $a \in [A]$ *and* $f \in [\Pi x : A.B]$, $[\![A]\!]_\xi(a) \in \mathrm{SAT}$ *and* $[\![B]\!]_{\xi,x=a}(f(a)) \in \mathrm{SAT}$. *Else,* $[\![\Pi x : A.B]\!]_\xi$ *is ill-defined*
  - $[\![\{x : A \mid P\}]\!]_\xi$ *is the function mapping* $a \in [\{x : A \mid P\}]$ *to* $[\![A]\!]_\xi(a) \tilde{\times} [\![P]\!]_{\xi,x=a}$ *whenever for all* $a \in [\{x : A \mid P\}]$, $[\![A]\!]_\xi(a) \in \mathrm{SAT}$ *and* $[\![P]\!]_{\xi,x=a} \in \mathrm{SAT}$. *Else,* $[\![\{x : A \mid P\}]\!]_\xi$ *is ill-defined*
- *The possible cases for expressions are the following:*
  - $[\![x]\!]_\xi = \xi(x)$ *whenever* ξ *is defined at* x
  - $[\![\Pi h : P.Q]\!]_\xi = [\![P]\!]_\xi \tilde{\to} [\![Q]\!]_\xi$ *whenever* $[\![P]\!]_\xi \in \mathrm{SAT}$ *and* $[\![Q]\!]_\xi \in \mathrm{SAT}$
  - $[\![\Pi x : A.P]\!]_\xi = \bigcap\{[\![A]\!]_\xi(a) \tilde{\to} [\![P]\!]_{\xi,x=a} \mid a \in [A]\}$ *whenever for all* $a \in [A]$, $[\![A]\!]_\xi(a) \in \mathrm{SAT}$ *and* $[\![P]\!]_{\xi,x=a} \in \mathrm{SAT}$
  - $[\![\lambda x : A.t]\!]_\xi$ *is the function mapping* $a \in [A]$ *to* $[\![t]\!]_{\xi,x=a}$ *whenever there exists* $u \in \mathcal{U}$ *such that for any* $a \in [A]$, $[\![t]\!]_{\xi,x=a} \in u$
  - $[\![t\ u]\!]_\xi = [\![t]\!]_\xi([\![u]\!]_\xi)$ *whenever* $[\![t]\!]_\xi$ *is a function defined at* $[\![u]\!]_\xi$
  - $[\![\langle t, M\rangle_A]\!]_\xi = [\![t]\!]_\xi$
  - $[\![\pi_1(t)]\!]_\xi = [\![t]\!]_\xi$

Remark 6.4.4. *The definition of set interpretations are partial, hence the proof of some proposition $[\![M]\!]_\xi \in S$ or $[\![M]\!]_\xi(a) \in S$ must contain the proof that $[\![M]\!]_\xi$ is well-defined.*

Remark 6.4.5. *For any set-valuation ξ, $[\![\cdot]\!]_\xi$ is stable under α-conversion.*

As mentioned before, the most important technical specificity in this definition is the fact that the interpretation of a type A is not a saturated set but a function mapping the elements of $[A]$ to saturated sets, i.e. a family of saturated sets indexed by $[A]$. This particularity distinguishes the case of types from the cases of $Kind$, $Type$, and expressions,

which are interpreted in a more usual way. It makes the overall definition more complex than the definitions which can be found in more usual proofs of strong normalization (such as the proofs presented in [4]). This choice is due to the presence of dependent pairs in PVS-Cert: if types were interpreted directly as saturated sets, it would have been possible to define $[\![\Pi x : A.B]\!]_\xi$ directly as the set of terms $[\![A]\!]_\xi \tilde{\to} \bigcap\{[\![B]\!]_{\xi,x=a} \mid a \in [A]\}$, but there would have been no simple definition for $[\![\{x : A \mid P\}]\!]_\xi$. This problem is identified and solved in the case of an extension of the calculus of constructions with dependent pairs in [35], introducing this idea of defining the interpretation of a type A as a family of saturated sets indexed by $[A]$. The present definition follows the same workaround, adapted to PVS-Cert.

This difficulty related to the presence of dependent pairs can be explained informally as follows. In most proofs of strong normalization based on the interpretation of inhabited terms as sets of strongly normalizing terms such as saturated sets, the main step – Theorem 6.5.1 in the present work – is the proof of the fact that the inhabitant of some term always belongs to its interpretation. In order to prove this property by induction on derivations, the interpretation of $\Pi x : A.B$ (resp. $\{x : A \mid P\}$) as a saturated set has to be sufficiently small to conclude by induction hypothesis in the case of an application (resp. a projection), and sufficiently large to conclude by induction hypothesis in the case of a λ-abstraction (resp. a dependent pair). In the case of a type $\Pi x : A.B$, there is some flexibility between these two constraints in the sense that we need a set larger than or equal to $[\![A]\!]_\xi \tilde{\to} \bigcap\{[\![B]\!]_{\xi,x=a} \mid a \in [A]\}$ to conclude in the case of an abstraction, but yet a set smaller than or equal to $[\![A]\!]_\xi \tilde{\to} [\![B]\!]_{\xi,x=a}$ for some appropriate $a \in [A]$ in the case of an application. Hence, we can use the definition $[\![\Pi x : A.B]\!]_\xi = [\![A]\!]_\xi \tilde{\to} \bigcap\{[\![B]\!]_{\xi,x=a} \mid a \in [A]\}$ to satisfy these two constraints. However, in the case of a dependent pair, this flexibility is lost in the sense that there is no simple way to obtain a saturated set satisfying these two constraints starting from the saturated set $[\![A]\!]_\xi$ and the family of saturated sets $[\![P]\!]_{\xi,x=a}$ for $a \in [A]$ and applying simple operators such as $\tilde{\times}$. In particular, the set $[\![\{x : A \mid P\}]\!]_\xi = [\![A]\!]_\xi \tilde{\times} \bigcap\{[\![P]\!]_{\xi,x=a} \mid a \in [A]\}$ is too small to conclude in the case of a pair, and the set $[\![\{x : A \mid P\}]\!]_\xi = [\![A]\!]_\xi \tilde{\times} \bigcup\{[\![P]\!]_{\xi,x=a} \mid a \in [A]\}$ is too large in the case of an projection. The set $\{M \in \mathcal{T} \mid \pi_1(M) \in [\![A]\!]_\xi \wedge [\![\pi_1(M)]\!]_\xi \text{ is well-defined} \wedge \pi_2(M) \in [\![P]\!]_{\xi,x=[\![\pi_1(M)]\!]_\xi}\}$ would appear as better suited to satisfy these two constraints, but as this complex set operator involves the definition of the interpretation $[\![\cdot]\!]$ itself, using it instead of the simple operator $\tilde{\times}$ would lead to severe complications. Instead, the current definition allows to use the much simpler definition $[\![A]\!]_\xi(a) \tilde{\times} [\![P]\!]_{\xi,x=a}$ by adding the extra parameter $a \in [A]$ as an index.

The most basic property expected from set interpretations is to be well-defined on well-typed terms and to belong to some appropriate domain. It is presented and proved in Theorem 6.4.1. Its proof uses the following lemma.

Lemma 6.4.1.

- *For any type A, any expression variable x, and any expression t, $[A[t/x]] = [A]$*

- *For any types* $A \equiv_{\beta *} B$, $[A] = [B]$.

Proof. The first proof is straightforward by induction on A. The second proof is straightforward by induction on A, using Theorem 5.3.2. □

We will also need the existence of default values in all sets $[A]$. Such default values can be defined as follows.

Definition 6.4.5 (Default values)**.** *We define default values in* $[A]$ *recursively:*

- *For* $[Kind]$, $[Type]$, $[X]$, *and* $[Prop]$, *the default value is* SN
- *For* $[\Pi x : A.B]$, *the default value is the constant function mapping any* $a \in [A]$ *to the default value of* $[B]$
- *For* $[\{x : A \mid P\}]$, *the default value is the default value of* $[A]$

We state and prove the following basic property expected for set interpretations.

Theorem 6.4.1. *For any derivable judgement* $\Gamma \vdash M : T$, *for any set-valuation* ξ *adapted to* Γ, *the following statements hold.*

- *If* T *is either* $Kind$ *or a type,* $[\![M]\!]_\xi \in [T]$.
- *If* T *is* $Type$, $[\![M]\!]_\xi$ *is a function from* $[M]$ *to* $[Type]$.

Proof. The proof is done by induction on the derivation.

- The cases Empty and Decl cannot occur.
- The case Sort is straightforward for both axioms.
- The case Var is proved in the following way. Discarding the notations in the original statement, and using the stratification theorem, a proof is needed only when the last inference step matches one of the following instances:
  - $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ Var $(x : A) \in \Gamma$
  - $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ Var $(X : Type) \in \Gamma$

  The first case follows by definition of an adapted set-valuation, and the second one is straightforward.
- The case Prod is proved as follows. Discarding the notations in the original statement, we can suppose that the last inference step matches one of the following instances:

– $\dfrac{\Gamma \vdash P : Prop \qquad \Gamma, h : P \vdash Q : Prop}{\Gamma \vdash \Pi h : P.Q : Prop}$ PROD

By induction hypothesis, $[\![P]\!]_\xi \in [Prop] = \mathrm{SAT}$ and, as ξ is also adapted to $\Gamma, h : P$, $[\![Q]\!]_\xi \in [Prop] = \mathrm{SAT}$. Hence $[\![\Pi h : P.Q]\!]_\xi$ is well-defined and belongs to $\mathrm{SAT} = [Prop]$.

– $\dfrac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \Pi x : A.P : Prop}$ PROD

Let $a \in [A]$. By induction hypothesis, $[\![A]\!]_\xi(a) \in [Type] = \mathrm{SAT}$. By the free variable theorem, $\xi, x = a$ is adapted to $\Gamma, x : A$. Therefore, by induction hypothesis, $[\![P]\!]_{\xi,x=a} \in [Prop] = \mathrm{SAT}$. $[A]$ is not empty as it admits a default value, hence $[\![\Pi x : A.P]\!]_\xi$ is well-defined and belongs to $\mathrm{SAT} = [Prop]$.

– $\dfrac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}$ PROD

Let $f \in [\Pi x : A.B]$ and $a \in [A]$. f is a function from $[A]$ to $[B]$. By induction hypothesis, $[\![A]\!]_\xi(a) \in [Type] = \mathrm{SAT}$. By the free variable theorem, $\xi, x = a$ is adapted to $\Gamma, x : A$. Therefore, by induction hypothesis, $[\![B]\!]_{\xi,x=a}(f(a)) \in [Type] = \mathrm{SAT}$. $[A]$ is not empty as it admits a default value, hence $[\![\Pi x : A.B]\!]_\xi$ is well-defined and belongs to the set of functions from $[\Pi x : A.B]$ to $\mathrm{SAT} = [Type]$.

- The case LAM is proved in the following way. Discarding the notations in the original statement, and using the stratification theorem, a proof is needed only when the last inference step matches an instance of the form

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x : A.B : Type}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B} \text{ LAM}$$

Let $a \in [A]$. By the free variable theorem, $\xi, x = a$ is adapted to $\Gamma, x : A$. Therefore, by induction hypothesis, $[\![t]\!]_{\xi,x=a} \in [B]$. Hence, $[\![\lambda x : A.t]\!]_\xi$ is well-defined and belongs to $[\Pi x : A.B]$.

- The case APP is proved in the following way. Discarding the notations in the original statement, and using the stratification theorem, a proof is needed only when the last inference step matches an instance of the form

$$\frac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]} \text{ APP}$$

By induction hypothesis, $[\![t]\!]_\xi$ is a function from $[A]$ to $[B]$ and $[\![u]\!]_\xi \in [A]$, hence $[\![t\ u]\!]_\xi \in [B]$ is well-defined and belongs to $[B]$. By Lemma 6.4.1, hence $[\![t\ u]\!]_\xi \in [B[u/x]]$.

- The case SUBTYPE is proved in the following way. Discarding the notations in the original statement, and using the stratification theorem, the last inference step matches some instance

$$\frac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \{x : A \mid P\} : Type}\ \textsc{Subtype}$$

  Let $a \in [A]$. By induction hypothesis, $[\![A]\!]_\xi(a) \in [Type] = \mathrm{SAT}$. By the free variable theorem, $\xi, x = a$ is adapted to $\Gamma, x : A$. Therefore, by induction hypothesis, $[\![P]\!]_{\xi,x=a} \in [Prop] = \mathrm{SAT}$. Hence $[\![\{x : A \mid P\}]\!]_\xi$ is well-defined and belongs to the set of functions from $[A]$ to $\mathrm{SAT} = [Type]$.

- The case PAIR is proved in the following way. Discarding the notations in the original statement, and using the stratification theorem, the last inference step matches some instance

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash M : P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : Type}{\Gamma \vdash \langle t, M\rangle_{\{x:A|P\}} : \{x : A \mid P\}}\ \textsc{Pair}$$

  By induction hypothesis, $[\![t]\!]_\xi \in [A] = [\{x : A \mid P\}]$, hence $[\![\langle t, M\rangle_{\{x:A|P\}}]\!]_\xi$ is well-defined and belongs to $[\{x : A \mid P\}]$.

- The case PROJ1 is proved in the following way. Discarding the notations in the original statement, and using the stratification theorem, the last inference step matches some instance

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \pi_1(t) : A}\ \textsc{Proj1}$$

  By induction hypothesis, $[\![t]\!]_\xi \in [\{x : A \mid P\}] = [A]$, hence $[\![\pi_1(t)]\!]_\xi$ is well-defined and belongs to $[A]$.

- In the case PROJ2 by the stratification theorem, there is nothing to prove.

- The case CONVERSION is straightforward by induction hypothesis using the stratification theorem and Lemma 6.4.1.

□

We end the current section with the following lemmas on set interpretations.

Lemma 6.4.2. *The following statements hold.*

- *If $[\![u]\!]_\xi$ and $[\![M]\!]_{\xi,x=[\![u]\!]_\xi}$ are well-defined, so is $[\![M[u/x]]\!]_\xi$, and $[\![M[u/x]]\!]_\xi = [\![M]\!]_{\xi,x=[\![u]\!]_\xi}$.*
- *If $[\![M]\!]_\xi$ is well-defined, so is $[\![M[N/h]]\!]_\xi$ for any term N and any proof variable h, and $[\![M[N/h]]\!]_\xi = [\![M]\!]_\xi$.*

- *If* $M_1 \equiv_{\beta *} M_2$ *and* $[\![M_i]\!]_\xi$ *is well-defined for all* $i \in \{1, 2\}$, $[\![M_1]\!]_\xi = [\![M_2]\!]_\xi$.

Proof. The proof of the first statement is done as follows. We can suppose that M is either $Kind$, $Type$, a type, or an expression, as set interpretations are not defined for other terms. The cases $Kind$ and $Type$ are straightforward. We prove the two last cases together by induction on M. As the set interpretations and substitution are both stable under α-conversion, we can suppose without loss of generality that no bound variable in M is free in u nor equal to x. The cases are the following:

- The cases X and $Prop$ are straightforward.
- The cases $\Pi y : A.B$, $\{y : A \mid P\}$, $\Pi h : P.Q$, $\Pi y : A.P$, and $\lambda y : A.t$ are similar. We present the case $\Pi y : A.P$ as an example.

  Let $a \in [A]$. As $[\![\Pi y : A.P]\!]_{\xi, x=[\![u]\!]_\xi}$ is well-defined, $[\![A]\!]_{\xi, x=[\![u]\!]_\xi}$ and $[\![P]\!]_{\xi, x=[\![u]\!]_\xi, y=a}$ are well-defined as well. As $y \neq x$ by hypothesis, $[\![P]\!]_{\xi, x=[\![u]\!]_\xi, y=a} = [\![P]\!]_{\xi, y=a, x=[\![u]\!]_\xi}$. By induction hypothesis, $[\![A[u/x]]\!]_\xi$ and $[\![P[u/x]]\!]_{\xi, y=a}$ are well-defined and equal to $[\![A]\!]_{\xi, x=[\![u]\!]_\xi}$ and $[\![P]\!]_{\xi, x=[\![u]\!]_\xi, y=a}$ respectively. By Lemma 6.4.1, $[A] = [A[u/x]]$, hence $a \in [A]$ if and only if $a \in [A[u/x]]$. Therefore, $[\![(\Pi y : A.P)[u/x]]\!]_\xi$ is well-defined and equal to $[\![M]\!]_{\xi, x=[\![u]\!]_\xi}$ as expected.
- In the case x, $[\![t[u/x]]\!]_\xi = [\![u]\!]_\xi$ is well-defined and equal to $[\![x]\!]_{\xi, x=[\![u]\!]_\xi}$.
- In the case $y \neq x$, as $[\![y]\!]_{\xi, x=[\![u]\!]_\xi}$ is well-defined and $[\![y]\!]_{\xi, x=[\![u]\!]_\xi} = \xi(y)$, hence $[\![t[u/x]]\!]_\xi = [\![y]\!]_\xi$ is well-defined too equal to $[\![y]\!]_{\xi, x=[\![u]\!]_\xi}$.
- The other cases $(t_1\ t_2)$, $\langle t, M' \rangle_T$, and $\pi_1(t)$ are straightforward by induction hypothesis.

  The second statement is proved in a similar way. All cases are straightforward.

The last statement is proved as follows. We first prove that if $M_1 \to_{\beta *} M_2$ and $[\![M_1]\!]_\xi$ is well-defined, then so is $[\![M_2]\!]_\xi$, and $[\![M_1]\!]_\xi = [\![M_2]\!]_\xi$. As M_1 is reducible and $[\![M_1]\!]_\xi$ is well-defined, M_1 is either a type or an expression. We prove the expected for these two cases together by induction on M_1. All cases except the case of an application are straightforward by induction hypothesis and Lemma 6.4.1. If M_1 is an application, we split the subcases $M_1 \not\triangleright_{\beta *} M_2$ and $M_1 \triangleright_{\beta *} M_2$. The subcase $M_1 \not\triangleright_{\beta *} M_2$ is straightforward by induction hypothesis. If $M_1 \triangleright_{\beta *} M_2$, then M_1 has the form $(\lambda x : A.t)u$ and $M_2 = t[u/x]$. As $[\![(\lambda x : A.t)u]\!]_\xi$ is well-defined, $[\![(\lambda x : A.t)u]\!]_\xi = [\![\lambda x : A.t]\!]_\xi([\![u]\!]_\xi) = [\![t]\!]_{\xi, x=[\![u]\!]_\xi}$, hence we can conclude using the first statement.

Then, we prove by induction on the length of reductions that if $M_1 \twoheadrightarrow_{\beta *} M_2$ and $[\![M_1]\!]_\xi$, then so is $[\![M_2]\!]_\xi$, and $[\![M_1]\!]_\xi = [\![M_2]\!]_\xi$. Finally, we conclude the expected result by the Church-Rosser theorem. □

Lemma 6.4.3. *If* $[\![M]\!]_\xi$ *is well-defined and* x *is not free in* M*, then for any* $a \in \bigcup \mathcal{U}$, $[\![M]\!]_{\xi, x=a}$ *is well-defined and* $[\![M]\!]_{\xi, x=a} = [\![M]\!]_\xi$.

Proof. The proof is straightforward by induction on M. □

6.5 Proof of strong normalization

The induction hypothesis used in the main proof of strong normalization is based parametrized not only by set-valuations, but also by term-valuations, which are defined as follows.

Definition 6.5.1 (Term-valuations)**.** *A term-valuation is a function from a finite set of variables to $\mathcal{T}$. For ρ a term-valuation, we use the notation $\rho, v = M$ for the set-valuation taking the value M on v and defined as ρ otherwise. We define $FV(\rho)$ as the set of free variables appearing in the range of ρ.*

For any stratified context Γ and any set-valuation ξ, we define term-valuations adapted to Γ and ξ as follows: a term-valuation ρ is adapted to Γ and ξ if

- *ξ is adapted to Γ*
- *for any $(x : A) \in \Gamma$, ρ is defined at x and $\rho(x) \in [\![A]\!]_\xi(\xi(x))$*
- *for any $(h : P) \in \Gamma$, ρ is defined at h and $\rho(h) \in [\![P]\!]_\xi$*

As any well-formed context is stratified (by the stratification theorem), term-valuations adapted to well-formed contexts and set-valuations are special cases of this more general definition.

The following lemma about adapted term-valuations will be useful.

Lemma 6.5.1. *For any well-formed context Γ, for any term-valuation ρ is adapted to Γ and to some set-valuation ξ, the following statement holds.*

- *For any well-formed extension of the form $\Gamma, h : P$, if $[\![P]\!]_\xi \in \mathrm{SAT}$ and $M \in [\![P]\!]_\xi$, then $\rho, h = M$ is adapted to $\Gamma, h : P$ and ξ.*
- *For any well-formed extension of the form $\Gamma, x : A$, any $a \in [A]$, if $[\![A]\!]_\xi(a) \in \mathrm{SAT}$ and $M \in [\![A]\!]_\xi(a)$, then $\rho, x = M$ is adapted to $\Gamma, x : A$ and $\xi, x = a$.*

Proof. The two cases are similar. We present the second one as an example.
We consider Γ, ξ, ρ, x, A, a, and M as defined in the second statement. The proof is done as follows.

- By the free variable theorem, $\xi, x = a$ is adapted to $\Gamma, x : A$.
- For any $(h : P) \in \Gamma$, $(\rho, x = M)(h) = \rho(h) \in [\![P]\!]_\xi$. By the free variable theorem and Lemma 6.4.3, $[\![P]\!]_{\xi,x=a} = [\![P]\!]_\xi$, hence $(\rho, x = M)(h) \in [\![P]\!]_{\xi,x=a}$.
- For any $(y : B) \in \Gamma$, by the free variable theorem, $y \neq x$, hence $(\rho, x = M)(y) = \rho(y) \in [\![B]\!]_\xi(\xi(y))$. By the free variable theorem and Lemma 6.4.3, $[\![B]\!]_{\xi,x=a} = [\![B]\!]_\xi$ and $\xi(y) = (\xi, x = a)(y)$, hence $(\rho, x = M)(y) \in [\![B]\!]_{\xi,x=a}((\xi, x = a)(y))$.

- By hypothesis, $(\rho, x = M)(x) = M \in [\![A]\!]_\xi(a)$. Hence, by the free variable theorem and Lemma 6.4.3, $(\rho, x = M)(x) \in [\![A]\!]_{\xi, x=a}((\xi, x = a)(x))$.

□

The previous definitions of term-valuations is used to define a notion of term interpretations, presented in the following definition. In the main proof of strong normalization, the induction hypothesis will be formalized as the belonging of some term interpretations in some set interpretations.

Definition 6.5.2 (Term interpretations)**.** *For ρ a term-valuation and M, we define the term interpretation $(\!|M|\!)_\rho \in \mathcal{T}$. We define it recursively on M. If necessary, we first α-rename M into a term in which no bound variable is in $FV(\rho)$.*

- $(\!|s|\!)_\rho = s$
- $(\!|X|\!)_\rho = X$
- $(\!|v|\!)_\rho = \rho(v)$ *if* $v \notin \mathcal{V}_{types}$
- $(\!|\lambda v : T.M|\!)_\rho = \lambda v : (\!|T|\!)_\rho.(\!|M|\!)_{\rho, v=v}$ *if* $v \notin FV(\rho)$
- $(\!|MN|\!)_\rho = (\!|M|\!)_\rho (\!|N|\!)_\rho$
- $(\!|\Pi v : T.U|\!)_\rho = \Pi v : (\!|T|\!)_\rho.(\!|U|\!)_{\rho, v=v}$ *if* $v \notin FV(\rho)$
- $(\!|\{v : T \mid M\}|\!)_\rho = \{v : (\!|T|\!)_\rho \mid (\!|M|\!)_{\rho, v=v}\}$ *if* $v \notin FV(\rho)$
- $(\!|\langle M, N\rangle_T|\!)_\rho = \langle (\!|M|\!)_\rho, (\!|N|\!)_\rho\rangle_{(\!|T|\!)_\rho}$
- $(\!|\pi_i(M)|\!)_\rho = \pi_i((\!|M|\!)_\rho)$ *for* $i \in \{1, 2\}$

Remark 6.5.1. *For any term-valuation ρ, $(\!|\cdot|\!)_\rho$ is stable under α-conversion. Hence, the precise choice of α-renaming used in its definition won't be relevant in the following.*

The last lemma used in the main proof is the following one.

Lemma 6.5.2. *Whenever $v \notin \mathcal{V}_{types}$, $(\!|M|\!)_\rho[N/v] = (\!|M|\!)_{\rho, v=N}$*

Proof. The proof is straightforward by induction on M. □

Theorem 6.5.1 (Main theorem)**.** *For any derivable judgement $\Gamma \vdash M : T$, any set-valuation ξ and any term-valuation ρ such that ρ is adapted to Γ and ξ,*

- *if T is a type, $(\!|M|\!)_\rho \in [\![T]\!]_\xi([\![M]\!]_\xi)$*
- *else, $(\!|M|\!)_\rho \in [\![T]\!]_\xi$*

Proof. We prove the expected result by strong induction on the height of the derivation.

- The cases Empty and Decl cannot occur.

- The case SORT is straightforward for both axioms.
- The case VAR is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches one of the following instances:

  – $\dfrac{\Gamma \vdash WF}{\Gamma \vdash h : P}$ VAR $(h : P) \in \Gamma$

  In this case, $(\!|h|\!)_\rho = \rho(h) \in [\![P]\!]_\xi$ by definition of an adapted term-valuation

  – $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ VAR $(x : A) \in \Gamma$

  In this case, $(\!|x|\!)_\rho = \rho(x) \in [\![A]\!]_\xi(\xi(x)) = [\![A]\!]_\xi([\![x]\!]_\xi)$ by definition of an adapted term-valuation

  – $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ VAR $(X : Type) \in \Gamma$

  In this case, $(\!|X|\!)_\rho = X \in \mathrm{SN} = [\![Type]\!]_\xi$ as expected

- The case PROD is proved as follows. Discarding the notations in the original statement, using the stratification theorem and the renaming theorem, we can suppose that the last inference step matches one of the following instances:

  – $\dfrac{\Gamma \vdash P : Prop \qquad \Gamma, h : P \vdash Q : Prop}{\Gamma \vdash \Pi h : P.Q : Prop}$ PROD

  with $h \notin FV(\rho)$, as the original proof can be transformed to meet this requirement without changing its height. Hence, $(\!|\Pi h : P.Q|\!)_\rho = \Pi h : (\!|P|\!)_\rho.(\!|Q|\!)_{\rho,h=h}$. By Theorem 6.4.1, $[\![\Pi h : P.Q]\!]_\xi \in [Prop]$. In order to prove that $\Pi h : (\!|P|\!)_\rho.(\!|Q|\!)_{\rho,h=h} \in [\![Prop]\!]_\xi([\![\Pi h : P.Q]\!]_\xi) = \mathrm{SN}$ as expected, it is sufficient to prove that $(\!|P|\!)_\rho \in \mathrm{SN}$ and $(\!|Q|\!)_{\rho,h=h} \in \mathrm{SN}$. By induction hypothesis, the first condition holds: $(\!|P|\!)_\rho \in [\![Prop]\!]_\xi([\![P]\!]_\xi) = \mathrm{SN}$. On the other hand, we can conclude $(\!|Q|\!)_{\rho,h=h} \in [\![Prop]\!]_\xi([\![P]\!]_\xi) = \mathrm{SN}$ by induction hypothesis if we prove that $\rho, h = h$ is adapted to ξ and $\Gamma, h : P$.

  By Lemma 6.5.1, it is sufficient to prove $[\![P]\!]_\xi \in \mathrm{SAT}$ and $h \in [\![P]\!]_\xi$. By Theorem 6.4.1, $[\![P]\!]_\xi \in [Prop] = \mathrm{SAT}$, hence, using the second property of saturated sets, $h \in [\![P]\!]_\xi$.

  – $\dfrac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \Pi x : A.P : Prop}$ PROD

  with $x \notin FV(\rho)$, as the original proof can be transformed to meet this requirement without changing its height. Hence, $(\!|\Pi x : A.P|\!)_\rho = \Pi x : (\!|A|\!)_\rho.(\!|P|\!)_{\rho,x=x}$.

By Theorem 6.4.1, $[\![\Pi x : A.P]\!]_\xi \in [Prop]$. In order to prove $\Pi x : (\!|A|\!)_\rho.(\!|P|\!)_{\rho,x=x} \in [\![Prop]\!]_\xi([\![\Pi x : A.P]\!]_\xi) = \mathrm{SN}$ as expected, it is sufficient to prove that $(\!|A|\!)_\rho \in \mathrm{SN}$ and $(\!|P|\!)_{\rho,x=x} \in \mathrm{SN}$. By induction hypothesis, the first condition holds: $(\!|A|\!)_\rho \in [\![Type]\!]_\xi = \mathrm{SN}$. Taking a the default value of $[A]$, we can conclude $(\!|P|\!)_{\rho,x=x} \in [\![Prop]\!]_{\xi,x=a}([\![P]\!]_{\xi=a}) = \mathrm{SN}$ by induction hypothesis if we prove that $\rho, x = x$ is adapted to $\xi, x = a$ and $\Gamma, x : A$.

By Lemma 6.5.1, it is sufficient to prove $[\![A]\!]_\xi(a) \in \mathrm{SAT}$ and $x \in [\![A]\!]_\xi(a)$. By Theorem 6.4.1, $[\![A]\!]_\xi(a) \in [Type] = \mathrm{SAT}$, hence, using the second property of saturated sets, $x \in [\![A]\!]_\xi(a)$.

– $\dfrac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}\ \textsc{Prod}$

with $x \notin FV(\rho)$, as the original proof can be transformed to meet this requirement without changing its height. Hence, $(\!|\Pi x : A.B|\!)_\rho = \Pi x : (\!|A|\!)_\rho.(\!|B|\!)_{\rho,x=x}$. We consider f the default value of $[\Pi x : A.B]$. In order to prove $\Pi x : (\!|A|\!)_\rho.(\!|B|\!)_{\rho,x=x} \in [\![Type]\!]_\xi = \mathrm{SN}$ as expected, it is sufficient to prove that $(\!|A|\!)_\rho \in \mathrm{SN}$ and $(\!|B|\!)_{\rho,x=x} \in \mathrm{SN}$. By induction hypothesis, the first condition holds: $(\!|A|\!)_\rho \in [\![Type]\!]_\xi = \mathrm{SN}$. Taking a the default value of $[A]$, we can conclude $(\!|B|\!)_{\rho,x=x} \in [\![Type]\!]_{\xi,x=a} = \mathrm{SN}$ by induction hypothesis if we prove that $\rho, x = x$ is adapted to $\xi, x = a$ and $\Gamma, x : A$.

By Lemma 6.5.1, it is sufficient to prove $[\![A]\!]_\xi(a) \in \mathrm{SAT}$ and $x \in [\![A]\!]_\xi(a)$. By Theorem 6.4.1, $[\![A]\!]_\xi(a) \in [Type] = \mathrm{SAT}$, hence, using the second property of saturated sets, $x \in [\![A]\!]_\xi(a)$.

- The case Subtype is proved as follows. Discarding the notations in the original statement, using the stratification theorem and the renaming theorem, we can suppose that the last inference step matches some instance

$$\frac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \{x : A \mid P\} : Type}\ \textsc{Subtype}$$

with $x \notin FV(\rho)$, as the original proof can be transformed to meet this requirement without changing its height. Hence, $(\!|\{x : A \mid P\}|\!)_\rho = \{x : (\!|A|\!)_\rho \mid (\!|P|\!)_{\rho,x=x}\}$. In order to prove $\{x : (\!|A|\!)_\rho \mid (\!|P|\!)_{\rho,x=x}\} \in [\![Type]\!]_\xi = \mathrm{SN}$ as expected, it is sufficient to prove that $(\!|A|\!)_\rho \in \mathrm{SN}$ and $(\!|P|\!)_{\rho,x=x} \in \mathrm{SN}$. By induction hypothesis, the first condition holds: $(\!|A|\!)_\rho \in [\![Type]\!]_\xi = \mathrm{SN}$. Taking a the default value of $[A]$, we can conclude $(\!|P|\!)_{\rho,x=x} \in [\![Prop]\!]_{\xi,x=a}([\![P]\!]_{\xi=a}) = \mathrm{SN}$ by induction hypothesis if we prove that $\rho, x = x$ is adapted to $\xi, x = a$ and $\Gamma, x : A$.

By Lemma 6.5.1, it is sufficient to prove $[\![A]\!]_\xi(a) \in \mathrm{SAT}$ and $x \in [\![A]\!]_\xi(a)$. By

Theorem 6.4.1, $[\![A]\!]_\xi(a) \in [Type] = \text{SAT}$, hence, using the second property of saturated sets, $x \in [\![A]\!]_\xi(a)$.

- The case Lam is proved in the following way. Discarding the notations in the original statement, using the renaming theorem and the stratification theorem, we can suppose that the last inference step matches one of the following instances:

$$- \ \frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x : A.B : Type}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B} \text{ Lam}$$

with $x \notin FV(\rho)$, as the original proof can be transformed to meet this requirement without changing its height. Hence, $(\!|\lambda x : A.t|\!)_\rho = \lambda x : (\!|A|\!)_\rho.(\!|t|\!)_{\rho,x=x}$. By Theorem 6.4.1 applied to the conclusion and to the second premise, $[\![\Pi x : A.B]\!]_\xi([\![\lambda x : A.t]\!]_\xi) \in [Type] = \text{SAT}$. In particular, $[\![\Pi x : A.B]\!]_\xi([\![\lambda x : A.t]\!]_\xi)$ is well-defined: therefore, for all $a \in [A]$, $[\![A]\!]_\xi(a) \in \text{SAT}$ and $[\![B]\!]_{\xi,x=a}([\![\lambda x : A.t]\!]_\xi(a)) \in \text{SAT}$. In order to prove $\lambda x : (\!|A|\!)_\rho.(\!|t|\!)_{\rho,x=x} \in [\![\Pi x : A.B]\!]_\xi([\![\lambda x : A.t]\!]_\xi)$ as expected, we consider $a \in [A]$ and $N \in [\![A]\!]_\xi(a)$, and we prove $(\lambda x : (\!|A|\!)_\rho.(\!|t|\!)_{\rho,x=x})N \in [\![B]\!]_{\xi,x=a}([\![\lambda x : A.t]\!]_\xi(a))$ in the following way.

By Lemma 6.5.1, $\rho, x = N$ is adapted to $\Gamma, x : A$ and $\xi, x = a$. Hence, by induction hypothesis and Lemma 6.5.2, $(\!|t|\!)_{\rho,x=x}[N/x] \in [\![B]\!]_{\xi,x=a}([\![t]\!]_{\xi,x=a})$. As $[\![\lambda x : A.t]\!]_\xi(a) = [\![t]\!]_{\xi,x=a}$, we can conclude the expected result using the third property of saturated sets if we prove $(\lambda x : (\!|A|\!)_\rho.(\!|t|\!)_{\rho,x=x})N \in \text{SN}$. This is done as follows.

By Proposition 6.2.2, it is sufficient to prove that $(\!|A|\!)_\rho$, $(\!|t|\!)_{\rho,x=x}[N/x]$, and N belong to SN. The two last requirements hold as $(\!|t|\!)_{\rho,x=x}[N/x]$ and N belong to saturated sets. On the other hand, by the subderivations theorem, as $s(x) = Type$, the second premise admits a subderivation of conclusion $\Gamma \vdash A : Type$. Hence, by strong induction hypothesis, $(\!|A|\!)_\rho \in [\![Type]\!]_\xi = \text{SN}$ as expected.

$$- \ \frac{\Gamma, x : A \vdash M : P \qquad \Gamma \vdash \Pi x : A.P : Prop}{\Gamma \vdash \lambda x : A.M : \Pi x : A.P} \text{ Lam}$$

with $x \notin FV(\rho)$, as the original proof can be transformed to meet this requirement without changing its height. Hence, $(\!|\lambda x : A.M|\!)_\rho = \lambda x : (\!|A|\!)_\rho.(\!|M|\!)_{\rho,x=x}$. By Theorem 6.4.1 applied to the second premise, $[\![\Pi x : A.P]\!]_\xi \in [Prop] = \text{SAT}$. In particular, $[\![\Pi x : A.P]\!]_\xi$ is well-defined: therefore, for all $a \in [A]$, $[\![A]\!]_\xi(a) \in \text{SAT}$ and $[\![P]\!]_{\xi,x=a} \in \text{SAT}$. In order to prove $\lambda x : (\!|A|\!)_\rho.(\!|M|\!)_{\rho,x=x} \in [\![\Pi x : A.P]\!]_\xi$ as expected, we consider $a \in [A]$ and $N \in [\![A]\!]_\xi(a)$, and we prove $(\lambda x : (\!|A|\!)_\rho.(\!|M|\!)_{\rho,x=x})N \in [\![P]\!]_{\xi,x=a}$ in the following way.

By Lemma 6.5.1, $\rho, x = N$ is adapted to $\Gamma, x : A$ and $\xi, x = a$. Hence, by induction hypothesis and Lemma 6.5.2, $(\!|M|\!)_{\rho,x=x}[N/x] \in [\![P]\!]_{\xi,x=a}$. We can conclude the expected result using the third property of saturated sets if we prove $(\lambda x : (\!|A|\!)_\rho.(\!|M|\!)_{\rho,x=x})N \in \mathrm{SN}$. This is done as follows.

By Proposition 6.2.2, it is sufficient to prove that $(\!|A|\!)_\rho$, $(\!|M|\!)_{\rho,x=x}[N/x]$, and N belong to SN. The two last requirements hold as $(\!|M|\!)_{\rho,x=x}[N/x]$ and N belong to saturated sets. On the other hand, by the subderivations theorem, as $s(x) = \mathit{Type}$, the second premise admits a subderivation of conclusion $\Gamma \vdash A : \mathit{Type}$. Hence, by strong induction hypothesis, $(\!|A|\!)_\rho \in [\![\mathit{Type}]\!]_\xi = \mathrm{SN}$.

$$_\ \frac{\Gamma, h : P \vdash M : Q \qquad \Gamma \vdash \Pi h : P.Q : \mathit{Prop}}{\Gamma \vdash \lambda h : P.M : \Pi h : P.Q}\ \textsc{Lam}$$

with $h \notin FV(\rho)$, as the original proof can be transformed to meet this requirement without changing its height. Hence, $(\!|\lambda h : P.M|\!)_\rho = \lambda h : (\!|P|\!)_\rho.(\!|M|\!)_{\rho,h=h}$. By Theorem 6.4.1 applied to the second premise, $[\![\Pi h : P.Q]\!]_\xi \in [\mathit{Prop}] = \mathrm{SAT}$. In particular, $[\![\Pi h : P.Q]\!]_\xi$ is well-defined: therefore, $[\![P]\!]_\xi \in \mathrm{SAT}$ and $[\![Q]\!]_\xi \in \mathrm{SAT}$. In order to prove $\lambda h : (\!|P|\!)_\rho.(\!|M|\!)_{\rho,h=h} \in [\![\Pi h : P.Q]\!]_\xi$ as expected, we consider $N \in [\![P]\!]_\xi$, and we prove $(\lambda h : (\!|P|\!)_\rho.(\!|M|\!)_{\rho,h=h})N \in [\![Q]\!]_\xi$ in the following way.

By Lemma 6.5.1, $\rho, h = N$ is adapted to $\Gamma, h : P$ and ξ. Hence, by induction hypothesis and Lemma 6.5.2, $(\!|M|\!)_{\rho,h=h}[N/h] \in [\![Q]\!]_\xi$. We can conclude the expected result using the third property of saturated sets if we prove $(\lambda h : (\!|P|\!)_\rho.(\!|M|\!)_{\rho,h=h})N \in \mathrm{SN}$. This is done as follows.

By Proposition 6.2.2, it is sufficient to prove that $(\!|P|\!)_\rho$, $(\!|M|\!)_{\rho,h=h}[N/h]$, and N belong to SN. The two last requirements hold as $(\!|M|\!)_{\rho,h=h}[N/h]$ and N belong to saturated sets. On the other hand, by the subderivations theorem, as $s(h) = \mathit{Prop}$, the second premise admits a subderivation of conclusion $\Gamma \vdash P : \mathit{Prop}$. Hence, by strong induction hypothesis, $(\!|P|\!)_\rho \in [\![\mathit{Prop}]\!]_\xi((\!|P|\!)_\xi) = \mathrm{SN}$.

- The case App is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches one of the following instances.

$$_\ \frac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}\ \textsc{App}$$

By induction hypothesis, $(\!|t|\!)_\rho \in [\![\Pi x : A.B]\!]_\xi([\![t]\!]_\xi)$ and by Theorem 6.4.1, $[\![u]\!]_\xi \in [A]$. Hence, $(\!|t|\!)_\rho \in [\![A]\!]_\xi([\![u]\!]_\xi) \tilde{\to} [\![B]\!]_{\xi,x=[\![u]\!]_\xi}([\![t]\!]_\xi([\![u]\!]_\xi))$. By induction hypothesis, $(\!|u|\!)_\rho \in [\![A]\!]_\xi([\![u]\!]_\xi)$. Therefore, $(\!|t|\!)_\rho(\!|u|\!)_\rho \in [\![B]\!]_{\xi,x=[\![u]\!]_\xi}([\![t]\!]_\xi([\![u]\!]_\xi))$,

which rewrites to $(\!|tu|\!)_\rho \in [\![B]\!]_{\xi,x=[\![u]\!]_\xi}([\![tu]\!]_\xi)$. Thus, by Lemma 6.4.2, $(\!|tu|\!)_\rho \in [\![B[u/x]]\!]_\xi([\![tu]\!]_\xi)$ as expected.

– $\dfrac{\Gamma \vdash M : \Pi x : A.P \qquad \Gamma \vdash t : A}{\Gamma \vdash Mt : P[t/x]}$ App

By induction hypothesis, $(\!|M|\!)_\rho \in [\![\Pi x : A.P]\!]_\xi$ and by Theorem 6.4.1, $[\![t]\!]_\xi \in [A]$. Hence, $(\!|M|\!)_\rho \in [\![A]\!]_\xi([\![t]\!]_\xi) \tilde{\to} [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$. By induction hypothesis, $(\!|t|\!)_\rho \in [\![A]\!]_\xi([\![t]\!]_\xi)$. Therefore, $(\!|Mt|\!)_\rho = (\!|M|\!)_\rho(\!|t|\!)_\rho \in [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$. Thus, by Lemma 6.4.2, $(\!|Mt|\!)_\rho \in [\![P[t/x]]\!]_\xi$ as expected.

– $\dfrac{\Gamma \vdash M : \Pi h : P.Q \qquad \Gamma \vdash N : P}{\Gamma \vdash MN : Q[N/h]}$ App

By induction hypothesis, $(\!|M|\!)_\rho \in [\![\Pi h : P.Q]\!]_\xi = [\![P]\!]_\xi \tilde{\to} [\![Q]\!]_\xi$ and $(\!|N|\!)_\rho \in [\![P]\!]_\xi$. Hence, $(\!|MN|\!)_\rho = (\!|M|\!)_\rho(\!|N|\!)_\rho \in [\![Q]\!]_\xi$. Thus, by Lemma 6.4.2, $(\!|MN|\!)_\rho \in [\![Q[N/h]]\!]_\xi$ as expected.

- The case Pair is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches the instance

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash M : P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : Type}{\Gamma \vdash \langle t, M\rangle_{\{x:A|P\}} : \{x : A \mid P\}}\ \textsc{Pair}$$

We define $B = \{x : A \mid P\}$. By Theorem 6.4.1 applied to the conclusion and to the second premise, $[\![\langle t, M\rangle_B]\!]_\xi \in [\{x : A \mid P\}] = [A]$ and $[\![\{x : A \mid P\}]\!]_\xi([\![\langle t, M\rangle_B]\!]_\xi) \in [Type] = \mathrm{SAT}$. In particular, $[\![\{x : A \mid P\}]\!]_\xi([\![\langle t, M\rangle_B]\!]_\xi)$ is well-defined: therefore, $[\![A]\!]_\xi([\![\langle t, M\rangle_B]\!]_\xi) = [\![A]\!]_\xi([\![t]\!]_\xi)$ and $[\![P]\!]_{\xi,x=[\![\langle t,M\rangle_B]\!]_\xi} = [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$ are well-defined and belong to SAT. In order to prove $\langle(\!|t|\!)_\rho, (\!|M|\!)_\rho\rangle_{(\!|B|\!)_\rho} \in [\![\{x : A \mid P\}]\!]_\xi([\![\langle t, M\rangle_B]\!]_\xi)$ as expected, we prove $\pi_1\langle(\!|t|\!)_\rho, (\!|M|\!)_\rho\rangle_{(\!|B|\!)_\rho} \in [\![A]\!]_\xi([\![t]\!]_\xi)$ and $\pi_2\langle(\!|t|\!)_\rho, (\!|M|\!)_\rho\rangle_{(\!|B|\!)_\rho} \in [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$ in the following way.

By induction hypothesis and Lemma 6.4.2, $(\!|t|\!)_\rho \in [\![A]\!]_\xi([\![t]\!]_\xi)$ and $(\!|M|\!)_\rho \in [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$. Hence, we can conclude the two expected results using the fourth property of saturated sets if we prove $\pi_i\langle(\!|t|\!)_\rho, (\!|M|\!)_\rho\rangle_{(\!|B|\!)_\rho} \in \mathrm{SN}$ for all $i \in \{1, 2\}$. This is done as follows.

By Proposition 6.2.3, $\pi_i\langle(\!|t|\!)_\rho, (\!|M|\!)_\rho\rangle_{(\!|B|\!)_\rho} \in \mathrm{SN}$ is ensured for all $i \in \{1, 2\}$ if $(\!|t|\!)_\rho$, $(\!|M|\!)_\rho$, and $(\!|B|\!)_\rho$ belong to SN. This is the case for $(\!|t|\!)_\rho$ and $(\!|M|\!)_\rho$ as these terms belong to saturated sets. This is also the case for $(\!|B|\!)_\rho$ as, by induction hypothesis, $(\!|B|\!)_\rho \in [\![Type]\!]_\xi = \mathrm{SN}$.

- The case Proj1 is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches the instance

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \pi_1(t) : A}\ \textsc{Proj1}$$

  By induction hypothesis, $(\!|t|\!)_\rho \in [\![\{x : A \mid P\}]\!]_\xi([\![t]\!]_\xi) = [\![A]\!]_\xi([\![t]\!]_\xi) \tilde{\times} [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$. Therefore, $(\!|\pi_1(t)|\!)_\rho = \pi_1((\!|t|\!)_\rho) \in [\![A]\!]_\xi([\![t]\!]_\xi)$ as expected.

- The case Proj2 is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches the instance

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \pi_2(t) : P[\pi_1(t)/x]}\ \textsc{Proj2}$$

  By induction hypothesis, $(\!|t|\!)_\rho \in [\![\{x : A \mid P\}]\!]_\xi([\![t]\!]_\xi) = [\![A]\!]_\xi([\![t]\!]_\xi) \tilde{\times} [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$. Therefore, $\pi_2((\!|t|\!)_\rho) \in [\![P]\!]_{\xi,x=[\![t]\!]_\xi}$, which rewrites to $\pi_2((\!|t|\!)_\rho) \in [\![P]\!]_{\xi,x=[\![\pi_1(t)]\!]_\xi}$. Thus, by Lemma 6.4.2, $\pi_2((\!|t|\!)_\rho) \in [\![P[\pi_1(t)/x]]\!]_\xi$ as expected.

- The case Conversion is straightforward, applying the induction hypothesis on the first premise, Theorem 6.4.1 on the second one, and Lemma 6.4.2 to conclude.

□

Theorem 6.5.2 (Strong normalization)**.** *For any derivable judgement $\Gamma \vdash M : T$, the two terms M and T belong to* SN*.*

Proof. We first prove that for any derivable judgement $\Gamma \vdash M : T$, M belongs to SN as follows. We consider ξ any set-valuation such that for all $(x : A) \in \Gamma$, $\xi(x)$ is the default value of $[A]$. In this setting, ξ is adapted to Γ. Then, we consider ρ a term-valuation mapping any variable $v \in DV(\Gamma)$ to itself. We prove that ρ is adapted to Γ and ξ in the following way.

- For any decomposition $\Gamma = \Gamma_1, x : A, \Gamma_2$, by the subderivations theorem, $\Gamma_1 \vdash A : Type$ is derivable. As ξ is adapted to Γ_1, using Theorem 6.4.1, $[\![A]\!]_\xi$ is a function from $[A]$ to $[Type] =$ SAT, hence $[\![A]\!]_\xi(\xi(x)) \in$ SAT. Thus, $\rho(x) = x \in [\![A]\!]_\xi(\xi(x))$.

- For any decomposition $\Gamma = \Gamma_1, h : P, \Gamma_2$, by the subderivations theorem, $\Gamma_1 \vdash P : Prop$ is derivable. As ξ is adapted to Γ_1, using Theorem 6.4.1, $[\![P]\!]_\xi \in [Prop] =$ SAT. Thus, $\rho(h) = h \in [\![P]\!]_\xi$.

Therefore, we can apply the main theorem to conclude that one of the following holds:

- T is a type and $M = (\!|M|\!)_\rho \in [\![T]\!]_\xi([\![M]\!]_\xi)$. As $[\![T]\!]_\xi([\![M]\!]_\xi) \in \bigcup \mathcal{U}$ and $M \in [\![T]\!]_\xi([\![M]\!]_\xi)$, $[\![T]\!]_\xi([\![M]\!]_\xi) \in$ SAT. Hence, $M \in [\![T]\!]_\xi([\![M]\!]_\xi) \subseteq$ SN.

- T is not a type and $M = (\!|M|\!)_\rho \in [\![T]\!]_\xi$. As $[\![T]\!]_\xi \in \bigcup \mathcal{U}$ and $M \in [\![T]\!]_\xi$, $[\![T]\!]_\xi \in \mathrm{SAT}$. Hence, $M \in [\![T]\!]_\xi \subseteq \mathrm{SN}$.

In order to conclude the expected result, we have to prove that for any derivable judgement $\Gamma \vdash M : T$, T belongs to SN. If $T = Kind$, we have directly $T \in \mathrm{SN}$. Else, by Theorem 5.2.3, $\Gamma \vdash T : s$ is derivable for some sort s. Hence, using the first result, $T \in \mathrm{SN}$. □

We conclude this section with the following corollary of the strong normalization theorem: the cut elimination theorem.

Theorem 6.5.3. *Whenever some PVS-Cert judgement of the form $\Gamma \vdash p : P$ is derivable for some proposition P and some proof p, p can be reduced using the reduction $\to_{\beta\sigma}$ to a normal form q such that the judgement $\Gamma \vdash q : P$ is derivable.*

Proof. By the strong normalization theorem, p belongs to SN. Hence, p can be reduced to a normal form q using the reduction $\to_{\beta\sigma}$. By the type preservation theorem (Theorem 5.5.2), the judgement $\Gamma \vdash q : P$ is derivable. □

6.6 Normal forms

As mentioned in the introduction of this chapter, the strong normalization of $\to_{\beta\sigma}$ will be used in the definition of a type-checking algorithm in Chapter 7. More precisely, we will use the restriction of $\to_{\beta\sigma}$ to weak head reduction, which will be denoted $\to^{wh}_{\beta\sigma}$. This reduction is defined as follows.

Definition 6.6.1. *For any reduction $\triangleright_-$ (e.g. $\triangleright_{\beta\sigma}$ or $\triangleright_{\beta\sigma}$), we define the associated weak head reduction $\to^{wh}_-$ as its congruence closure restricted to elimination contexts: $M \to^{wh}_- N$ if and only if there exists some elimination context e such that M has the form $e[M']$, N' has the form $e[N']$, and $M' \triangleright_- N'$.*

In this section, we analyze the structure of well-typed terms in normal form with respect to $\to^{wh}_{\beta\sigma}$. This analysis is also useful to study well-typed terms in normal form with respect to $\to_{\beta\sigma}$: given such a term M, all subterms of M admit the structure of well-typed terms in normal form with respect to $\to^{wh}_{\beta\sigma}$.

We prove the expected characterization of well-typed terms in normal form with respect to $\to^{wh}_{\beta\sigma}$ as follows.

Proposition 6.6.1. *If $\Gamma \vdash M : T$ is derivable, then M in normal form with respect to $\to^{wh}_{\beta\sigma}$ if and only if it has one of the following forms: s, $e[v]$, $\lambda v : T.N$, $\Pi v : T.U$, $\{v : T \mid U\}$, or $\langle M_1, M_2 \rangle_T$.*

Proof. We first suppose that M has one of the following forms: s, $e[v]$, $\lambda v : T.N$, $\Pi v : T.U$, $\{v : T \mid U\}$, or $\langle M_1, M_2 \rangle_T$. We prove that M in normal form with respect to $\to^{wh}_{\beta\sigma}$ by splitting the different cases. All of them are straightforward. On the other

hand, we suppose that M is in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$ and we prove that M has one of the expected forms by induction on M. Discarding the notations M and T, the possible cases are the following.

- The cases s, v, $\lambda v : T.M$, $\Pi v : T.U$, $\{v : T \mid U\}$, and $\langle M, N\rangle_T$ are straightforward.
- Case MN: by the subderivations theorem, there exists some term $\Pi v : T.U$ (α-renamed to ensure $v \notin DV(\Gamma)$) such that $\Gamma \vdash M : \Pi v : T.U$ is derivable. Moreover, as MN is in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$, this is also the case for M. Hence, we can conclude the expected result by induction hypothesis if we prove that M doesn't have the form s, $\lambda v' : T'.M'$, $\Pi v' : T'.U'$, $\{v' : T' \mid U'\}$, or $\langle M', N'\rangle_{T'}$. On the one hand, M cannot have the form $\lambda v' : T'.M'$ because MN wouldn't be in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$ if it was the case. On the other hand, if M has the form $\Pi v' : T'.U'$ (resp. $\{v' : T' \mid U'\}$, $\langle M', N'\rangle_{T'}$), by the subderivations theorem and the renaming theorem, $\Pi v : T.U \equiv_{\beta *} s$ for some sort s (resp. $\Pi v : T.U \equiv_{\beta *} Type$, $\Pi v : T.U \equiv_{\beta *} \{v : T'' \mid U''\}$ for some terms T'' and U''), which is impossible by Theorem 5.3.2. Finally, M cannot be a sort by the stratification theorem.
- Case $\pi_i(M)$: by the subderivations theorem, there exists some term $\{v : T \mid U\}$ such that $\Gamma \vdash M : \{v : T \mid U\}$ is derivable, where we choose $v \notin DV(\Gamma)$. Moreover, as $\pi_i(M)$ is in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$, this is also the case for M. Hence, we can conclude the expected result by induction hypothesis if we prove that M cannot have the form s, $\lambda v' : T'.M'$, $\Pi v' : T'.U'$, $\{v' : T' \mid U'\}$, or $\langle M', N'\rangle_{T'}$. On the one hand, M cannot have the form $\langle M', N'\rangle_{T'}$ because $\pi_i(M)$ wouldn't be in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$ if it was the case. On the other hand, if M has the form $\lambda v' : T'.M'$ (resp. $\Pi v' : T'.U'$, $\{v' : T' \mid U'\}$), by the subderivations theorem and the renaming theorem, $\{v : T \mid U\} \equiv_{\beta *} \Pi v : T''.U''$ for some terms T'' and U'' (resp. $\{v : T \mid U\} \equiv_{\beta *} s$ for some sort s, $\{v : T \mid U\} \equiv_{\beta *} Type$), which is impossible by Theorem 5.3.2. Finally, M cannot be a sort by the stratification theorem.

□

The following corollary will be useful for inhabited terms.

Corollary 6.6.1. *If $\Gamma \vdash M : T$ is derivable, then T in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$ if and only if it has one of the following forms: s, $e[v]$, $\Pi v : T_1.T_2$, $\{v : T_1 \mid T_2\}$.*

Proof. We first suppose that T has one of the following forms: s, $e[v]$, $\Pi v : T_1.T_2$, $\{v : T_1 \mid T_2\}$. We prove that T in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$ by splitting the different cases. All of them are straightforward. On the other hand, if T is in normal form with respect to $\rightarrow_{\beta\sigma}^{wh}$, we prove that it has one of the expected forms as follows. By Theorem 5.2.3, either $T = Kind$, in which case the result is straightforward, or $\Gamma \vdash T : s$ is derivable for some sort s. In this case, we can conclude by Proposition 6.6.1 if we prove that T doesn't have the form $\lambda v' : T'.M'$ or $\langle M', N'\rangle_{T'}$. On the one hand, if T has the

form $\lambda v' : T'.M'$, we choose $v' \notin DV(\Gamma)$, and conclude by the subderivations theorem and the renaming theorem that $s \equiv_{\beta *} \Pi v' : T'.U'$ for some term U', which is impossible by Theorem 5.3.2. On the other hand, if T has the form $\langle M', N' \rangle_{T'}$ we conclude by the subderivations theorem that $s \equiv_{\beta *} \{v' : T'_1.T'_2\}$ for some term $\{v' : T'_1.T'_2\}$, which is impossible by Theorem 5.3.2. □

The possible forms for normal forms presented in Corollary 6.6.1 are subject to the following extension of Theorem 5.3.2.

Proposition 6.6.2. *For all terms $M_1 \equiv_{\beta *} M_2$ such that for all $i \in \{1, 2\}$, M_i has one of the forms s, $e[v]$, $\Pi v : T.U$, or $\{v : T \mid M\}$, one of the following holds.*

- *There exists a sort s such that $M_1 = M_2 = s$.*
- *M_1 has the form $e_1[v]$ and M_2 has the form $e_2[v]$ for some variable v and some elimination contexts e_1 and e_2.*
- *M_1 has the form $\Pi v : T_1.U_1$ and M_2 has the form $\Pi v : T_2.U_2$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$.*
- *M_1 has the form $\{v : T_1 \mid U_1\}$ and M_2 has the form $\{v : T_2 \mid U_2\}$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$.*

Proof. We first prove that for any terms M_1 and M_2 such that $M_1 \twoheadrightarrow_{\beta *} M_2$, the following statements hold:

- If M_1 is a sort, $M_1 = M_2$.
- If M_1 has the form $e_1[v]$, then M_2 has the form $e_2[v]$.
- If M_1 has the form $\Pi v : T_1.U_1$, then M_2 has the form $\Pi v : T_2.U_2$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$
- If M_1 has the form $\{v : T_1 \mid U_1\}$, then M_2 has the form $\{v : T_2 \mid U_2\}$ where $T_1 \equiv_{\beta *} T_2$ and $U_1 \equiv_{\beta *} U_2$

The proof is done by induction on the length of the reduction $M_1 \twoheadrightarrow_{\beta *} M_2$. In the second case, we use the fact that whenever $e_1[v] \rightarrow_{\beta *} N$, N has the form $e_2[v]$, which is a direct consequence of Lemma 6.2.1. The other cases are straightforward.

In a second step, the expected result is proved as follows. If two terms $M_1 \equiv_{\beta *} M_2$ are such that for all $i \in \{1, 2\}$, M_i has one of the forms s, $e[v]$, $\Pi v : T.U$, or $\{v : T \mid M\}$, by the Church-Rosser theorem, there exists a term N such that $M_1 \twoheadrightarrow_{\beta *} N$ and $M_2 \twoheadrightarrow_{\beta *} N$. We conclude the expected result by splitting the 16 different cases for (M_1, M_2). All cases are straightforward. □

6.7 Defining and using cut elimination in PVS-Cert

We conclude this chapter showing how the cut elimination theorem (Theorem 6.5.3) can be used together with the characterizations of terms in normal form with respect to $\rightarrow_{\beta *}$ to investigate whether some PVS-Cert propositions are provable or not in some given context. The simplest question of this kind is the question of consistency.

Theorem 6.7.1. *PVS-Cert is consistent: there exists no proof p such that the judgement $\vdash p : \Pi x : Prop.x$ is derivable.*

Proof. We suppose that there exists a proof p such that the judgement $\vdash p : \Pi x : Prop.x$ admits some derivation, and find a contradiction in the following way. Using the thinning theorem, $x : Prop \vdash p : \Pi x : Prop.x$ is also derivable. Hence, applying the rule Lam followed by the rule App, $\vdash \lambda x : Prop.(px) : \Pi x : Prop.x$ is derivable.

By the cut elimination theorem 6.5.3, $\lambda x : Prop.(px)$ admits a normal form p' with respect to $\twoheadrightarrow_{\beta\sigma}$ which is such that the judgement $\vdash p' : \Pi x : Prop.x$ is derivable. By straightforward induction on the length of the reduction leading to this normal form, p' has the form $\lambda x : Prop.q$ for some proof q. Hence, by the type preservation theorem 5.5.2, $\vdash \lambda x : Prop.q : \Pi x : Prop.x$ is derivable.

By the subderivations theorem together with the renaming theorem and the stratification theorem, there exists a derivable judgement $x : Prop \vdash q : t$ for some expression $t \equiv_{\beta *} x$. Hence, as the judgement $x : Prop \vdash x : Prop$ is derivable, the conversion rule can be applied to conclude that $x : Prop \vdash q : x$ is also derivable.

Using Proposition 6.6.1 and the fact that q is a proof, q has one of the following forms: $\lambda v : T.M$ or $e[v]$. We discard the first possibility as follows. If $q = \lambda v : T.M$, by the subderivations theorem and the stratification theorem, there exists some term of the form $\Pi v' : T'.U'$ such that $\Pi v' : T'.U' \equiv_{\beta *} x$. This is not possible by Theorem 5.3.2. As a consequence, q has the form $e[v]$ for some elimination context e and some variable v.

By straightforward induction on e, some judgement of the form $x : Prop \vdash v : T$ is derivable. Hence, by the subderivations theorem and Theorem 5.3.2, $v = x$. As q is a proof, $e[x] \neq x$. In this setting, by straightforward induction on e, there exists a derivable judgement of the form $x : Prop \vdash \pi_1(x) : T$, of the form $x : Prop \vdash \pi_2(x) : T$, or of the form $x : Prop \vdash xt' : T$. By the subderivations theorem again and Theorem 5.6.1, this implies that there exists a term U of the form $\Pi v' : T'.U'$ or $\{v' : T' \mid U'\}$ such that $U \equiv_{\beta *} Prop$. This is not possible by Theorem 5.3.2. As a consequence, there exists no proof term p such that the judgement $\vdash p : \Pi x : Prop.x$ is derivable.

□

Although the question of consistency is a simple illustration of the way the cut elimination theorem can be applied, it is not a question for which using the cut elimination

is necessary. One alternative proof is to use the conservativity of PVS-Core over higher-order logic, which will be establish in Chapter 8, to prove the consistency of PVS-Core from the consistency of higher-order logic – assuming that this latter property was already established. Another alternative is to use a much simpler set-theoretical model than the one defined to establish the strong normalization theorem of PVS-Cert: a notion of *standard* set-theoretical interpretation of predicate subtyping, where the type $Prop$ is interpreted as a set exactly two elements, distinguishing *true* propositions from *false* ones. As presented in Section 1.5 of the introductory chapter, this approach has been formalized in [61] for some subset of PVS which is larger that PVS-Core.

However, the set of provable propositions in higher-order logic is strictly included in the set of propositions that are true in all standard models and, as PVS-Core is a conservative extension of higher-order logic, this statement also holds for PVS-Core. As a consequence, the study of such interpretations is not sufficient to delineate the set of provable propositions of PVS-Core. The same situation holds whenever one aims at delineating the set of provable propositions using the conservativity of PVS-Cert over higher-order logic established in Chapter 8 and its underlying translation from PVS-Cert to the PTS λ-HOL, which will be denoted $[\![\cdot]\!]_{\lambda HOL}$ in the remaining of this chapter. More precisely, proving that the image of some PVS-Cert judgement through the translation $[\![\cdot]\!]_{\lambda HOL}$ is derivable in λ-HOL is not sufficient to conclude that the original statement is derivable in PVS-Cert. The following theorem illustrates a situation where neither the standard set-theoretical interpretation nor the conservativity theorem over higher-order logic can be applied to conclude the expected result: in this situation, using the cut elimination theorem becomes necessary.

Theorem 6.7.2. *In PVS-Cert, Leibniz's definition of equality matches conversion: whenever some judgement of the form* $\vdash p : \Pi x : (\Pi y : A.Prop).\Pi h : xt.xu$ *is derivable, then* $t \equiv_{\beta *} u$.

Before proving this result, we illustrate briefly why neither a standard set-theoretical interpretation nor the conservativity of PVS-Cert over higher-order logic can be applied to conclude the expected result. We consider the case $A = Prop$, $t = \Pi z : \{y : Prop \mid y\}.\pi_1(z)$ and $u = \Pi z : Prop.\Pi h : z.z$. We denote $P = \Pi x : (\Pi y : A.Prop).\Pi h : xt.xu$. Using Theorem 5.3.2, we first prove that $\{y : Prop \mid y\}$ and $Prop$ are not convertible. Hence, using Theorem 5.3.2 again, we conclude that t and u are not convertible either. As a consequence, by Theorem 6.7.2, P admits no proof in the empty context. We show that this result cannot be obtained neither using a standard set-theoretical interpretation nor using the conservativity theorem over λ-HOL and its underlying translation to λ-HOL:

- Under the standard set-theoretical interpretation, we first notice that two propositions t and u are both *true*, by analyzing the two only possible interpretations for a proposition. As a consequence, using a simple case analysis, the proposition P is *true* as well. Thus, the set-theoretical interpretation is not sufficient to prove that the proposition P is not provable.

- Under the translation from PVS-Cert to PVS-Core, $[\![t]\!]_{\lambda HOL} = [\![u]\!]_{\lambda HOL} = u$. As a consequence, $[\![P]\!]_{\lambda HOL} = \Pi x : (\Pi y : Prop.Prop).\Pi h : xu.xu$. We consider the proof $p = \lambda x : (\Pi y : Prop.Prop).\lambda h : (xu).h$. The judgement $\vdash p : [\![P]\!]_{\lambda HOL}$ is derivable in λ-HOL. Hence, the translation from PVS-Cert to λ-HOL used in the conservativity theorem is not sufficient to prove that the proposition P is not provable.

Theorem 6.7.2 is proved as follows using the cut elimination theorem.

Proof. We suppose that the judgement $\vdash p : \Pi x : (\Pi y : A.Prop).\Pi h : xt.xu$ admits some derivation. By Theorem 5.2.3 and the stratification theorem, the judgement $\vdash \Pi x : (\Pi y : A.Prop).\Pi h : xt.xu : Prop$ is derivable. Hence, applying the subderivations theorem together with the renaming theorem iteratively, the judgement $x : (\Pi y : A.Prop), h : xt \vdash WF$ is derivable. As a consequence, applying the thinning theorem followed by the application of the rules LAM and APP iteratively, the judgement $\vdash \lambda x : (\Pi y : A.Prop).\lambda h : xt.(pxh) : \Pi x : (\Pi y : A.Prop).\Pi h : xt.xu$ is derivable.

By the strong normalization theorem 6.5.2, $\lambda x : (\Pi y : A.Prop).\lambda h : xt.(pxh)$ admits a normal form with respect to $\twoheadrightarrow_{\beta\sigma}$. By straightforward induction on the length of the reduction leading to this normal form, it has the form $\lambda x : (\Pi y : A.Prop).\lambda h : xt.q$ for some proof q. Hence, by the type preservation theorem 5.5.2, $\vdash \lambda x : (\Pi y : A.Prop).\lambda h : xt.q : \Pi x : (\Pi y : A.Prop).\Pi h : xt.xu$ is derivable.

By the subderivations theorem together with the renaming theorem and the stratification theorem, there exists a derivable judgement $x : (\Pi y : A.Prop) \vdash \lambda h : xt.q : P$ for some expression $P \equiv_{\beta *} \Pi h : xt.xu$. On the other hand, as the judgement $\vdash \Pi x : (\Pi y : A.Prop).\Pi h : xt.xu : Prop$ is derivable, the subderivations theorem can be used together with the stratification theorem to conclude that the judgement $x : (\Pi y : A.Prop) \vdash \Pi h : xt.xu : Prop$ is also derivable. In this setting, using the conversion rule, the judgement $x : (\Pi y : A.Prop) \vdash \lambda h : xt.q : \Pi h : xt.xu$ is derivable.

By the subderivations theorem together with the renaming theorem and the stratification theorem, there exists a derivable judgement $x : (\Pi y : A.Prop), h : xt \vdash q : Q$ for some expression $Q \equiv_{\beta *} xu$. On the other hand, as the judgement $x : (\Pi y : A.Prop) \vdash \Pi h : xt.xu : Prop$ is derivable, the subderivations theorem can be used together with the stratification theorem to conclude that the judgement $x : (\Pi y : A.Prop), h : xt \vdash xu : Prop$ is also derivable. In this setting, using the conversion rule, the judgement $x : (\Pi y : A.Prop), h : xt \vdash q : xu$ is derivable.

Using Proposition 6.6.1 and the fact that q is a proof, q has one of the following forms: $\lambda v : T.M$ or $e[v]$. We discard the first possibility as follows. If $q = \lambda v : T.M$, by the subderivations theorem and the stratification theorem, there exists some term of the form $\Pi v' : T'.U'$ such that $\Pi v' : T'.U' \equiv_{\beta *} xu$. This is not possible by Theorem 6.6.2.

As a consequence, q has the form $e[v]$.

By straightforward induction on e, some judgement of the form $x : (\Pi y : A.Prop), h : xt \vdash v : T$ is derivable. Hence, by the subderivations theorem and Theorem 5.3.2, either $v = x$ or $v = h$. We discard the case $e \neq \bullet$ as follows. If $e \neq \bullet$, by straightforward induction on e, there exists a derivable judgement of the form $x : (\Pi y : A.Prop), h : xt \vdash \pi_1(v) : T$, $x : (\Pi y : A.Prop), h : xt \vdash \pi_2(v) : T$, or $x : (\Pi y : A.Prop), h : xt \vdash vt' : T$. By the subderivations theorem again and Theorem 5.6.1, this implies that there exists a term U of the form $\Pi v' : T'.U'$ or $\{v' : T' \mid U'\}$ such that either $U \equiv_{\beta *} Prop$ in the case $v = x$ or $U \equiv_{\beta *} xt$ in the case $v = h$. This is not possible by Theorem 6.6.2. As a consequence, $e \neq \bullet$. As a consequence, $q = v$, and, as q is a proof term, this implies that $q = h$. Thus, the judgement $x : (\Pi y : A.Prop), h : xt \vdash h : xu$ is derivable.

On the other hand, as the judgement $x : (\Pi y : A.Prop), h : xt \vdash WF$ is derivable, the rule Var can be applied to conclude that the judgement $x : (\Pi y : A.Prop), h : xt \vdash h : xt$ is derivable as well. Hence, by Theorem 5.6.1, $xt \equiv_{\beta *} xu$. By straightforward induction on the length of reductions, any term M such that $xt \twoheadrightarrow_{\beta *} M$ (resp. $xu \twoheadrightarrow_{\beta *} M$) has the form xN with $N \equiv_{\beta *} t$ (resp. $N \equiv_{\beta *} u$). Hence, using the Church-Rosser theorem, $t \equiv_{\beta *} u$.

□

Chapter 7

Type-checking in PVS-Cert

The purpose of this chapter is to present a type-checking algorithm for PVS-Cert. In spite of being a relatively simple consequence of the strong normalization theorem 6.5.2 established in the previous chapter, this result is one of the most important result expected for PVS-Cert. In particular, it will be used in Chapter 10 together with the translation from PVS-Core derivations to PVS-Cert established in Chapter 9 to show that PVS-Cert can be used as verifiable certificates for PVS-Core (Definition 10.1.1).

The type-checking algorithm presented for PVS-Cert is comparable to the algorithm given in [53] for the type system ECC, as both extend the type system PVS-Cert$^-$ presented in Definition 4.2.3. In both cases, type checking is based on type inference, and most cases are handled in a similar way. There are two main differences between the two algorithms.

- In PVS-Cert, type checking and type inference use the two reductions $\twoheadrightarrow_{\beta *}$ and $\twoheadrightarrow_{\beta\sigma}^{wh}$, while only $\twoheadrightarrow_{\beta\sigma}$ is used in ECC. More precisely, $\twoheadrightarrow_{\beta *}$ is used to define type checking from type inference, while $\twoheadrightarrow_{\beta\sigma}^{wh}$ is used in type inference in the case of applications, to reduce a type to some expected form $\Pi v : T.U$.

- The second important difference affects Lam rule:

$$\frac{\Gamma, v : T \vdash M : U \qquad \Gamma \vdash \Pi v : T.U : s}{\Gamma \vdash \lambda v : T.M : \Pi v : T.U}\ \textsc{Lam}$$

  The specific set of PTS rules used in ECC makes the second premise of the Lam rule unnecessary, which is not the case in PVS-Cert$^-$ and PVS-Cert. For instance, the judgement $x : \mathit{Prop} \vdash \mathit{Prop} : \mathit{Type}$ is derivable PVS-Cert, while the judgement $\vdash \lambda x : \mathit{Prop}.\mathit{Prop} : \Pi x : \mathit{Prop}.\mathit{Type}$ is not. However, applying a recursive call on this second premise would be problematic. On the one hand, it would make the algorithm slower. On the other hand, it would break the simplicity of the proof of termination, based on the fact that recursive calls of type inference are done on subterms exclusively. This problem is shared by several type systems.

In the case of injective PTSs, one of the simplest solution for this problem is presented in [6], using some classification of terms to avoid this unwanted recursive call. The solution presented for PVS-Cert in this chapter is similar, based on the stratification theorem. However, it is simpler than the classification presented in [6] for several reasons. On the one hand, it is suited for a specific system instead of a wide family of type systems. On the other hand, the choice of presentation of the type system with one domain of variables per sort – which follows the presentation of PTSs given in [36] – makes classification easier than in the presentation given in [6], where there is only one single set of variables.

We present the type-checking algorithm together with two other algorithms: a type inference algorithm, and a well-formedness checking algorithm. The type-checking algorithm takes as input a context Γ, a term M, and a term T, and decides whether $\Gamma \vdash M : T$ is derivable. The type inference algorithm takes as inputs a context Γ and a term M, decides whether there exists a term T such that $\Gamma \vdash M : T$ is derivable, and outputs such a term in case of success. The well-formedness algorithm takes as input a context Γ and decides whether $\Gamma \vdash WF$ is derivable.

We first present the three algorithms as partial algorithms. Then, we prove that they are sound, i.e. that, in case of success, the associated judgement is derivable. We use this proof of soundness to prove that these algorithms terminate. Finally, we prove that they are complete, i.e. that in case of failure, the associated judgements are not derivable.

These proofs will be based on several results of the previous chapters. They will use in particular the uniqueness of types theorem 5.6.1, the type preservation theorem 5.5.2 for $\twoheadrightarrow_{\beta\sigma}$ and the strong normalization theorem 6.5.2 for both $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta *}$.

7.1 Definitions

We first define the following algorithm to distinguish types, expressions, $Type$, and $Kind$. As mentioned in the beginning of this chapter, this algorithm will be used to avoid an unwanted recursive call on the second premise of the LAM rule.

Definition 7.1.1. *We define the algorithm* $\textsc{Level}(\cdot)$ *by recursion on its argument. The possible cases are the following.*

- $\textsc{Level}(h) = 0$, $\textsc{Level}(x) = 1$, $\textsc{Level}(X) = 2$
- $\textsc{Level}(Prop) = 2$, $\textsc{Level}(Type) = 3$, $\textsc{Level}(Kind) = 4$
- $\textsc{Level}(\lambda v : T.M) = 1$, $\textsc{Level}(MN) = 1$
- $\textsc{Level}(\langle M, N \rangle_T) = 1$, $\textsc{Level}(\pi_1(M)) = 1$, $\textsc{Level}(\pi_2(M)) = 0$
- $\textsc{Level}(\Pi v : T.U) = \textsc{Level}(U)$, $\textsc{Level}(\{v : T \mid U\}) = 2$

The following proposition will be the only property needed for $\textsc{Level}(M)$. More precisely, only the two first statements will be used. The two last statements are presented to give a clearer view of the algorithm.

Proposition 7.1.1. *Whenever M is either an expression, a type, $Type$, or $Kind$, the following statements hold:*

- *M is an expression if and only if* $\textsc{Level}(M) = 1$
- *M is a type if and only if* $\textsc{Level}(M) = 2$
- *$M = Type$ if and only if* $\textsc{Level}(M) = 3$
- *$M = Kind$ if and only if* $\textsc{Level}(M) = 4$

Proof. The proof is straightforward by induction on M, using the fact that M is either an expression, a type, $Type$, or $Kind$. □

The algorithms of type-checking, type inference, and well-formedness checking rely on the two following auxiliary algorithms, which are variants of the type-checking and inference algorithms where the context given as input will be supposed to be well-formed. These auxiliary algorithms contain the core of the main algorithms.

Definition 7.1.2. *We define the two auxiliary algorithms* $\textsc{Infer-type-aux}(\Gamma \mid M)$ *and* $\textsc{Check-type-aux}(\Gamma \mid M \mid T)$. *The first algorithm is a partial type inference algorithm, which is only ensured to be sound when $\Gamma \vdash WF$ is derivable. The second algorithm is a partial type-checking algorithm, which is only ensured to be sound when $\Gamma \vdash T : s$ is derivable for some sort s.*

$\textsc{Infer-type-aux}(\Gamma \mid M)$ *is defined according to M:*

- *Case $M = s$: if there exists a unique axiom $(s, s') \in \mathcal{A}$, return s', else fail.*
- *Case $M = v$: if v belongs to some unique declaration $(v : T) \in \Gamma$, return T, else fail.*
- *Case $M = \lambda v : T.N$: if necessary, α-rename M to ensure $v \notin DV(\Gamma)$. If* $\textsc{Infer-type-aux}(\Gamma \mid T)$ *return the sort $s(v)$ successfully, continue as follows: if* $\textsc{Infer-type-aux}(\Gamma, v : T \mid N)$ *returns some term U successfully and if one of the following conditions holds, return $\Pi v : T.U$.*
  - *$s(v) = Prop$ and* $\textsc{Level}(U) = 1$,
  - *$s(v) = Type$ and* $\textsc{Level}(U) = 1$,
  - *$s(v) = Type$ and* $\textsc{Level}(U) = 2$.

  In all other cases, fail.

- *Case* $M = M_1 M_2$ *: if* INFER-TYPE-AUX$(\Gamma \mid M_1)$ *returns some term* T *successfully, reduce* T *to a normal form using the reduction* $\rightarrow_{\beta\sigma}^{wh}$*. If the term obtained has the form* $\Pi v : T_1.T_2$ *and* CHECK-TYPE-AUX$(\Gamma \mid M_2 \mid T_1)$ *succeeds, return* $T_2[M_2/v]$*. In all other cases, fail.*

- *Case* $M = \Pi v : T.U$*: if necessary,* α*-rename* M *to ensure* $v \notin DV(\Gamma)$*. If* INFER-TYPE-AUX$(\Gamma \mid T)$ *return the sort* $s(v)$ *successfully, continue as follows: if* INFER-TYPE-AUX$(\Gamma, v : T \mid U)$ *returns some sort* s *successfully and if there exists some unique sort* s' *such that* $(s(v), s, s') \in \mathcal{R}$*, return* s_3*. Else, fail.*

- *Case* $M = \{v : T \mid U\}$*: if necessary,* α*-rename* M *to ensure* $v \notin DV(\Gamma)$*. If* INFER-TYPE-AUX$(\Gamma \mid T)$ *return the sort* $s(v)$ *successfully and* $s(v) = Type$*, continue as follows: if* INFER-TYPE-AUX$(\Gamma, v : T \mid U)$ *returns* $Prop$*, return* $Type$*, else fail.*

- *Case* $M = \langle M_1, M_2 \rangle_T$*: if* T *has the form* $\{v : T_1 \mid T_2\}$ *and if* INFER-TYPE-AUX$(\Gamma \mid T)$ *returns* $Type$*, continue as follows. If* CHECK-TYPE-AUX$(\Gamma \mid M_1 \mid T_1)$ *succeeds, continue as follows: if* CHECK-TYPE-AUX$(\Gamma \mid M_2 \mid T_2[M_1/v])$ *succeeds, return* T*. In all other cases, fail.*

- *Case* $M = \pi_1(N)$*: if* INFER-TYPE-AUX$(\Gamma \mid N)$ *returns some term of the form* $\{v : T \mid U\}$ *successfully, return* T*, else fail.*

- *Case* $M = \pi_2(N)$*: if* INFER-TYPE-AUX$(\Gamma \mid N)$ *returns some term of the form* $\{v : T \mid U\}$ *successfully, return* $U[\pi_1(N)/v]$*, else fail.*

CHECK-TYPE-AUX$(\Gamma \mid M \mid T)$ *is defined as follows: if* INFER-TYPE-AUX$(\Gamma \mid M)$ *returns some term* U*, reduce* T *and* U *using the reduction* $\rightarrow_{\beta *}$ *until they reach normal forms, and succeed if the two normal forms are* α*-convertible. In all other cases, fail.*

The algorithms of type checking, type inference, and well-formedness are defined as follows.

Definition 7.1.3. *We define the algorithms of well-formedness checking, type inference, and type checking using the two auxiliary algorithms of Definition 13.5.2. They are denoted, respectively,* CHECK-WF(Γ)*,* INFER-TYPE$(\Gamma \mid M)$*, and* CHECK-TYPE$(\Gamma \mid M \mid T)$*, where* Γ *is a context, and* M *and* T *are terms.*

CHECK-WF(Γ) *is defined as follows: if* $\Gamma = \varnothing$*, terminate successfully. Else,* Γ *has the form* $\Gamma', v : T$*. If* $v \notin DV(\Gamma')$ *and* CHECK-WF(Γ') *succeeds, continue as follows: if* INFER-TYPE-AUX$(\Gamma' \mid T)$ *returns* $s(v)$ *successfully, terminate successfully. In all other cases, fail.*

INFER-TYPE$(\Gamma \mid M)$ *is defined as follows: if* CHECK-WF(Γ) *succeeds, return* INFER-TYPE-AUX$(\Gamma \mid M)$*, else fail.*

CHECK-TYPE($\Gamma \mid M \mid T$) *is defined as follows: if* CHECK-WF(Γ) *succeeds, continue as follows. If* $T = Kind$ *and* INFER-TYPE-AUX($\Gamma \mid M$) *returns* $Kind$*, terminate successfully. If* $T \neq Kind$ *and* INFER-TYPE-AUX($\Gamma \mid T$) *returns some sort* s *successfully, return* CHECK-TYPE-AUX($\Gamma \mid M \mid T$).

The main theorem of this chapter is the correspondence between the three algorithms presented Definition 7.1.3 and their expected specifications. It is expressed as follows.

Theorem 7.1.1. *The algorithms* CHECK-WF*,* INFER-TYPE*, and* CHECK-TYPE *always terminate, and admit the following properties:*

- CHECK-WF(Γ) *succeeds if and only if* $\Gamma \vdash WF$ *is derivable*
- INFER-TYPE($\Gamma \mid M$) *succeeds if only if there exists a term* T *such that* $\Gamma \vdash M : T$ *is derivable, in which case it outputs such a term*
- CHECK-TYPE($\Gamma \mid M \mid T$) *succeeds if only if* $\Gamma \vdash M : T$ *is derivable*

Proof. The theorem is the direct consequence of the Propositions 7.2.1, 7.3.1, and 7.4.1 proved in the following of this chapter. □

Remark 7.1.1. *Using the property of uniqueness of types proved in Theorem 5.6.1, whenever some judgement* $\Gamma \vdash M : T$ *is derivable, the algorithm* INFER-TYPE($\Gamma \mid M$) *outputs some term* U *such that* $\Gamma \vdash M : U$ *is derivable and* $T \equiv_{\beta *} U$.

7.2 Soundness

The expected properties of soundness are the following.

Proposition 7.2.1. *The following statements hold:*

- *Whenever* CHECK-WF(Γ) *succeeds,* $\Gamma \vdash WF$ *is derivable*
- *Whenever* INFER-TYPE($\Gamma \mid M$) *outputs some term* T *successfully,* $\Gamma \vdash M : T$ *is derivable*
- *Whenever* CHECK-TYPE($\Gamma \mid M \mid T$) *succeeds,* $\Gamma \vdash M : T$ *is derivable*

The proof relies on the following lemma, which will use the type preservation theorem 5.5.2 for $\twoheadrightarrow_{\beta\sigma}$.

Lemma 7.2.1. *The following statements hold:*

- *Whenever* $\Gamma \vdash WF$ *is derivable and* INFER-TYPE-AUX($\Gamma \mid M$) *outputs some term* T *successfully,* $\Gamma \vdash M : T$ *is derivable*
- *Whenever* $\Gamma \vdash T : s$ *is derivable and* CHECK-TYPE-AUX($\Gamma \mid M \mid T$) *succeeds,* $\Gamma \vdash M : T$ *is derivable*

Proof. We prove the two statements together by induction on M. We suppose that the two statements hold for any strict subterm of M. We prove the first statement for M as follows.

- Case $M = s$: we conclude applying the rule SORT to the derivation of $\Gamma \vdash WF$.
- Case $M = v$: we conclude applying the rule VAR to the derivation of $\Gamma \vdash WF$.
- Case $M = \lambda v : T.N$ (α-renamed to ensure $v \notin DV(\Gamma)$): by induction hypothesis, $\Gamma \vdash T : s(v)$ is derivable, hence $\Gamma, v : T \vdash WF$ is derivable. By induction hypothesis again, $\Gamma, v : T \vdash N : U$ is derivable, where U is the output of INFER-TYPE-AUX$(\Gamma, v : T \mid N)$. By the stratification theorem, U is either an expression, a type, $Type$, or $Kind$. We conclude as follows:
  - If $s(v) = Prop$ and LEVEL$(U) = 1$, then U is an expression by Proposition 7.1.1. By Theorem 5.2.3 followed by the stratification theorem, $\Gamma, v : T \vdash U : Prop$, from which we conclude that $\Gamma \vdash \Pi v : T.U : Prop$ and $\Gamma \vdash \lambda v : T.N : \Pi v : T.U$ are derivable
  - If $s(v) = Type$ and LEVEL$(U) = 1$, then U is an expression by Proposition 7.1.1. By Theorem 5.2.3 followed by the stratification theorem, $\Gamma, v : T \vdash U : Prop$, from which we conclude that $\Gamma \vdash \Pi v : T.U : Prop$ and $\Gamma \vdash \lambda v : T.N : \Pi v : T.U$ are derivable
  - If $s(v) = Type$ and LEVEL$(U) = 2$, then U is a type by Proposition 7.1.1. By Theorem 5.2.3 followed by the stratification theorem, $\Gamma, v : T \vdash U : Type$, from which we conclude that $\Gamma \vdash \Pi v : T.U : Type$ and $\Gamma \vdash \lambda v : T.N : \Pi v : T.U$ are derivable
- Case $M = M_1 M_2$: by induction hypothesis, $\Gamma \vdash M_1 : T$ is derivable, where T is the output of INFER-TYPE-AUX$(\Gamma \mid M_1)$. We consider $\Pi v : T_1.T_2$ the normal form obtained from T using $\rightarrow^{wh}_{\beta\sigma}$. By the type preservation theorem 5.5.2 for $\twoheadrightarrow_{\beta\sigma}$, $\Gamma \vdash M_1 : \Pi v : T_1.T_2$ is derivable. On the other hand, by the stratification theorem, $T_1 \neq Kind$, hence, by Theorem 5.2.3, $\Gamma \vdash T_1 : s$ is derivable for some sort s. Therefore, by induction hypothesis, $\Gamma \vdash M_2 : T_1$ is derivable too. Hence, $\Gamma \vdash M_1 M_2 : T_2[M_2/v]$ is derivable.
- Case $M = \Pi v : T.U$ (α-renamed to ensure $v \notin DV(\Gamma)$): by induction hypothesis, $\Gamma \vdash T : s(v)$ is derivable, hence $\Gamma, v : T \vdash WF$ is derivable. By induction hypothesis again, $\Gamma, v : T \vdash U : s$ is derivable, where s is the output of INFER-TYPE-AUX$(\Gamma, v : T \mid U)$. We conclude applying the rule PROD.
- Case $M = \{v : T \mid U\}$ (α-renamed to ensure $v \notin DV(\Gamma)$): by induction hypothesis, $\Gamma \vdash T : Type$ and $s(v) = Type$, hence $\Gamma, v : T \vdash WF$ is derivable. By induction hypothesis again, $\Gamma, v : T \vdash U : Prop$ is derivable. We conclude applying the rule SUBTYPE.

- Case $M = \langle M_1, M_2 \rangle_T$: T has the form $\{v : T_1 \mid T_2\}$ (α-renamed to ensure $v \notin DV(\Gamma)$). By induction hypothesis, $\Gamma \vdash \{v : T_1 \mid T_2\} : Type$ is derivable. Hence, by the subderivations theorem and the renaming theorem, $\Gamma \vdash T_1 : Type$ and $\Gamma, v : T_1 \vdash T_2 : Prop$ are derivable. Therefore, by induction hypothesis, $\Gamma \vdash M_1 : T_1$ is derivable. By the substitution theorem, $\Gamma \vdash T_2[M_1/v] : Prop$ is derivable. Hence, by induction hypothesis, $\Gamma \vdash M_2 : T_2[M_1/v]$ is derivable. We conclude applying the rule PAIR.

- Case $M = \pi_1(N)$ (resp. $M = \pi_2(N)$): by induction hypothesis, $\Gamma \vdash N : \{v : T \mid U\}$ is derivable, where $\{v : T \mid U\}$ is the output of INFER-TYPE-AUX($\Gamma \mid N$). We conclude applying the rule PROJ1 (resp. PROJ2).

We prove the second statement as follows. As $\Gamma \vdash T : s$ is derivable for some sort s, the judgement $\Gamma \vdash WF$ is derivable by the subderivations theorem. Applying the first statement already proved for M, the judgement $\Gamma \vdash M : U$ is derivable, where U is the output of INFER-TYPE-AUX($\Gamma \mid M$). As T and U share a common normal form using $\rightarrow_{\beta *}$, $T \equiv_{\beta *} U$. Therefore, applying the conversion rule, $\Gamma \vdash M : T$ is derivable. □

We conclude the proof of Proposition 7.2.1 as follows.

Proof. [Proposition 7.2.1] The first statement is proved by induction on Γ. If $\Gamma = \varnothing$, we conclude using the rule EMPTY. On the other hand, we prove that the statement holds for a context $\Gamma', v : T$ whenever it holds for Γ' as follows. By induction hypothesis, $\Gamma' \vdash WF$ is derivable. Hence, by Lemma 7.2.1, $\Gamma' \vdash T : s(v)$ is derivable. Therefore, as $v \notin DV(\Gamma')$, $\Gamma', v : T \vdash WF$ is derivable as expected.

The second statement is proved as follows: using the first statement, $\Gamma \vdash WF$ is derivable. Hence, by Lemma 7.2.1, $\Gamma \vdash M : T$ is derivable, where T is the output of INFER-TYPE-AUX($\Gamma \mid M$).

The third statement is proved as follows: using the first statement, $\Gamma \vdash WF$ is derivable. If $T = Kind$, by Lemma 7.2.1, $\Gamma \vdash M : T$ is derivable as expected. Else, by Lemma 7.2.1, $\Gamma \vdash T : s$ is derivable for some sort s. Hence, by Lemma 7.2.1 again, $\Gamma \vdash M : T$ is derivable. □

7.3 Termination

The expected properties of termination are the following.

Proposition 7.3.1. *The algorithms* CHECK-WF*,* INFER-TYPE*, and* CHECK-TYPE *always terminate.*

The proof relies on the following lemma, which will use the strong normalization theorem 6.5.2 for $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta *}$, as well as Lemma 7.2.1 proved in the previous section.

Lemma 7.3.1. *The following statements hold:*

- *Whenever* $\Gamma \vdash WF$ *is derivable,* INFER-TYPE-AUX$(\Gamma \mid M)$ *terminates*
- *Whenever* $\Gamma \vdash T : s$ *is derivable,* CHECK-TYPE-AUX$(\Gamma \mid M \mid T)$ *terminates*

Proof. We prove the two statements together by induction on M. We suppose that the two statements hold for any strict subterm of M. We prove the first statement for M as follows.

- The cases $M = s$ and $M = v$ are straightforward.
- Cases $M = \lambda v : T.N$, $M = \Pi v : T.U$, $M = \{v : T \mid U\}$, $M = \langle M_1, M_2 \rangle_T$, $M = \pi_1(N)$, and $M = \pi_2(N)$: following the proof of Lemma 7.2.1, every call of the form INFER-TYPE-AUX$(\Gamma' \mid M')$ (resp. CHECK-TYPE-AUX$(\Gamma' \mid M' \mid T')$) is done when M' is a strict subterm of M and $\Gamma' \vdash WF$ is derivable (resp. $\Gamma' \vdash T' : s$ is derivable for some sort s). Hence, we can conclude termination by induction hypothesis.
- Case $M = M_1 M_2$: on the one hand, following the proof of Lemma 7.2.1, every call of the form INFER-TYPE-AUX$(\Gamma' \mid M')$ (resp. CHECK-TYPE-AUX$(\Gamma' \mid M' \mid T')$) is done when M' is a strict subterm of M and $\Gamma' \vdash WF$ is derivable (resp. $\Gamma' \vdash T' : s$ is derivable for some sort s). Hence, these call terminate by induction hypothesis. On the other hand, we have to check that whenever INFER-TYPE-AUX$(\Gamma \mid M_1)$ returns some term T successfully, the reduction of T with $\rightarrow_{\beta\sigma}^{wh}$ terminates. By Lemma 7.2.1, $\Gamma \vdash M_1 : T$ is derivable. Hence, by the strong normalization theorem 6.5.2, the reduction of T with $\rightarrow_{\beta\sigma}^{wh}$ terminates.

We prove the second statement as follows. Applying the first statement already proved for M, INFER-TYPE-AUX$(\Gamma \mid M)$ terminates. We also have to prove that, if a term U is returned, the reduction of T and U with $\rightarrow_{\beta *}$ terminates. On the one hand, by hypothesis, $\Gamma \vdash T : s$ is derivable for some sort s, hence, by the strong normalization theorem 6.5.2, the reduction of T with $\rightarrow_{\beta *}$ terminates. On the other hand, by Lemma 7.2.1, $\Gamma \vdash M : U$ is derivable. Hence, by the strong normalization theorem 6.5.2, the reduction of U with $\rightarrow_{\beta *}$ terminates. □

We conclude the proof of Proposition 7.3.1 as follows.

Proof. [Proposition 7.3.1] The termination of CHECK-WF(Γ) is proved by induction on Γ, and the termination of INFER-TYPE$(\Gamma \mid M)$ and CHECK-TYPE$(\Gamma \mid M \mid T)$ is proved subsequently. In each case, following the proof of Proposition 7.2.1, we show that every call of the form INFER-TYPE-AUX$(\Gamma' \mid M')$ (resp. CHECK-TYPE-AUX$(\Gamma' \mid M' \mid T')$) is done when $\Gamma' \vdash WF$ is derivable (resp. $\Gamma' \vdash T' : s$ is derivable for some sort s), which allows to conclude using Lemma 7.3.1. □

7.4 Completeness

The expected properties of completeness are the following.

Proposition 7.4.1. *The following statements hold:*

- *Whenever* $\Gamma \vdash WF$ *is derivable,* CHECK-WF(Γ) *succeeds*
- *Whenever* $\Gamma \vdash M : T$ *is derivable,* INFER-TYPE($\Gamma \mid M$) *outputs some term* U
- *Whenever* $\Gamma \vdash M : T$, CHECK-TYPE($\Gamma \mid M \mid T$) *succeeds*

The proof relies on the following lemma, which uses the theorem of uniqueness of types and its corollary 5.6.1.

Lemma 7.4.1. *The following statements hold:*

- *Whenever* $\Gamma \vdash M : T$ *is derivable,* INFER-TYPE-AUX($\Gamma \mid M$) *outputs some term successfully*
- *Whenever* $\Gamma \vdash M : T$ *and* $\Gamma \vdash T : s$ *are derivable,* CHECK-TYPE-AUX($\Gamma \mid M \mid T$) *succeeds*

Proof. We prove the two statements together by induction on M. We suppose that the two statements hold for any strict subterm of M. We prove the first statement for M as follows.

- Cases $M = s$ and $M = v$: by the subderivations theorem and Theorem 5.3.2, there exists some axiom of the form (s, s') in $\mathcal{A}$. Given the content of $\mathcal{A}$, it is unique.
- Cases $M = v$: by the subderivations theorem and the free variable theorem, there exists a unique declaration $(v : T') \in \Gamma$.
- Case $M = \lambda v : T'.N$ (α-renamed to ensure $v \notin DV(\Gamma)$): by the subderivations theorem and the renaming theorem, there exists some term $\Pi v : T'.U$ and some sort s such that $\Gamma, v : T' \vdash N : U$ and $\Gamma \vdash \Pi v : T'.U : s$ are derivable. By the subderivations theorem and the renaming theorem again, $\Gamma \vdash T' : s(v)$ is derivable and there exists some sort s' such that $\Gamma, v : T' \vdash U : s'$ is derivable. By induction hypothesis, INFER-TYPE-AUX($\Gamma \mid T'$) returns some term T''. By Lemma 7.2.1 and Corollary 5.6.1, $T'' = s(v)$. By induction hypothesis, INFER-TYPE-AUX($\Gamma, v : T' \mid N$) returns some term U', and, by Lemma 7.2.1, $\Gamma, v : T' \vdash N : U'$ is derivable. We conclude the expected properties relating $s(v)$ and LEVEL(U') by the stratification theorem and Proposition 7.1.1.
- Case $M = M_1 M_2$: by the subderivations theorem, there exists some term $\Pi v : T_1.T_2$ (α-renamed to ensure $v \notin DV(\Gamma)$) such that both $\Gamma \vdash M_1 : \Pi v : T_1.T_2$ and $\Gamma \vdash M_2 : T_1$ are derivable. By induction hypothesis and Lemma 7.2.1, INFER-TYPE-AUX($\Gamma \mid M_1$) returns some term T' such that $\Gamma \vdash M_1 : T'$ is derivable. Following the proof of Lemma 7.3.1, the reduction of T' with $\rightarrow_{\beta\sigma}^{wh}$ terminates, yielding a term T''. By the type preservation theorem 5.5.2 for $\twoheadrightarrow_{\beta\sigma}$, $\Gamma \vdash M_1 : T''$ is derivable. Hence, by Theorem 5.6.1, $\Pi v : T_1.T_2 \equiv_{\beta *} T''$. Hence, by Corollary 6.6.1 and Proposition 6.6.2, T'' has the form $\Pi v : T_1''.T_2''$ with $T_1'' \equiv_{\beta *} T_1$ and

$T_2'' \equiv_{\beta *} T_2$. By Theorem 5.2.3, $\Gamma \vdash \Pi v : T_1''.T_2'' : s$ is derivable for some sort s. Hence, by the subderivations theorem, $\Gamma \vdash T_1'' : s'$ is derivable for some sort s'. Using the conversion rule, we conclude that $\Gamma \vdash M_2 : T_1''$ is derivable. Hence, applying induction hypothesis, CHECK-TYPE-AUX($\Gamma \mid M_2 \mid T_1''$) succeeds.

- Case $M = \Pi v : U_1.U_2$ (α-renamed to ensure $v \notin DV(\Gamma)$): by the subderivations theorem and the renaming theorem, there exists some rule $(s_1, s_2, s_3) \in \mathcal{R}$ such that $\Gamma \vdash U_1 : s_1$, $\Gamma, v : U_1 \vdash U_2 : s_2$, and $\Gamma \vdash \Pi v : U_1.U_2 : s_3$ are derivable. By the subderivations theorem again, $\Gamma \vdash U_1 : s(v)$ is derivable. Hence, by Corollary 5.6.1, $s_1 = s(v)$. By induction hypothesis, INFER-TYPE-AUX($\Gamma \mid U_1$) returns some term T'. By Lemma 7.2.1 and Corollary 5.6.1, $T' = s(v)$. By induction hypothesis, INFER-TYPE-AUX($\Gamma, v : U_1 \mid U_2$) returns some term T''. By Lemma 7.2.1 and Corollary 5.6.1, $T'' = s_2$, as expected. Finally, given the content of $\mathcal{R}$, s_2 is the unique sort such that $(s(v), s_1, s_2) \in \mathcal{R}$.

- The case $M = \{v : U_1 \mid U_2\}$ is similar to $M = \Pi v : U_1.U_2$.

- Case $M = \langle M_1, M_2 \rangle_U$: by the subderivations theorem and the renaming theorem, U has the form $\{v : U_1 \mid U_2\}$ (α-renamed to ensure $v \notin DV(\Gamma)$), and the three derivations $\Gamma \vdash M_1 : U_1$, $\Gamma \vdash M_2 : U_2[M_1/v]$, and $\Gamma \vdash U : Type$ are derivable. By induction hypothesis, INFER-TYPE-AUX($\Gamma \mid U$) returns some term T'. By Lemma 7.2.1 and Corollary 5.6.1, $T' = Type$. By the stratification theorem, $U_1 \neq Kind$ and $U_2[M_1/v] \neq Kind$, hence, by Theorem 5.2.3, $\Gamma \vdash U_1 : s$ and $\Gamma \vdash U_2[M_1/v] : s'$ are derivable for some sorts s and s'. Therefore, applying the induction hypothesis, CHECK-TYPE-AUX($\Gamma \mid M_1 \mid U_1$) and CHECK-TYPE-AUX($\Gamma \mid M_2 \mid U_2[M_1/v]$) succeed, as expected.

- Case $M = \pi_i(N)$ ($i \in \{1, 2\}$): by the subderivations theorem, there exists some term $\{v : T_1 \mid T_2\}$ such that $\Gamma \vdash N : \{v : T_1 \mid T_2\}$ is derivable. Hence, INFER-TYPE-AUX($\Gamma \mid N$) succeeds, as expected.

The second statement is proved as follows. Applying the first statement for M, INFER-TYPE-AUX($\Gamma \mid M$) returns some term U. By Lemma 7.2.1 and Theorem 5.6.1, $T \equiv_{\beta *} U$. Following the proof of Lemma 7.3.1, the reduction of T and U with $\rightarrow_{\beta *}$ terminates, yielding two normal forms T' and U' respectively. By the Church-Rosser theorem, T' and U' are α-convertible, as expected. □

We conclude the proof of Proposition 7.4.1 as follows.

Proof. [Proposition 7.4.1] The first statement is proved by induction on Γ. The case $\Gamma = \varnothing$ is straightforward. The case $\Gamma = \Gamma', v : T$ is proved as follows. By the subderivations theorem, $\Gamma' \vdash WF$ is derivable. Hence, by induction hypothesis, CHECK-WF(Γ') succeeds. By the subderivations again, $\Gamma' \vdash T : s(v)$ is derivable. Hence, by Lemma 7.4.1, INFER-TYPE-AUX($\Gamma' \mid T$) returns some term U. By Lemma 7.2.1 and Corollary 5.6.1, $U = s(v)$, as expected.

The second statement is proved as follows. On the one hand, applying the first statement, CHECK-WF(Γ) succeeds. On the other hand, by Lemma 7.4.1, INFER-TYPE-AUX($\Gamma \mid M$) returns some term successfully.

The third statement is proved as follows. Applying the first statement, CHECK-WF(Γ) succeeds. We first conclude for the case $T = Kind$ as follows. By Lemma 7.4.1, INFER-TYPE-AUX($\Gamma \mid M$) returns some term U. By Lemma 7.2.1 and Corollary 5.6.1, $U = Kind$, as expected. On the other hand, we conclude for the case $T \neq Kind$ as follows. By Theorem 5.2.3, $\Gamma \vdash T : s$ is derivable for some sort s. By Lemma 7.4.1, INFER-TYPE-AUX($\Gamma \mid T$) returns some term U. By Lemma 7.2.1 and Corollary 5.6.1, $U = s$, as expected. By Lemma 7.4.1 again, CHECK-TYPE-AUX($\Gamma \mid M \mid T$) succeeds. $\square$

Chapter 8

A conservative extension of higher-order logic

The purpose of this chapter is to prove that PVS-Cert is a conservative extension of λ-HOL: whenever $\Gamma \vdash P : Prop$ is derivable in λ-HOL, the fact that P is inhabited in Γ holds in PVS-Cert if and only if it holds in λ-HOL. Together with the strong normalization theorem 6.5.2, this theorem is one of the major contributions of this work on the theoretical point of view.

This theorem can be used as a tool to study the inhabitance of propositions in PVS-Cert contexts. Given any λ-HOL theory composed of a finite number of axioms, it allows to deduce the consistency of this theory in PVS-Cert from its consistency in λ-HOL. In particular, it shows that the addition of the most important logical principles used in the proof assistant PVS, such as the law of excluded middle or extensionality properties (mentioned in Section 2.2.2), can be safely added to PVS-Cert as they can be safely added to λ-HOL.

This theorem is comparable to the following result, proved in [54]: the extension CC+ of λ-HOL with the rule $(Prop, Type, Type)$ – which corresponds equivalently to the extension of the calculus of constructions [19] equipped with a sort $Kind$ and the axiom $(Type, Kind)$ – is conservative.

Remark 8.0.1. *Because of the presence of the rule $(Prop, Type, Type)$, there exists another natural way to express higher-order logic in CC+ (as well as in the plain calculus of constructions): instead of expressing types as types, it is also possible to express all non propositional types as terms typed by $Prop$. In this alternative setting, conservativity does not hold, as proved in [9] and [34].*

In both cases, the core of the problem is, starting from a judgement $\Gamma \vdash P : Prop$ derivable in λ-HOL and a judgement $\Gamma \vdash p : P$ derivable in PVS-Cert (resp. CC+), to find a proof q such that $\Gamma \vdash q : P$ is derivable is λ-HOL. A first natural approach would be to use strong normalization with the type preserving reduction $\rightarrow_{\beta\sigma}$ (resp.

$\rightarrow_\beta$), but this approach fails in both cases for similar reasons, as shown by the following counter-examples.

Example 8.0.1. *We consider for both PVS-Cert and CC+ the λ-HOL expression $P = \Pi x : Prop.\Pi h : (\Pi y : Prop.x).x$. In higher-order logic notations, P corresponds to $\forall x : Prop.(\forall y : Prop.x) \Rightarrow x$, i.e. to the impredicative expression of the existence of some proposition y. In λ-HOL, $\vdash P : Prop$ is derivable.*

We consider the following propositions $Q_{PVS-Cert}$ and Q_{CC+} which will be used to prove P in PVS-Cert and CC+ respectively, and include specific features of PVS-Cert and CC+ respectively.

- $Q_{PVS-Cert} = \Pi x : \{y : Prop \mid y\}.\pi_1(x)$
- $Q_{CC+} = \Pi x : Prop.\Pi y : (\Pi h : x.Prop).x$

Using these propositions, we consider the following proofs $p_{PVS-Cert}$ and p_{CC+}.

- $p_{PVS-Cert} = \lambda x : Prop.\lambda h : (\Pi y : Prop.x).hQ_{PVS-Cert}$
- $p_{CC+} = \lambda x : Prop.\lambda h : (\Pi y : Prop.x).hQ_{CC+}$

These proofs are counter-examples as the following propositions hold

- $\vdash p_{PVS-Cert} : P$ *is derivable in PVS-Cert and*
  $\vdash p_{CC+} : P$ *is derivable in CC+*
- $p_{PVS-Cert}$ *is in normal form with respect to* $\rightarrow_{\beta\sigma}$ *and*
  p_{CC+} *in normal form with respect to* $\rightarrow_\beta$
- *Neither* $\vdash p_{PVS-Cert} : P$ *nor* $\vdash p_{CC+} : P$ *are derivable in λ-HOL*
  as neither $\vdash Q_{PVS-Cert} : Prop$ *nor* $\vdash Q_{CC+} : Prop$ *are derivable in λ-HOL*

Another natural approach would be to use some erasure function from PVS-Cert (resp. CC+) to λ-HOL removing all parts of the syntax that are not defined (or cannot be well-typed) in λ-HOL. In the case of CC+, this approach is successfully used in [54] to prove the expected conservativity result. The erasure function corresponds to a natural extension of the erasure function for the calculus of constructions, defined in [8] and [63]. In PVS-Cert, a similar erasure function can be defined, at least at the levels of types, propositions, and stratified contexts.

Definition 8.0.1. *We define an erasing function $[\cdot]$ from PVS-Cert types, expressions, and contexts to λ-HOL types, expressions, and stratified contexts. This function erases all predicate subtypes $\{x : \cdot | P\}$ and coercions $\langle \cdot, p \rangle_A$ and $\pi_1(\cdot)$. More precisely, the translation of types is defined as follows.*

- $[X] = X$
- $[Prop] = Prop$

- $[\Pi x : A.B] = \Pi x : [A].[B]$
- $[\{x : A \mid P\}] = [A]$

The translation of expressions is defined as follows.

- $[x] = x$
- $[\Pi x : A.P] = \Pi x : [A].[P]$
- $[\Pi h : P.Q] = \Pi h : [P].[Q]$
- $[\lambda x : A.t] = \lambda x : [A].[t]$
- $[t\ u] = [t][u]$
- $[\langle t, M\rangle_A] = [t]$
- $[\pi_1(t)] = [t]$

The translation of stratified contexts is defined as follows.

- $[\varnothing] = \varnothing$
- $[\Gamma, h : P] = [\Gamma], h : [P]$
- $[\Gamma, x : A] = [\Gamma], x : [A]$
- $[\Gamma, X : Type] = [\Gamma], X : Type$

In the same way as the erasure function defined in [54], this function preserves any type or expression that is well-typed in λ-HOL, and any context that is well-formed in λ-HOL. However, unlike the case of CC+, $[\cdot]$ cannot be easily extended to proofs, as there is no natural definition of erasure for proofs of the form $\pi_2(t)$. Worse, there exists an expression P inhabited in a well-formed context Γ in PVS-Cert such that $[P]$ is empty in $[\Gamma]$ in λ-HOL, which shows that this erasure function $[\cdot]$ cannot be used directly to prove the expected conservativity result. This situation is illustrated in the following example.

Example 8.0.2. *We present a judgement $\Gamma \vdash p : P$ derivable in PVS-Cert such that $[P]$ is empty in $[\Gamma]$ in λ-HOL.*

We consider $\Gamma = x : \{y : Prop \mid y\}$, $P = \pi_1(x)$, and $p = \pi_2(x)$. The judgement $x : \{y : Prop \mid y\} \vdash p : P$ is derivable in PVS-Cert. The erasures of P and Γ are $[P] = x$ and $[\Gamma] = x : Prop$ respectively. As a direct consequence of the consistency of λ-HOL, there exists no λ-HOL proof q such that $x : Prop \vdash q : x$ is derivable in λ-HOL: $[P]$ is empty in $[\Gamma]$ in λ-HOL.

Because of the existence of such counter-examples, the erasure function from PVS-Cert types and expressions is not adapted to prove the conservativity of PVS-Cert over λ-HOL. We will present an alternative translation to fit this purpose. More precisely, the erasure function $[\cdot]$ will be kept to translate PVS-Cert types, but we will define a new translation $[\![\cdot]\!]$ to translate PVS-Cert expressions and stratified contexts. The expected result of conservativity will follow directly from the following properties of this new translation $[\![\cdot]\!]$.

- In the same way as the erasure function $[\cdot]$, any λ-HOL expression or any stratified context in λ-HOL will be translated by itself through the translation $[\![\cdot]\!]$.
- Unlike the erasure function $[\cdot]$, whenever, in PVS-Cert, an expression P is inhabited in a context Γ, then, in λ-HOL, $[\![P]\!]$ is inhabited in the context $[\![\Gamma]\!]$.

In the following, the second property will be referred to as the conservativity of $[\![\cdot]\!]$. It will be proved in Theorem 8.5.1. This proof is constructive: in particular, it is possible, starting from this proof, to extend the translation $[\![\cdot]\!]$ to proofs, and to reformulate conservativity as follows: whenever some judgement $\Gamma \vdash p : P$ is derivable in PVS-Cert (where p is a proof and P an expression), $[\![\Gamma]\!] \vdash [\![p]\!] : [\![P]\!]$ is derivable in λ-HOL. However, the explicit formalization of the underlying translation of proofs would be is much heavier than the presented translation of expressions.

8.1 The choice of the translation

This section is dedicated to informal justifications for the choice of the translation $[\![\cdot]\!]$, which will be entirely formalized in Definition 8.2.5. This translation is obtained by modifying the erasure function $[\cdot]$ as little as possible to ensure the expected conservativity property (whenever a PVS-Cert expression P is inhabited in a context Γ, $[\![P]\!]$ is inhabited in the context $[\![\Gamma]\!]$ in λ-HOL).

The erasure function $[\cdot]$ is not conservative because it erases crucial information about expression variables. Example 8.0.2 illustrates this situation: in the judgement $x : \{y : Prop \mid y\} \vdash \pi_2(x) : \pi_1(x)$, the erasure $x : Prop$ of the original declaration $x : \{y : Prop \mid y\}$ doesn't contain the required information of provability about x. The core idea of the definition of the translation $[\![\cdot]\!]$ is to compensate this erasure of information through the addition of corresponding axioms: in the previous example, the context $x : \{y : Prop \mid y\}$ is translated as a context of the form $x : Prop, h : x$, where h is a fresh proof variable. Using this new axiom h, the translation of the whole judgement is the judgement $x : Prop, h : x \vdash h : x$, which is derivable in λ-HOL.

This erasure of information impacts all expression variables. In a well-formed judgement, an expression variable x is not necessarily introduced through a declaration of the form $x : A$ (as in the previous example): it can be also introduced in a universal quantification of the form $\Pi x : A.P$ or in a λ-abstraction of the form $\lambda x : A.t$.

- The case of universal quantifications can be handled exactly in the same way as the case of declarations. As an example, the derivable PVS-Cert judgement $\vdash (\lambda x : \{y : Prop \mid y\}.\pi_2(x)) : (\Pi x : \{y : Prop \mid y\}.\pi_1(x))$ can be handled following the same strategy as in Example 8.0.2, by translating the universal quantification $\Pi x : \{y : Prop \mid y\}.\pi_1(x)$ as a universal quantification followed by an implication $\Pi x : Prop.\Pi h : x.x$ – which would be written $\forall x : Prop.x \Rightarrow x$ in more usual notations of higher-order logic. Using this new local axiom $h : x$ introduced through this form of *bounded* quantification, the translation of the whole judgement can be defined as the judgement $\vdash (\lambda x : Prop.\lambda h : x.h) : (\Pi x : Prop.\Pi h : x.x)$, which is derivable in λ-HOL.

- In the case of λ-abstractions, the information erased through the translation doesn't require the addition of any axiom. This is due to the fact that, in a more general way, the information required from functions (here, λ-abstractions) to their arguments can always be erased. For instance, the PVS-Cert judgement $x : Prop, h : x \vdash (\lambda z : \{y : Prop \mid y\}.\pi_1(z))\langle x, h\rangle_{\{y:Prop|y\}} : Prop$ is translated as $x : Prop, h : x \vdash (\lambda z : Prop.z)x : Prop$, in which the erasure of the predicate subtype information in $\{y : Prop \mid y\}$ is compensated by the erasure of the coercion $\langle x, h\rangle_{\{y:Prop|y\}}$.

In this setting, the following of this analysis will be focused on the translation of declarations of the form $x : A$, as universal quantifications can be handled accordingly and λ-abstractions can be translated straightforwardly.

- The most simple generalization of Example 8.0.2 is the case of a declaration of the form $x : \{y : A \mid P\}$, where A is a λ-HOL type. In this situation, the erased information is compensated through the introduction of an axiom $(h : [\![P]\!][x/y])$ – applying the translation function recursively on P is necessary here as P does not necessarily belong to λ-HOL.
  Whenever $x : \{y : A \mid P\}$ is well-formed, this allows for instance to translate the derivable judgement $x : \{y : A \mid P\} \vdash \pi_2(x) : P[\pi_1(x)/y]$ as the judgement $x : A, h : [\![P]\!][x/y] \vdash h : [\![P]\!][x/y]$, which is derivable in λ-HOL.

- A more advanced example is the case of a declaration $x : \{y : \{z : A \mid Q\} \mid P\}$, where A is a λ-HOL type. In this situation, the erased information is compensated through the introduction of two axioms, $h_1 : [\![P]\!][x/y]$ and $h_2 : [\![Q]\!][x/z]$.
  In this setting, whenever $x : \{y : \{z : A \mid Q\} \mid P\}$ is well-formed, the following examples illustrate how these axioms can be used.
  - The derivable judgement $x : \{y : \{z : A \mid Q\} \mid P\} \vdash \pi_2(x) : P[\pi_1(x)/y]$ is translated as $x : A, h_1 : [\![P]\!][x/y], h_2 : [\![Q]\!][x/z] \vdash h_1 : [\![P]\!][x/y]$, which is derivable in λ-HOL.
  - The derivable judgement $x : \{y : \{z : A \mid Q\} \mid P\} \vdash \pi_2(\pi_1(x)) : Q[\pi_1(\pi_1(x))/z]$ is translated as $x : A, h_1 : [\![P]\!][x/y], h_2 : [\![Q]\!][x/z] \vdash h_2 : [\![Q]\!][x/z]$, which is derivable in λ-HOL.

- Another interesting example is the case of a declaration $x : \Pi y : A.\{z : B \mid P\}$, where both A and B are λ-HOL types. In this situation, the erased information is compensated through the introduction of one axiom ($h : \Pi y : A.[\![P]\!][xy/z]$): in this situation, the main idea is to handle the occurrence of the predicate subtype *in the range of a function type* through the introduction of a corresponding universal quantification in the emitted axiom.
  Whenever $x : \Pi y : A.\{z : B \mid P\}$ is a well-formed context in which some term t admits the type A, this allows for instance to translate the PVS-Cert derivable judgement $x : \Pi y : A.\{z : B \mid P\} \vdash \pi_2(xt) : P[t/y][\pi_1(xt)/z]$ as the derivable λ-HOL judgement $x : \Pi y : A.B, h : \Pi y : A.[\![P]\!][xy/z] \vdash h[\![t]\!] : [\![P]\!][[\![t]\!]/y][x[\![t]\!]/z]$.

- A last interesting example is the case of a declaration $x : \Pi y : \{z : A \mid P\}.B$, where both A and B are λ-HOL types. This situation corresponds to the case where the occurrence of the predicate subtype is *in the domain of a function type*. In this situation, as in the case of λ-abstractions, no axiom is required: as mentioned previously, the information required from functions (here, x) to their arguments can always be erased.
  For instance, whenever $x : \Pi y : \{z : A \mid P\}.B$ is a well-formed context in which some term t admits the type A and some proof p proves $P[t/z]$, the derivable judgement $x : \Pi y : \{z : A \mid P\}.B \vdash x\langle t, p\rangle_{\{z:A|P\}} : B$ is translated as the judgement $x : \Pi y : A.B \vdash x[\![t]\!] : B$, in which the erasure of the predicate subtype information in the declaration of x is compensated by the erasure of the coercion $\langle t, p\rangle_{\{z:A|P\}}$.

As a generalization of these examples, the translation of an arbitrary declaration $x : A$ involves the addition of *one axiom per strictly positive occurrence of a predicate subtype* inside A: indeed, all predicate subtypes appearing inside a negative occurrence of A can be erased, as illustrated in the last example. For this reason, the notion of strictly positive occurrence will be crucial in the following. We formalize it as follows.

Definition 8.1.1. *A strictly positive occurrence in some type A is defined recursively from the following statements.*

- *If $A = X$ or $A = Prop$, the root of A is its only strictly positive occurrence.*
- *If $A = \Pi x : A_1.A_2$, the strictly positive occurrences of A are its root and the strictly positive occurrences of A_2*
- *If $A = \{x : B \mid P\}$, the strictly positive occurrences of A are its root and the strictly positive occurrences of B*

Although the translation of a declaration $x : A$ contains only one axiom per strictly positive occurrence of a predicate subtype in A, the other occurrences of predicate subtype may also affect the translation of $x : A$, as they may affect the content of these axioms. The following example illustrates this situation.

Example 8.1.1. *We consider a declaration $x : \Pi y : \{z : A \mid P\}.\{z : B \mid Q\}$ where both A and B are λ-HOL types. In this setting, $\{z : B \mid Q\}$ is the only strictly positive*

occurrence of a predicate subtype in $\Pi y : \{z : A \mid P\}.\{z : B \mid Q\}$.

Yet, the translation of this declaration is affected by the two predicates subtypes, as the emitted axiom will correspond to $h_1 : [\![\Pi y : \{z : A \mid P\}.Q]\!][xy/z]$, *which is equal to* $h_1 : \Pi y : A.\Pi h_2 : [\![P]\!][y/z].[\![Q]\!][xy/z]$.

We summarize this whole informal analysis as the following core ideas leading to the definition of the translation $[\![\cdot]\!]$:

- Every declaration of the form $x : A$ is translated as $x : [A], h_1 : P_1, ..., h_n : P_n$, where each axiom P_i corresponds to some strictly positive occurrence of a predicate subtype in A.
- In a similar way, every expression of the form $\Pi x : A.Q$ is translated as $\Pi x : [A].\Pi h_1 : P_1 ... \Pi h_n : P_n.[\![Q]\!]$, where each axiom P_i corresponds to some strictly positive occurrence of a predicate subtype in A.
- In all other cases, the translation $[\![\cdot]\!]$ acts like the erasure function. In particular, $[\![\langle t, p\rangle_A]\!] = [\![\pi_1(t)]\!] = [\![t]\!]$.

8.2 The translation of expressions and stratified contexts

As presented in the previous section, the definition of the translation $[\![\cdot]\!]$ depends crucially on the computation of strictly positive occurrences in a type. These occurrences will be ordered from the root of a type toward its leafs. The following algorithm $\textsc{Occurrence}(A, i)$ computes, whenever it exists, the subterm corresponding to the i-th strictly positive occurrence of a type A. In order to define an algorithm that is stable under α-renaming, we record all bound variables admitting this subterm in their scope, using the binder λ.

Definition 8.2.1. *Given some PVS-Cert type* A *and some index* i, *we define the partial algorithm* $\textsc{Occurrence}(A, i)$ *by recursion on* i. *The result, whenever it exists, has the form* $\lambda x_1 : A_1 ... \lambda x_n : A_n.B$. *The possible cases are the following.*

- $\textsc{Occurrence}(A, 0) = A$
- $\textsc{Occurrence}(Prop, i + 1)$ *fails*
- $\textsc{Occurrence}(X, i + 1)$ *fails*
- $\textsc{Occurrence}(\Pi x : A.B, i + 1)$ *is defined as follows. If* $\textsc{Occurrence}(B, i)$ *returns some term of the form* $\lambda x_1 : A_1 ... \lambda x_n : A_n.B'$ *successfully, return* $\lambda x : A.\textsc{Occurrence}(B, i)$. *Else, fail.*
- $\textsc{Occurrence}(\{x : A \mid P\}, i + 1)$ *is defined as follows. If* $\textsc{Occurrence}(A, i)$ *returns some term of the form* $\lambda x_1 : A_1 ... \lambda x_n : A_n.A$ *successfully, return it. Else, fail.*

Remark 8.2.1. *The decomposition of the result as a term of the form* $\lambda x_1 : A_1 ... \lambda x_n : A_n.B$ *is not ambiguous: as* B *refers to a type, its root cannot be the binder* λ*.*

The first basic property needed to use this definition is to determine, for every type A, the domain on which the algorithm $\textsc{Occurrence}(A, \cdot)$ succeeds. For this purpose, we define a well-suited notion of *size* of a type as follows. We emphasize that this specific notion of size does not correspond to the usual notions of height or length.

Definition 8.2.2. *For any PVS-Cert type* A*, we define* $\textsc{Size}(A)$ *recursively:*

- $\textsc{Size}(X) = 0$
- $\textsc{Size}(Prop) = 0$
- $\textsc{Size}(\{x : A \mid P\}) = \textsc{Size}(A) + 1$
- $\textsc{Size}(\Pi x : A.B) = \textsc{Size}(B) + 1$

Using this algorithm, the domain of the algorithm $\textsc{Occurrence}(A, \cdot)$ is characterized in the following way.

Proposition 8.2.1. $\textsc{Occurrence}(A, i)$ *succeeds if and only if* $i \leq \textsc{Size}(A)$*.*

Proof. The proof is straightforward by induction on A.

□

The translation $[\![\cdot]\!]$ also relies on the definition of new proof variables corresponding to the axioms and hypotheses added through the translation. This definition will be based on the following notion of *injective indexing*.

Definition 8.2.3. *An injective indexing* $h(x, i)$ *is an injective function mapping any expression variable* x *and any index* i *to some proof variable.*

We also present the following notation, which eases the presentation of the translation.

Definition 8.2.4. *Given any context* Γ *of the form* $v_1 : T_1, ..., v_n : T_n$*, we denote any term of the form* $\Pi v_1 : T_1 ... \Pi v_n : T_n.M$ *(resp.* $\lambda v_1 : T_1 ... \lambda v_n : T_n.M$*) as* $\Pi(\Gamma).M$ *(resp.* $\lambda(\Gamma).M$*).*

For instance, the expression $\Pi x : A.\Pi y : B.P$ *can be denoted* $\Pi(x : A, y : B).P$

Using these definitions and notations, the translation of expressions is defined as follows.

Definition 8.2.5. *Given any injective indexing* $h(\cdot, \cdot)$*, we define three partial algorithms:*

- $\textsc{Axioms}(x : A)$*, which translates a PVS-Cert declaration of the form* $x : A$ *into a* λ*-HOL context of the form* $h_1 : P_1, ..., h_n : P_n$

- $[\![t]\!]$, *which translates a PVS-Cert expression into a* λ*-HOL expression*
- $[\![\Gamma]\!]$, *which translates a PVS-Cert stratified context into a* λ*-HOL context*

The dependence to the injective indexing $h(\cdot,\cdot)$ *is left implicit in order to avoid burdening the notations. The two first algorithms are defined by mutual recursion.*

AXIOMS$(x : A)$ *is defined as follows. We first consider* I *the set of all indexes* $i \in \{0, ..., \text{SIZE}(A)\}$ *such that* OCCURRENCE(A, i) *outputs some term of the form* $\lambda x_i^1 : A_i^1 ... \lambda x_i^{n_i} : A_i^{n_i}.\{y_i : B_i \mid P_i\}$. *In this setting,* AXIOMS$(x : A)$ *is the context containing, for all* $i \in I$*, in increasing order, the declarations* $h(x, i) : (\Pi x_i^1 : [A_i^1].\Pi(\text{AXIOMS}(x_i^1 : A_i^1))...\Pi x_i^{n_i} : [A_i^{n_i}].\Pi(\text{AXIOMS}(x_i^{n_i} : A_i^{n_i})).[\![P_i]\!][xx_i^1..., x_i^{n_i}/y_i])$.

The algorithm $[\![t]\!]$ *is defined as follows.*

- $[\![x]\!] = x$
- $[\![\Pi x : A.P]\!] = \Pi x : [A].\Pi(\text{AXIOMS}(x : A)).[\![P]\!]$.
- $[\![\Pi h : P.Q]\!] = \Pi h : [\![P]\!].[\![Q]\!]$
- $[\![\lambda x : A.t]\!] = \lambda x : [A].[\![t]\!]$
- $[\![tu]\!] = [\![t]\!][\![u]\!]$
- $[\![\langle t, M\rangle_A]\!] = [\![t]\!]$
- $[\![\pi_1(t)]\!] = [\![t]\!]$

The algorithm $[\![\Gamma]\!]$ *is defined recursively, using the stratification theorem. The possible cases are the following.*

- $[\![\varnothing]\!] = \varnothing$
- $[\![\Gamma, X : Type]\!] = [\![\Gamma]\!], X : Type$
- $[\![\Gamma, x : A]\!] = [\![\Gamma]\!], x : [A], \text{AXIOMS}(x : A)$
- $[\![\Gamma, h : P]\!] = [\![\Gamma]\!], h : [\![P]\!]$

In the following, we will use without justification the fact that, given any injective indexing $h(\cdot,\cdot)$, the associated algorithms AXIOMS$(x : A)$, $[\![t]\!]$, and $[\![\Gamma]\!]$ are stable under α-renaming. This property is a direct consequence of the fact that λ-HOL types and expressions do not contain any free proof variable, which will be presented in Proposition 8.3.3.

Using the definition of the translation of contexts, the propositions of the form $\Pi x_i^1 : [A_i^1].\Pi(\text{AXIOMS}(x_i^1 : A_i^1))...\Pi x_i^{n_i} : [A_i^{n_i}].\Pi(\text{AXIOMS}(x_i^{n_i} : A_i^{n_i})).[\![P_i]\!][xx_i^1..., x_i^{n_i}/y_i]$ appearing in the definition of AXIOMS will be often designated using the lighter notation

$\Pi[\![x_i^1 : A_i^1, ..., x_i^{n_i} : A_i^{n_i}]\!].[\![P_i]\!][xx_i^1 ..., x_i^{n_i}/y_i]$.

The first property we shall prove is the termination of this translation algorithm.

Proposition 8.2.2. *Given any injective indexing $h(\cdot,\cdot)$, the three associated algorithms* $\textsc{Axioms}(x : A)$, $[\![t]\!]$, *and* $[\![\Gamma]\!]$ *terminate.*

Proof. We first prove that for all n, $\textsc{Axioms}(x : A)$ and $[\![t]\!]$ terminate as long as the height of A (resp. t) is less or equal to n by induction on n. The only difficult case is the termination of $\textsc{Axioms}(x : A)$, in which we conclude by noticing that every recursive call of the form $\textsc{Axioms}(x_i^j : A_i^j)$ or $[\![P_i]\!]$ occurring in this definition is such that A_i^j (resp. P_i) is a strict subterm of A. We conclude the termination of the third algorithm $[\![\Gamma]\!]$ by induction on the length of contexts. □

We can already prove that this translation leaves λ-HOL expressions and λ-HOL stratified contexts unchanged, which is the first result expected from this translation.

Proposition 8.2.3. *The following statements hold.*

- *For every λ-HOL declaration of the form $(x : A)$,* $\textsc{Axioms}(x : A) = \varnothing$
- *For every λ-HOL expression t,* $[\![t]\!] = t$
- *For every λ-HOL stratified context Γ,* $[\![\Gamma]\!] = \Gamma$

Proof. The first property is straightforward as a λ-HOL type does not contain any subterm of the form $\{y : B \mid Q\}$. The other properties follow by induction on their respective arguments. □

Before showing the expected conservativity property of $[\![\cdot]\!]$, which will lead to the conservativity of PVS-Cert over λ-HOL, we will first prove that this translation is sound in the sense that, whenever some judgement of the form $\Gamma \vdash t : A$ is derivable in PVS-Cert, $[\![\Gamma]\!] \vdash [\![t]\!] : [A]$ is derivable in λ-HOL. This soundness property will be proved in Proposition 8.4.4. This property, as well as the conservativity property itself, relies on several properties of λ-HOL, presented in the next section. The expected soundness property will be shown after, followed by the expected conservativity results.

8.3 Properties of λ-HOL

The PTS λ-HOL is defined as a subsystem of PVS-Cert in Definition 4.2.1. In this setting, some properties, such as the free variable theorem or the stratification theorem, hold by definition on λ-HOL derivations.

The soundness and the conservativity of the translation $[\![\cdot]\!]$ are based on several properties of λ-HOL. We first present properties that also hold in PVS-Cert, and that have been already presented for this system.

8.3.1 Subderivations, renaming, thinning, substitution

The following theorem is the adaptation of the subderivations theorem to λ-HOL.

Theorem 8.3.1. *For any derivation of a judgement $\Gamma \vdash WF$ or $\Gamma \vdash M : U$ in λ-HOL, the following properties hold:*

- *For any prefix Γ' of Γ, there exists a subderivation of $\Gamma' \vdash WF$*
- *For any prefix $\Gamma', v : T$ of Γ, there exists a subderivation of $\Gamma' \vdash T : s(v)$*
- *In the case $\Gamma \vdash M : U$, there exists a subderivation of conclusion $\Gamma \vdash M : T$ where $T \equiv_\beta U$ and the last inference step matches some instance of the rule determined by M (*App *for an application, etc.).*

Proof. The proof is similar to the proof given for PVS-Cert. □

The following theorem is the adaptation of the renaming theorem to λ-HOL.

Theorem 8.3.2. *For any λ-HOL derivation of height n and conclusion $\Gamma \vdash M : N$ where the last inference step matches some instance of a rule R, the following holds:*

1. *If $R =$ App and v is its variable parameter, for any $v \in \mathcal{V}_{s(v)} \backslash DV(\Gamma)$, the last inference step matches some instance of R of variable parameter v.*
2. *If $R =$ Prod or $R =$ Lam and if v is its variable parameter, for any $v \in \mathcal{V}_{s(v)} \backslash DV(\Gamma)$, there exists a λ-HOL derivation of height n and conclusion $\Gamma \vdash M : N$ such that the last inference step matches some instance of R of variable parameter v.*

Proof. The proof is similar to the proof given for PVS-Cert. □

The following theorem is the adaptation of the thinning theorem to λ-HOL.

Theorem 8.3.3. *If $\Gamma \vdash M : N$ and $\Delta \vdash WF$ are derivable in λ-HOL with $\Gamma \subseteq \Delta$, then $\Delta \vdash M : N$ is derivable in λ-HOL.*

Proof. The proof is similar to the proof given for PVS-Cert. □

The following theorem is the adaptation of the substitution theorem to λ-HOL.

Theorem 8.3.4. *If*

- *$\Gamma, v : T, \Delta \vdash M : U$ (resp. $\Gamma, v : T, \Delta \vdash WF$) is derivable in λ-HOL*
- *$\Gamma \vdash N : T$ is derivable in λ-HOL*

then $\Gamma, \Delta[N/v] \vdash M[N/v] : U[N/v]$ (resp. $\Gamma, \Delta[N/v] \vdash WF$) is derivable in λ-HOL too.

Proof. The proof is similar to the proof given for PVS-Cert. □

8.3.2 Technical properties in λ-HOL

The following technical properties will be useful in the proof of soundness of the translation function $[\![\cdot]\!]$. They are not specific to λ-HOL. In particular, they could be proved in PVS-Cert as well.

Proposition 8.3.1. *For every judgement of the form* $\Gamma, v_1 : T_1, ..., v_n : T_n \vdash P : Prop$ *such that* $v_1, ..., v_n \in \mathcal{V}_{proofs} \cup \mathcal{V}_{expressions}$*, the two following statements are equivalent.*

- $\Gamma, v_1 : T_1, ..., v_n : T_n \vdash P : Prop$ *is derivable in λ-HOL.*
- *The variables* v_i *are pairwise distinct, they do not appear in* $DV(\Gamma)$*, and* $\Gamma \vdash \Pi(v_1 : T_1, ..., v_n : T_n).P : Prop$ *is derivable in λ-HOL.*

Proof. The proof is done by induction on n. The case $n = 0$ is straightforward. Whenever the statement holds for some n, we prove it for $n + 1$ in the following way.

If $\Gamma, v_1 : T_1, ..., v_{n+1} : T_{n+1} \vdash P : Prop$, then, by the free variable theorem, the variables v_i are pairwise distinct and do not appear in $DV(\Gamma)$. We define $P' = \Pi(v_2 : T_2, ..., v_{n+1} : T_{n+1}).P$. By induction hypothesis with the variables $v_2, ..., v_{n+1}$, the judgement $\Gamma, v_1 : T_1 \vdash P' : Prop$ is derivable in λ-HOL. Using the stratification theorem, one of the following statements holds.

- $v_1 : T_1$ has the form $x : A$ with x an expression variable and A a type. In this case, by the λ-HOL subderivations theorem 8.3.1, $\Gamma \vdash A : Type$ is derivable in λ-HOL. Hence, applying the rule Prod, $\Gamma \vdash \Pi x : A.P' : Prop$ is derivable in λ-HOL, as expected.
- $v_1 : T_1$ has the form $h : Q$ with h an proof variable and Q an expression. In this case, we conclude in a similar way.

On the other hand, if the variables v_i are pairwise distinct, do not appear in $DV(\Gamma)$, and if $\Gamma \vdash \Pi(v_1 : T_1, ..., v_{n+1} : T_{n+1}).P : Prop$ is derivable in λ-HOL, we first apply the induction hypothesis on $v_1, ..., v_n$ to conclude that $\Gamma, v_1 : T_1, ..., v_n : T_n \vdash \Pi v_{n+1} : T_{n+1}.P : Prop$ is derivable in λ-HOL. Then, we conclude the expected result by the λ-HOL subderivations theorem 8.3.1 and the λ-HOL renaming theorem 8.3.2. □

Proposition 8.3.2. *For every judgement of the form* $\Gamma, v_1 : T_1, ..., v_n : T_n \vdash p : P$ *such that* $v_1, ..., v_n \in \mathcal{V}_{proofs} \cup \mathcal{V}_{expressions}$*, the two following statements are equivalent.*

- $\Gamma, v_1 : T_1, ..., v_n : T_n \vdash p : P$ *is derivable in λ-HOL*
- *The variables* v_i *are pairwise distinct, do not appear in* $DV(\Gamma)$*, and* $\Gamma \vdash \lambda(v_1 : T_1, ..., v_n : T_n).p : \Pi(v_1 : T_1, ..., v_n : T_n).P$ *is derivable in λ-HOL*

Proof. The proof is done by induction on n, and similar to the proof of Proposition 8.3.1. The case $n = 0$ is straightforward. Whenever the statement holds for some n, we prove

it for $n+1$ in the following way.

If $\Gamma, v_1 : T_1, ..., v_{n+1} : T_{n+1} \vdash p : P$, then, by the free variable theorem, the variables v_i are pairwise distinct and do not appear in $DV(\Gamma)$. We define $p' = \lambda(v_2 : T_2, ..., v_{n+1} : T_{n+1}).p$ and $P' = \Pi(v_2 : T_2, ..., v_{n+1} : T_{n+1}).P$. By induction hypothesis with the variables $v_2, ..., v_{n+1}$, the judgement $\Gamma, v_1 : T_1 \vdash p' : P'$ is derivable in λ-HOL. Using the stratification theorem, one of the following statements holds.

- $v_1 : T_1$ has the form $x : A$ with x an expression variable and A a type. In this case, by the λ-HOL subderivations theorem 8.3.1, $\Gamma \vdash A : Type$ is derivable in λ-HOL. On the other hand, by Theorem 5.2.3 and the stratification theorem, $\Gamma P' : Prop$ is derivable in λ-HOL. Hence, applying the rule Prod, $\Gamma \vdash \Pi x : A.P' : Prop$ is derivable in λ-HOL and applying the rule Lam, $\Gamma \vdash \lambda x : A.p' : \Pi x : A.P'$ is derivable in λ-HOL, as expected.

- $v_1 : T_1$ has the form $h : Q$ with h an proof variable and Q an expression. In this case, we conclude in a similar way.

On the other hand, if the variables v_i are pairwise distinct, do not appear in $DV(\Gamma)$, and if $\Gamma \vdash \lambda(v_1 : T_1, ..., v_{n+1} : T_{n+1}).p : \Pi(v_1 : T_1, ..., v_{n+1} : T_{n+1}).P$ is derivable in λ-HOL, we first apply the induction hypothesis on $v_1, ..., v_n$ to conclude that $\Gamma, v_1 : T_1, ..., v_n : T_n \vdash \lambda v_{n+1} : T_{n+1}.p : \Pi v_{n+1} : T_{n+1}.P$ is derivable in λ-HOL. Then, we conclude the expected result by the λ-HOL subderivations theorem 8.3.1 and the λ-HOL renaming theorem 8.3.2. □

8.3.3 Specific properties of λ-HOL

The list of useful properties of λ-HOL is ended in this section with properties that are specific to λ-HOL and do not hold in PVS-Cert. The main property which is specific to λ-HOL is the fact that proofs never appear free in λ-HOL types and expressions.

Proposition 8.3.3. *The two following statements hold.*

- *Whenever a type A belongs to the syntax of λ-HOL, it contains no free variable of $\mathcal{V}_{proofs}$ nor $\mathcal{V}_{expressions}$.*

- *Whenever an expression t belongs to the syntax of λ-HOL, it contains no free variable of $\mathcal{V}_{proofs}$.*

Proof. The first statement is straightforward by induction on types, using the fact there is no type of the form $\{x : A \mid P\}$ in λ-HOL.

The second statement is straightforward by induction on expressions, using the first statement and the fact that there is no expression of the form $\langle t, M \rangle_A$ in λ-HOL. □

The following proposition is a direct consequence of Proposition 8.3.3.

Proposition 8.3.4. *We consider any context* Γ*, any expressions* $t, u_1, ..., u_n$ *and any types* $B, A_1, ..., A_n$ *such that the following judgements are derivable in* λ*-HOL.*

- $\Gamma \vdash t : \Pi(x_1 : A_1, ..., x_n : A_n).B$
- $\Gamma \vdash u_i : A_i$ *for all* $i \in \{1, ..., n\}$

Then $\Gamma \vdash tu_1...u_n : B$ *is derivable in* λ*-HOL.*

Proof. The proof is done by induction on n, and similar to the proof of Proposition 8.3.1. The case $n = 0$ is straightforward. Whenever the statement holds for some n, we prove it for $n + 1$ in the following way. We consider judgements of the following form.

- $\Gamma \vdash t : \Pi x_1 : A_1 ... \Pi x_{n+1} : A_{n+1}.B$
- $\Gamma \vdash u_i : A_i$ for all $i \in \{1, ..., n + 1\}$

By induction hypothesis, $\Gamma \vdash tu_1...u_n : \Pi x_{n+1} : A_{n+1}.B$ is derivable in λ-HOL. By Proposition 8.3.3, $B[u_{n+1}/x_{n+1}] = B$. Hence, applying the rule App, $\Gamma \vdash tu_1...u_nu_{n+1} : B$ is derivable, as expected. □

This section is ended with the following strengthening property, which follows from Proposition 8.3.3.

Proposition 8.3.5. *For every contexts* $\Delta \subseteq \Gamma$ *such that* Γ *can be obtained from* Δ *by adding only declarations of the form* $(h : P)$*, the following statements hold.*

- *Whenever* $\Gamma \vdash WF$ *is derivable in* λ*-HOL, so is* $\Delta \vdash WF$
- *Whenever* $\Gamma \vdash Type : Kind$ *is derivable in* λ*-HOL, so is* $\Delta \vdash Type : Kind$
- *Whenever some judgement of the form* $\Gamma \vdash A : Type$ *is derivable in* λ*-HOL, so is* $\Delta \vdash A : Type$
- *Whenever some judgement of the form* $\Gamma \vdash t : A$ *is derivable in* λ*-HOL, so is* $\Delta \vdash t : A$

Proof. The four statements are proved together by induction on the corresponding λ-HOL derivation. The possible cases are the following.

- The case Empty is straightforward.
- The case Decl is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches one of the following instances:
  - $\dfrac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash WF}$ Decl $x \in \mathcal{V}_{expressions} \backslash DV(\Gamma)$

– $\dfrac{\Gamma \vdash Type : Kind}{\Gamma, X : Type \vdash WF}$ DECL $X \in \mathcal{V}_{types} \backslash DV(\Gamma)$

– $\dfrac{\Gamma \vdash P : Prop}{\Gamma, h : P \vdash WF}$ DECL $h \in \mathcal{V}_{proofs} \backslash DV(\Gamma)$

The two first cases are straightforward by induction hypothesis. The last case splits into two subcases, depending on whether the declaration $(h : P)$ belongs to Δ. If $(h : P) \in \Delta$, we conclude directly by induction hypothesis. Else, we first prove by induction hypothesis that $\Delta \vdash P : Prop$ is derivable in λ-HOL, and conclude the expected result by the λ-HOL subderivations theorem 8.3.1.

- The case SORT is straightforward for the two axioms by induction hypothesis.
- The case VAR is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches one of the following instances:

  – $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ VAR $(x : A) \in \Gamma$

  – $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ VAR $(X : Type) \in \Gamma$

  In both cases, the corresponding declaration belongs to Δ, hence we can conclude directly by induction hypothesis.
- The cases PROD, LAM, APP, and CONVERSION are straightforward by induction hypothesis.

□

8.4 Soundness of the translation

This section is dedicated to the proof of the soundness of the translation $[\![\cdot]\!]$, presented in Theorem 8.4.4. This theorem expresses, among others, the fact that whenever some judgement $\Gamma \vdash t : A$ is derivable in PVS-Cert, $[\![\Gamma]\!] \vdash [\![t]\!] : [A]$ is derivable in λ-HOL.

The following proposition on the erasing function $[\cdot]$ will be used in the proof of soundness to handle in particular the case of applications and conversions.

Proposition 8.4.1. *The following statements hold.*

- *For all type A, any expression variable x, and any expression t, $[A[t/x]] = [A]$*
- *For all type A, any expression variable x, and any expression t, $[A][t/x] = [A]$*
- *For all types A and B, $A \equiv_{\beta *} B$ implies $[A] = [B]$.*

Proof. The two first statements are proved straightforwardly by induction on A. The last statement is also proved by induction on A, using Theorem 5.3.2. □

The proof of Proposition 8.4.4 also uses two technical properties of the algorithm Occurrence. The first one is the following.

Proposition 8.4.2. *For any type A and any index i, whenever* Occurrence(A, i) *outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).B$, then $[A] = \Pi(x_1 : [A_1], ..., x_n : [A_n]).[B]$.*

Proof. The proof is straightforward by induction on A. □

The second technical property of the algorithm Occurrence used in the proof of Proposition 8.4.4 is the following.

Proposition 8.4.3. *For any derivation of some judgement of the form $\Gamma \vdash A : Type$ and any index i such that* Occurrence(A, i) *outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).B$ where the bound variables x_i are α-renamed to be pairwise distinct and different from the variables of $DV(\Gamma)$, the judgement $\Gamma, x_1 : A_1, ..., x_n : A_n \vdash B : Type$ admits some derivation whose height is smaller than or equal to the height of the original derivation.*

Proof. The proof is done by induction on A. Discarding the notation A, the possible cases are the following.

- The cases X and $Prop$ are straightforward.
- The case $\Pi x : A.B$ is proved as follows. If $i = 0$, the result is straightforward. If $i > 0$, Occurrence(B, i) outputs a term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).B'$, and Occurrence$(\Pi x : A.B, i) = \lambda(x : A, x_1 : A_1, ..., x_n : A_n).B'$. If the variables $x, x_1, ..., x_n$ are chosen to be pairwise distinct and different from the variables in $DV(\Gamma)$, then, by the subderivations theorem and the renaming theorem, $\Gamma, x : A \vdash B : Type$ is derivable with some derivation of smaller height than the derivation of $\Gamma \vdash \Pi x : A.B : Type$. Hence, we can conclude the expected result directly by induction hypothesis.
- The case $\{x : A \mid P\}$ is proved as follows. If $i = 0$, the result is straightforward. If $i > 0$, Occurrence(A, i) outputs a term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).B'$, and Occurrence$(\{x : A \mid P\}, i)$ = Occurrence(A, i). By the subderivations theorem, $\Gamma \vdash A : Type$ is derivable with some derivation of smaller height than the derivation of $\Gamma \vdash \{x : A \mid P\} : Type$. Hence, if the variables $x_1, ..., x_n$ are chosen to be pairwise distinct and different from the variables in $DV(\Gamma)$, we can conclude the expected result directly by induction hypothesis.

□

The following technical definition will be useful both in the proof of Proposition 8.4.4 and in the proof of the conservativity of $[\![\cdot]\!]$ presented in Theorem 8.5.1.

Definition 8.4.1. *For any number k, we define the following notations.*

- $\textsc{Axioms}(x : A)_{<k}$ *is the restriction of* $\textsc{Axioms}(x : A)$ *to declarations corresponding to* $i \in I \cap \{0, ..., k-1\}$.
- $\textsc{Axioms}(x : A)_{\geq k}$ *is the restriction of* $\textsc{Axioms}(x : A)$ *to declarations corresponding to* $i \in I \backslash \{0, ..., k-1\}$.

The soundness of the translation $[\![\cdot]\!]$ is expressed and proved as follows.

Proposition 8.4.4. *For any context* Γ *and any injective indexing* $h(\cdot, \cdot)$ *such that* $DV(\Gamma)$ *and the image of* $h(\cdot, \cdot)$ *are disjoint, the following statements hold.*

- *If* $\Gamma \vdash WF$ *is derivable,* $[\![\Gamma]\!] \vdash WF$ *is derivable in* λ*-HOL*
- *If* $\Gamma \vdash Type : Kind$ *is derivable,* $[\![\Gamma]\!] \vdash Type : Kind$ *is derivable in* λ*-HOL*
- *If* $\Gamma \vdash A : Type$ *is derivable,* $[\![\Gamma]\!] \vdash [A] : Type$ *is derivable in* λ*-HOL*
- *If* $\Gamma \vdash t : A$ *is derivable,* $[\![\Gamma]\!] \vdash [\![t]\!] : [A]$ *is derivable in* λ*-HOL*

Proof. These four statements are proved together by induction on the height of the corresponding derivation. The possible cases are the following:

- The case Empty is straightforward
- The case Decl is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches one of the following instances:

  – $\dfrac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash WF}$ Decl $x \in \mathcal{V}_{expressions} \backslash DV(\Gamma)$

  This case is the most complex one. In order to prove that the judgement $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A) \vdash WF$ is derivable in λ-HOL, we prove that for all k, $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A)_{<k} \vdash WF$ is derivable in λ-HOL by induction on k and conclude choosing $k = \textsc{Size}(A) + 1$.

  On the one hand, by induction hypothesis, $[\![\Gamma]\!] \vdash [A] : Type$ is derivable in λ-HOL. Moreover, as $DV(\Gamma)$ and $DV([\![\Gamma]\!])$ share the same expression variables, $x \notin DV([\![\Gamma]\!])$. Hence, $[\![\Gamma]\!], x : [A] \vdash WF$ is derivable in λ-HOL. As a consequence $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A)_{<k} \vdash WF$ is derivable in λ-HOL for $k = 0$.

  On the other hand, supposing that $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A)_{<k} \vdash WF$ is derivable in λ-HOL for some k, we prove that $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A)_{<k+1} \vdash WF$ is derivable in λ-HOL in the following way. We consider I the set of all indexes $i \in \{0, ..., \textsc{Size}(A)\}$, such that $\textsc{Occurrence}(A, i)$ outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).\{y : B \mid P\}$. If $k \notin I$, the result

is straightforward as $\textsc{Axioms}(x : A)_{<k+1} = \textsc{Axioms}(x : A)_{<k}$.

If $k \in I$, $\textsc{Occurrence}(A, i)$ outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).\{y : B \mid P\}$. In the following, we use the notation $\Delta = [\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A)_{<k}$. As mentioned in Remark **??**, $\textsc{Axioms}(x : A)$ is stable under α-renaming, hence we can choose the variables $x_1, ..., x_n, y$ to be to be pairwise distinct and different from any variable in Δ. The expected result is the derivability of $\Delta, h(x, k) : \Pi x_1 : [A_1].\Pi(\textsc{Axioms}(x_1 : A_1))...\Pi x_n : [A_n].\Pi(\textsc{Axioms}(x_n : A_n)).[\![P]\!][xx_1...x_n/y] \vdash WF$ in λ-HOL. Using the definition of the translation on contexts, this judgement will be expressed as $\Delta, h(x, k) : \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][xx_1...x_n/y] \vdash WF$.

We first verify that $h(x, k) \notin DV(\Delta)$. As presented in the case $k = 0$, $x \notin DV([\![\Gamma]\!])$. The set $DV([\![\Gamma]\!])$ only contains variables of $DV(\Gamma)$ and variables that can be written $h(z, j)$ with $z \in DV(\Gamma)$. By hypothesis, $h(x, k) \notin DV(\Gamma)$. Moreover, $h(x, k)$ cannot be equal to some $h(z, j)$ with $z \in DV(\Gamma)$ as $x \notin DV(\Gamma)$ and $h(\cdot, \cdot)$ in injective. Hence, $h(x, k) \notin DV([\![\Gamma]\!])$. On the other hand, $h(x, k)$ is different from x as it is not an expression variable, and it is different any variable $h(x, j)$ with $j < k$ by injectivity of $h(\cdot, \cdot)$. Therefore, $h(x, k) \notin DV(\Delta)$. Thus, the expected judgement is derivable in λ-HOL as long as the following judgement is derivable in λ-HOL: $\Delta \vdash \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][xx_1...x_n/y] : Prop$ is derivable in λ-HOL. This latter condition is proved in the following way.

As mentioned earlier, the variables $x_1, ..., x_n$ are chosen to be pairwise distinct and different from any variable in Δ. Moreover, using the injectivity of $h(\cdot, \cdot)$, the proof variables declared in the contexts $\textsc{Axioms}(x_i : A_i)$ are pairwise distinct, and, following the same reasoning as done in the previous paragraph for $h(x, k)$, they do not belong to $DV(\Delta)$. Hence, by Proposition 8.3.1, it is sufficient to prove that $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!] \vdash [\![P]\!][xx_1...x_n/y] : Prop$ is derivable in λ-HOL. By the λ-HOL substitution theorem 8.3.4, this can be done by proving that $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ and $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!] \vdash xx_1...x_n : [B]$ are derivable in λ-HOL. These two conditions are proved successively.

On the one hand, we prove that $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ is derivable in λ-HOL in the following way. As mentioned earlier, the variables $x_1, ..., x_n, y$ are pairwise distinct and different from any variable in Δ, hence they are also different from any variable in Γ. Therefore, by Proposition 8.4.3 followed by the subderivations theorem and the renaming theorem, $\Gamma, x_1 : A_1, ..., x_n : A_n, y : B \vdash P : Prop$ admits some derivation of equal or smaller height than the derivation of $\Gamma \vdash A : Type$. Hence, applying the induction hypothesis followed by Proposition 8.3.5, the judge-

ment $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ is derivable in λ-HOL. We conclude the expected result by the following combination of the λ-HOL substitution theorem 8.3.3 and Proposition 8.3.1. By Proposition 8.3.1, $[\![\Gamma]\!] \vdash \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].\Pi y : [B].[\![P]\!] : Prop$ is derivable in λ-HOL. By induction hypothesis, $\Delta \vdash WF$ is derivable in λ-HOL: hence, by the λ-HOL substitution theorem 8.3.3, $\Delta \vdash \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].\Pi y : [B].[\![P]\!] : Prop$ is derivable in λ-HOL. As the variables $x_1, ..., x_n, y$ are different from any variable in Δ, we can apply Proposition 8.3.1 in the opposite direction to conclude that $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ is derivable in λ-HOL.

On the other hand, we prove that $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!] \vdash xx_1...x_n : [B]$ is derivable in λ-HOL in the following way. As proved in the previous paragraph, the judgement $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ is derivable in λ-HOL. Hence, by the subderivations theorem, the judgement $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!] \vdash WF$ is derivable in λ-HOL. Applying the rule VAR and using Proposition 8.4.2, $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!] \vdash x : \Pi x_1 : [A_1]...\Pi x_n : [A_n].[B]$ is derivable in λ-HOL, as well as all judgements $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!] \vdash x_i : [A_i]$. Hence, by Proposition 8.3.4, $\Delta, [\![x_1 : A_1, ..., x_n : A_n]\!] \vdash xx_1...x_n : [B]$ is derivable in λ-HOL.

- $\dfrac{\Gamma \vdash Type : Kind}{\Gamma, X : Type \vdash WF}$ DECL $X \in \mathcal{V}_{types} \backslash DV(\Gamma)$

  As the set of variables $DV([\![\Gamma]\!])$ only contains variables of $DV(\Gamma)$ and variables that can be written $h(y, i)$ for $y \in DV(\Gamma)$, $X \notin DV([\![\Gamma]\!])$, which allows to conclude the expected result by induction hypothesis.
- $\dfrac{\Gamma \vdash P : Prop}{\Gamma, h : P \vdash WF}$ DECL $h \in \mathcal{V}_{proofs} \backslash DV(\Gamma)$

  The set of variables $DV([\![\Gamma]\!])$ only contains variables of $DV(\Gamma)$ and variables that can be written $h(y, i)$. By hypothesis on $h(\cdot, \cdot)$, h is not equal to any variable $h(y, i)$, hence $h \notin DV([\![\Gamma]\!])$. This allows to conclude the expected result by induction hypothesis.

- The case SORT is straightforward for the two axioms by induction hypothesis.
- The case VAR is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches one of the following instances:
  - $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ VAR $(x : A) \in \Gamma$
  - $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ VAR $(X : Type) \in \Gamma$

Both cases are straightforward by induction hypothesis.

- The case PROD is proved as follows. Discarding the notations in the original statement, using the stratification theorem, we can suppose that the last inference step matches one of the following instances.

  $$- \ \frac{\Gamma \vdash P : Prop \qquad \Gamma, h : P \vdash Q : Prop}{\Gamma \vdash \Pi h : P.Q : Prop}\ \textsc{Prod}$$

  In this case, by the renaming theorem, we can suppose without loss of generality that h does not belong to the image of $h(\cdot, \cdot)$ as the original derivation can be transformed to a derivation of the same height and conclusion matching this requirement. Using this hypothesis, we conclude directly by induction hypothesis.

  $$- \ \frac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \Pi x : A.P : Prop}\ \textsc{Prod}$$

  In this case, by induction hypothesis, $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A) \vdash [\![P]\!] : Prop$ is derivable in λ-HOL. We define the notation $h_1 : P_1, ..., h_n : P_n$ for $\textsc{Axioms}(x : A)$. As the context $x : [A], h_1 : P_1, ..., h_n : P_n$ contains no declaration of the form $(X : Type)$, the Proposition 8.3.1 can be applied to conclude that $[\![\Gamma]\!] \vdash \Pi x : [A].\Pi h_1 : P_1 ... \Pi h_n : P_n.[\![P]\!] : Prop$ is derivable, as expected.

  $$- \ \frac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}\ \textsc{Prod}$$

  In this case, by induction hypothesis, $[\![\Gamma]\!] \vdash [A] : Type$ and $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A) \vdash [B] : Type$ are both derivable in λ-HOL. By Proposition 8.3.5, $[\![\Gamma]\!], x : [A] \vdash [B] : Type$ is derivable in λ-HOL, which allows to conclude the expected result applying the rule PROD.

- The case SUBTYPE is straightforward by induction hypothesis and the stratification theorem.

- The case LAM is proved in the following way. Discarding the notations in the original statement, using the stratification theorem, the last inference step matches some instance of the form

  $$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x : A.B : Type}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}\ \textsc{Lam}$$

By induction hypothesis, $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A) \vdash [\![t]\!] : [B]$ and $[\![\Gamma]\!] \vdash \Pi x : [A].[B] : \mathit{Type}$ are derivable. By Proposition 8.3.5, $[\![\Gamma]\!], x : [A] \vdash [\![t]\!] : [B]$ is derivable in λ-HOL, which allows to conclude the expected result applying the rule Lam.

- The case App is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, we can suppose that the last inference step matches some instance of the form

$$\frac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}\ \textsc{App}$$

  By induction hypothesis and the application of the rule App, the judgement $[\![\Gamma]\!] \vdash [\![tu]\!] : [B][[\![u]\!]/x]$ is derivable in λ-HOL. We conclude the expected result by Proposition 8.4.1, as $[B[u/x]] = [B] = [B][[\![u]\!]/x]]$.

- The cases Pair and Proj1 are straightforward by induction hypothesis and the stratification theorem.

- The case Proj2 doesn't occur by the stratification theorem.

- The case Conversion is proved in the following way. Discarding the notations in the original statement and using the stratification theorem, the last inference step matches some rule instance of the form

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \mathit{Type}}{\Gamma \vdash t : B}\ \textsc{Conversion}\ A \equiv_{\beta *} B$$

  By induction hypothesis, $[\![\Gamma]\!] \vdash [\![t]\!] : [A]$ and $[\![\Gamma]\!] \vdash [B] : \mathit{Type}$ are derivable in λ-HOL. By Proposition 8.4.1, $[A] \equiv_\beta [B]$. Hence, applying the λ-HOL conversion rule, $[\![\Gamma]\!] \vdash [\![t]\!] : [B]$ is derivable in λ-HOL.

□

8.5 Conservativity of the translation

The main proof of conservativity uses two properties of the translation $[\![\cdot]\!]$, to handle in particular the case of conversions and applications. The first proposition on the translation $[\![t]\!]$ is the following.

Proposition 8.5.1. *For any expression variable x and any expression u, the two following statements hold.*

- *For any expression t, $[\![t[u/x]]\!] = [\![t]\!][[\![u]\!]/x]$.*

- *For any type A and any expression variable $y \neq x$,* $\textsc{Axioms}(y : A[u/y]) = \textsc{Axioms}(y : A)[[\![u]\!]/x]$

This proposition relies on the following technical property of Occurrence, which will be also useful in the proof of conservativity.

Lemma 8.5.1. *For any type A, any expression t, any expression variable x, and any index i,* $\textsc{Occurrence}(A, i)$ *outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).B$ if and only if* $\textsc{Occurrence}(A[t/x], i)$ *outputs some term of the form $\lambda(x_1 : A'_1, ..., x_n : A'_n).B'$. Moreover, in this case, choosing the bound variables $x_1, ..., x_n$ to be different from x and outside $FV(t)$, $B' = B[t/x]$ and $A'_i = A_i[t/x]$ for all $i \in \{1, ..., n\}$.*

Proof. The proof is straightforward by induction on A. □

Using this lemma, Proposition 8.5.1 is proved as follows.

Proof. [Proposition 8.5.1] The two statements are proved together by induction on the height of t in the first statement, and A in the second statement.

In the first statement, the only difficult case is $t = \Pi y : A.P$: all other cases are straightforward by induction hypothesis and Proposition 8.4.1. The case $t = \Pi y : A.P$ is proved as follows. We choose y to ensure that $x \neq y$ and $y \notin FV([\![u]\!])$. Moreover, as the variables declared in $\textsc{Axioms}(y : A)$ are only proof variables, they are distinct form x and, by Proposition 8.3.3, they cannot occur free in the λ-HOL expressions appearing $\textsc{Axioms}(y : A)$. Hence, $[\![t]\!][[\![u]\!]/x] = \Pi y : [A][[\![u]\!]/x].\Pi(\textsc{Axioms}(y : A)[[\![u]\!]/x]).[\![P]\!][[\![u]\!]/x]$, and we conclude the expected result by Proposition 8.4.1 and by induction hypothesis.

The case corresponding to the second statement is proved as follows. By straightforward induction on A, $\textsc{Size}(A[u/x]) = \textsc{Size}(A)$. We consider I (resp. I') the set of all indexes $i \in \{1, ..., \textsc{Size}(A)\}$ such that $\textsc{Occurrence}(A, i)$ (resp. $\textsc{Occurrence}(A[u/x], i)$) outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).\{y : B \mid P\}$. Using Lemma 8.5.1, we first prove that $I = I'$. We can conclude the expected result as long as we prove that, for all $i \in I$, defining $h : P$ (resp. $h : P'$) the corresponding declaration in $\textsc{Axioms}(y : A)$ (resp. $\textsc{Axioms}(y : A[u/x])$), $P' = [[\![u]\!]/x]$. This is done as follows.

We consider $i \in I$. $\textsc{Occurrence}(A, i)$ has the form $\lambda(x_1 : A_1, ..., x_n : A_n).\{z : B \mid P\}$. We choose $x_1, ..., x_n, z$ different from x and outside $FV([\![u]\!])$. By Lemma 8.5.1 again, the expected result becomes the equality between $\Pi[\![x_1 : A_1[u/x], ..., x_n : A_n[u/x]]\!].[\![P[u/x]]\!][yx_1..., x_n/z]$ and $(\Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][yx_1..., x_n/z])[[\![u]\!]/x]$. On the one hand, the variables $x_1, ..., x_n$ are different from x and outside $FV([\![u]\!])$. On the other hand, so are the proof variables declared in $[\![x_1 : A_1, ..., x_n : A_n]\!]$ by Proposition 8.3.3. Hence, writing $[\![x_1 : A_1, ..., x_n : A_n]\!]$ as $x_1 : [A_1], \textsc{Axioms}(x_1 : A_1), ..., x_n : [A_n], \textsc{Axioms}(x_n : A_n)$, applying Proposition 8.4.1 and the induction hypothesis on the types $A_1, ..., A_n$ (which are strict subterms of A), we can conclude the expected equality

as long as we prove $[\![P[u/x]]\!][yx_1...,x_n/z] = [\![P]\!][yx_1...,x_n/z][[\![u]\!]/x]$. This is done as follows.

As $z \notin FV([\![u]\!])$ and $z \neq x$, using Lemma 5.1.1 the expressions $[\![P]\!][yx_1...,x_n/z][[\![u]\!]/x]$ and $[\![P]\!][[\![u]\!]/x][(yx_1...,x_n)[[\![u]\!]/x]/z]$ are equal. Moreover, the $x_1,...,x_n$ are distinct from x, and, by hypothesis, so is y. Hence, $[\![P]\!][yx_1...,x_n/z][[\![u]\!]/x] = [\![P]\!][[\![u]\!]/x][yx_1...,x_n/z]$, and we conclude the expected result by induction hypothesis on the expression P, which is a strict subterm of A. □

The second proposition on the translation $[\![t]\!]$ is the following.

Proposition 8.5.2. *The two following statements hold.*

- *For any expression t and t' such that $t \equiv_{\beta *} t'$, $[\![t]\!] \equiv_\beta [\![t']\!]$*
- *For any expression variable x, for any types A and A' such that $A \equiv_{\beta *} A'$, one can write* Axioms$(x : A) = h_1 : P_1, ..., h_n : P_n$ *and* Axioms$(x : A') = h_1 : P'_1, ..., h_n : P'_n$ *with, for all $i \in \{1, ..., n\}$, $P_i \equiv_\beta P'_i$*

The proof of this proposition relies first on the following technical property of the algorithm Occurrence.

Lemma 8.5.2. *For any types A and A' such that $A \to_{\beta *} A'$, and any index i, the algorithm* Occurrence(A, i) *outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).B$ if and only if* Occurrence(A', i) *outputs some term of the form $\lambda(x_1 : A'_1, ..., x_n : A'_n).B'$. Moreover, in this case, either $B' = B$ or $B \to_{\beta *} B'$, and, for all $i \in \{1, ..., n\}$, either $A'_i = A_i$ or $A'_i \to_{\beta *} A_i$.*

Proof. The proof is straightforward by induction on A. □

The proof of Proposition 8.5.2 also relies on the following lemma.

Lemma 8.5.3. *The two following statements hold.*

- *For any expression t and t' such that $t \to_{\beta *} t'$, $[\![t]\!] \equiv_\beta [\![t']\!]$*
- *For any expression variable x, for any types A and A' such that $A \to_{\beta *} A'$, one can write* Axioms$(x : A) = h_1 : P_1, ..., h_n : P_n$ *and* Axioms$(x : A') = h_1 : P'_1, ..., h_n : P'_n$ *with, for all $i \in \{1, ..., n\}$, $P_i \equiv_\beta P'_i$*

Proof. The two statements are proved together by induction on the height of t in the first statement, and A in the second statement. In the first statement, all cases are straightforward by induction hypothesis and Proposition 8.4.1. The case corresponding to the second statement is proved as follows.

By straightforward induction on A, Size(A) = Size(A'). We consider I (resp. I') the set of all indexes $i \in \{1, ..., \text{Size}(A)\}$ such that Occurrence(A, i) (resp. such that Occurrence(A', i)) outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).\{y : B \mid P\}$.

Using Lemma 8.5.2, we first prove that $I = I'$. We can conclude the expected result as long as we prove that, for all $i \in I$, defining $h : P$ (resp. $h : P'$) the corresponding declaration in $\text{AXIOMS}(y : A)$ (resp. $\text{AXIOMS}(y : A')$), $P \equiv_\beta P'$. This is done as follows.

We consider $i \in I$. $\text{OCCURRENCE}(A, i)$ has the form $\lambda(x_1 : A_1, ..., x_n : A_n).\{z : B \mid P\}$. Using Lemma 8.5.2 again, $\text{OCCURRENCE}(A', i)$ has the form $\lambda(x_1 : A'_1, ..., x_n : A'_n).\{z : B' \mid P'\}$ with either $P' = P$ or $B \rightarrow_{\beta *} B'$, and, for all $i \in \{1, ..., n\}$, either $A'_i = A_i$ or $A'_i \rightarrow_{\beta *} A_i$. In this setting, the expected result becomes the equivalence between $\Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][yx_1..., x_n/z]$ and $\Pi[\![x_1 : A'_1, ..., x_n : A'_n]\!].[\![P']\!][yx_1..., x_n/z]$ using the conversion $\equiv_\beta$. Writing $[\![x_1 : A_1, ..., x_n : A_n]\!]$ as $x_1 : [A_1], \text{AXIOMS}(x_1 : A_1), ..., x_n : [A_n], \text{AXIOMS}(x_n : A_n)$, we conclude the expected result applying Proposition 8.4.1 and the induction hypothesis on the types $A_1, ..., A_n$ and the expression P (which are all strict subterms of A). □

Using this lemma, Proposition 8.5.2 is proved as follows.

Proof. [Proposition 8.5.2] The first statement is proved as follows. By Proposition 5.4.2 followed by Lemma 8.5.3, we first prove that whenever an expression t is such that $t \rightarrow_{\beta *} M$ for some term M, then M is an expression and $[\![t]\!] \equiv_\beta [\![M]\!]$. We conclude that whenever an expression t is such that $t \twoheadrightarrow_{\beta *} M$ for some term M, then M is an expression and $[\![t]\!] \equiv_\beta [\![M]\!]$ by induction on the length of the reduction $t \twoheadrightarrow_{\beta *} M$. Finally, we conclude the expected result by the Church-Rosser theorem.

The second statement is proved as follows. By Proposition 5.4.2 followed by Lemma 8.5.3, we first prove that whenever a type A is such that $A \rightarrow_{\beta *} M$ for some term M, then M is a type and one can write $\text{AXIOMS}(x : A) = h_1 : P_1, ..., h_n : P_n$ and $\text{AXIOMS}(x : M) = h_1 : P'_1, ..., h_n : P'_n$ with, for all $i \in \{1, ..., n\}$, $P_i \equiv_\beta P'_i$. We conclude that whenever a type A is such that $A \twoheadrightarrow_{\beta *} M$ for some term M, then M is a type and the previous property holds for $\text{AXIOMS}(x : A)$ and $\text{AXIOMS}(x : M)$ by induction on the length of the reduction $A \twoheadrightarrow_{\beta *} M$. Finally, we conclude the expected result by the Church-Rosser theorem. □

We will also use the following consequence of Lemma 8.5.2 in the proof of conservativity.

Lemma 8.5.4. *For any types A and A' such that $A \equiv_{\beta *} A'$, and any index i, the algorithm $\text{OCCURRENCE}(A, i)$ outputs some term of the form $\lambda(x_1 : A_1, ..., x_n : A_n).B$ if and only if $\text{OCCURRENCE}(A', i)$ outputs some term of the form $\lambda(x_1 : A'_1, ..., x_n : A'_n).B'$. Moreover, in this case, $B \equiv_{\beta *} B'$, and, for all $i \in \{1, ..., n\}$, $A'_i \equiv_{\beta *} A_i$.*

Proof. By Proposition 5.4.2 followed by Lemma 8.5.3, we first prove that whenever a type A is such that $A \rightarrow_{\beta *} M$ for some term M, then M is a type and the expected statement holds for A and M.

We conclude that whenever a type A is such that $A \twoheadrightarrow_{\beta *} M$ for some term M, then M is a type and the expected statement holds for A and M by induction on the length of the reduction $t \twoheadrightarrow_{\beta *} M$. Finally, we conclude the expected result in the general case $A \equiv_{\beta *} A'$ by the Church-Rosser theorem. □

Proposition 8.5.3. *For any context Γ and any injective indexing $h(\cdot,\cdot)$ such that $DV(\Gamma)$ and the image of $h(\cdot,\cdot)$ are disjoint, the following statement hold. Whenever $\Gamma \vdash t : A$ is derivable and* Occurrence(A, i) *has the form $\lambda(x_1 : A_1, ..., x_n : A_n).\{y : B \mid P\}$ for some index i, using the notations of the translations of contexts, the judgement $[\![\Gamma]\!] \vdash \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][[\![t]\!]x_1...x_n/y] : Prop$ is derivable in λ-HOL.*

Proof. We choose the variables $x_1, ..., x_n, y$ pairwise distinct and outside $DV([\![\Gamma]\!])$. By Proposition 8.3.1, it is sufficient to prove that $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!] \vdash [\![P]\!][[\![t]\!]x_1...x_n/y] : Prop$ is derivable in λ-HOL. By the λ-HOL substitution theorem 8.3.4, this can be done by proving that $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ and $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!] \vdash [\![t]\!]x_1...x_n : [B]$ are derivable in λ-HOL. These two conditions are proved successively.

On the one hand, we prove that $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ is derivable in λ-HOL in the following way. By Theorem 5.2.3 and the stratification theorem, $\Gamma \vdash A : Type$ is derivable in PVS-Cert. As mentioned earlier, the variables $x_1, ..., x_n, y$ are pairwise distinct and different from any variable in $DV([\![\Gamma]\!])$, hence they are also different from any variable in $DV(\Gamma)$. Therefore, by Proposition 8.4.3 followed by the subderivations theorem and the renaming theorem, $\Gamma, x_1 : A_1, ..., x_n : A_n, y : B \vdash P : Prop$ is derivable in PVS-Cert. As a consequence, by the soundness proposition 8.4.4 followed by Proposition 8.3.5, the judgement $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ is derivable in λ-HOL, as expected.

On the other hand, we prove that $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!] \vdash [\![t]\!]x_1...x_n : [B]$ is derivable in λ-HOL in the following way. As proved in the previous paragraph, the judgement $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!], y : [B] \vdash [\![P]\!] : Prop$ is derivable in λ-HOL. Hence, by the subderivations theorem, the judgement $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!] \vdash WF$ is derivable in λ-HOL. By the soundness proposition 8.4.4 and Proposition 8.4.2, $[\![\Gamma]\!] \vdash [\![t]\!] : \Pi x_1 : [A_1]...\Pi x_n : [A_n].[B]$ is derivable in λ-HOL. Hence, by the λ-HOL substitution theorem 8.3.3, $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!] \vdash [\![t]\!] : \Pi x_1 : [A_1]...\Pi x_n : [A_n].[B]$ is derivable in λ-HOL. On the other hand, using the rule Var, all judgements $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!] \vdash x_i : [A_i]$ are derivable in λ-HOL. Hence, by Proposition 8.3.4, $[\![\Gamma, x_1 : A_1, ..., x_n : A_n]\!] \vdash [\![t]\!]x_1...x_n : [B]$ is derivable in λ-HOL.

□

The conservativity property of the translation $[\![\cdot]\!]$ is expressed and proved as follows.

Theorem 8.5.1. *For any context Γ and any injective indexing $h(\cdot,\cdot)$ such that $DV(\Gamma)$ and the image of $h(\cdot,\cdot)$ are disjoint, the following statements hold.*

- *If* $\Gamma \vdash p : P$ *is derivable, there exists a proof term* q *such that* $[\![\Gamma]\!] \vdash q : [\![P]\!]$ *is derivable in* λ*-HOL*
- *If* $\Gamma \vdash t : A$ *is derivable and* $\textsc{Occurrence}(A, i)$ *has the form* $\lambda x_1 : A_1 ... \lambda x_n : A_n.\{y : B \mid P\}$ *for some index* i*, there exists a proof term* p *such that the judgement* $[\![\Gamma]\!] \vdash p : \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][[\![t]\!]x_1 ... x_n/y]$ *is derivable in* λ*-HOL.*

Proof. These two statements are proved together by induction on the height of the corresponding derivation. Discarding the notations p and P, the possible cases are the following.

- The cases Empty and Decl cannot occur by hypothesis.
- The case Sort cannot occur by the stratification theorem.
- The case Var is proved in the following way. By the stratification theorem, the last inference step matches one of the following instances.

  – $\dfrac{\Gamma \vdash WF}{\Gamma \vdash h : P}\ \textsc{Var}\ (h : P) \in \Gamma$

  As $(h : P) \in \Gamma$, $(h : [\![P]\!]) \in [\![\Gamma]\!]$. By Proposition 8.4.4, $[\![\Gamma]\!] \vdash WF$ is derivable in λ-HOL, hence $[\![\Gamma]\!] \vdash h : [\![P]\!]$ is derivable in λ-HOL.

  – $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}\ \textsc{Var}\ (x : A) \in \Gamma$

  We consider some index i such that $\textsc{Occurrence}(A, i)$ has the form $\lambda x_1 : A_1 ... \lambda x_n : A_n.\{y : B \mid P\}$. We find the expected proof in the following way.

  By Proposition 8.2.1, $i \leq \textsc{Size}(A)$. In this setting, as $(x : A) \in \Gamma$, $h(x, i) : \Pi[\![x_1 : A_1, ..., \Pi x_n : A_n]\!].[\![P]\!][x x_1 ... x_n/y] \in [\![\Gamma]\!]$. By Proposition 8.4.4, $[\![\Gamma]\!] \vdash WF$ is derivable in λ-HOL. Hence, using the Var rule, $[\![\Gamma]\!] \vdash h(x, i) : \Pi[\![x_1 : A_1, ..., \Pi x_n : A_n]\!].[\![P]\!][x x_1 ... x_n/y]$ is derivable in λ-HOL.

- The case Prod is proved in the following way. Using the stratification theorem, we can suppose that the last inference step matches one of the following instances.

  – $\dfrac{\Gamma \vdash P : Prop \qquad \Gamma, h : P \vdash Q : Prop}{\Gamma \vdash \Pi h : P.Q : Prop}\ \textsc{Prod}$

  $\textsc{Occurrence}(Prop, i)$ is well-defined only if $i = 0$ and, in this case, its value is $Prop$. In this setting, the expected result is straightforward.

  – $\dfrac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \Pi x : A.P : Prop}\ \textsc{Prod}$

$\textsc{Occurrence}(Prop, i)$ is well-defined only if $i = 0$ and, in this case, its value is $Prop$. In this setting, the expected result is straightforward.

- The case $\textsc{Lam}$ is proved in the following way. Using the stratification theorem, we can suppose that the last inference step matches one of the following instances.

$$_\ \frac{\Gamma, h : P \vdash q : Q \qquad \Gamma \vdash \Pi h : P.Q : Prop}{\Gamma \vdash \lambda h : P.q : \Pi h : P.Q}\ \textsc{Lam}$$

By the renaming theorem, we can suppose without loss of generality that h does not belong to the image of $h(\cdot, \cdot)$ as the original derivation can be transformed to a derivation of the same height and conclusion matching this requirement. In this setting, we can conclude the expected result directly applying the induction hypothesis on the first premise and Proposition 8.4.4 on the second one.

$$_\ \frac{\Gamma, x : A \vdash p : P \qquad \Gamma \vdash \Pi x : A.P : Prop}{\Gamma \vdash \lambda x : A.p : \Pi x : A.P}\ \textsc{Lam}$$

In this case, by induction hypothesis, there exists some proof q such that $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A) \vdash q : [\![P]\!]$ is derivable in λ-HOL. Hence, by Proposition 8.3.2, there exists a proof q' such that $[\![\Gamma]\!] \vdash q' : [\![\Pi x : A.P]\!]$ is derivable in λ-HOL.

$$_\ \frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x : A.B : Type}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}\ \textsc{Lam}$$

We consider some index i such that $\textsc{Occurrence}(\Pi x : A.B, i)$ has the form $\lambda x_1 : A_1 ... \lambda x_n : A_n.\{y : B' \mid P\}$: in this case, $i > 0$, and $\textsc{Occurrence}(\Pi x : A.B, i) = \lambda x : A.\textsc{Occurrence}(B, i - 1)$. Hence, $A_1 = A$ and, choosing $x_1 = x$, $\textsc{Occurrence}(B, i - 1) = \lambda x_2 : A_2 ... \lambda x_n : A_n.\{y : B' \mid P\}$. We find the expected proof in the following way.

By induction hypothesis, there exists a proof p such that $[\![\Gamma]\!], x : [A], \textsc{Axioms}(x : A) \vdash p : \Pi [\![x_2 : A_2, ..., x_n : A_n]\!].[\![P]\!][[\![t]\!]x_2 ... x_n/y]$ is derivable in λ-HOL. Hence, by Proposition 8.3.2, there exists a proof q such that $[\![\Gamma]\!] \vdash q : \Pi [\![x : A, x_2 : A_2, ..., x_n : A_n]\!].[\![P]\!][[\![t]\!]x_2 ... x_n/y]$ is derivable in λ-HOL.

We notice that $[\![t]\!]x_2 ... x_n \equiv_{\beta *} [\![\lambda x : A.t]\!]x x_2 ... x_n$. Hence, we can conclude the expected result by conversion as long as we prove that $[\![\Gamma]\!] \vdash \Pi [\![x : A, x_2 : A_2, ..., x_n : A_n]\!].[\![P]\!][[\![\lambda x : A.t]\!]x x_2 ... x_n/y] : Prop$ is derivable in λ-HOL, which is the case by Proposition 8.5.3.

- The case $\textsc{App}$ is the most complex one. It is proved in the following way. Using the stratification theorem, we can suppose that the last inference step matches one

of the following instances.

– $\dfrac{\Gamma \vdash p : \Pi h : P.Q \qquad \Gamma \vdash q : P}{\Gamma \vdash pq : Q[q/h]}$ App

In this case, we conclude directly by induction hypothesis and Proposition 8.5.1.

– $\dfrac{\Gamma \vdash p : \Pi x : A.P \qquad \Gamma \vdash t : A}{\Gamma \vdash pt : P[t/x]}$ App

By the renaming theorem, we can suppose without loss of generality that $x \notin DV(\Gamma)$. By induction hypothesis, there exists a proof q such that $[\![\Gamma]\!] \vdash q : \Pi x : [A].\Pi(\textsc{Axioms}(x : A)).[\![P]\!]$ is derivable in λ-HOL. By Proposition 8.4.4, $[\![\Gamma]\!] \vdash [\![t]\!] : [A]$ is derivable in λ-HOL. Hence, $[\![\Gamma]\!] \vdash q[\![t]\!] : (\Pi(\textsc{Axioms}(x : A)).[\![P]\!])[[\![t]\!]/x]$ is derivable in λ-HOL.

Starting from this result, we prove by induction that for all k, there exists a proof q_k such that $[\![\Gamma]\!] \vdash q_k : (\Pi(\textsc{Axioms}(x : A)_{\geq k}).[\![P]\!])[[\![t]\!]/x]$ is derivable in λ-HOL. The case $k = 0$ corresponds to the previous derivation with $q_0 = q[\![t]\!]$.

Supposing that $[\![\Gamma]\!] \vdash q_k : (\Pi(\textsc{Axioms}(x : A)_{\geq k}).[\![P]\!])[[\![t]\!]/x]$ is derivable in λ-HOL for some k, we find a proof q_{k+1} in the following way. We consider I the set of all indexes $i \in \{0, ..., \textsc{Size}(A)\}$ such that $\textsc{Occurrence}(A, i)$ outputs some term of the form $\lambda x_1 : A_1 ... \lambda x_n : A_n.\{y : B \mid Q\}$. If $k \notin I$, the expected result is straightforward, taking $q_{k+1} = q_k$.

If $k \in I$, $\textsc{Occurrence}(A, k)$ outputs some term of the form $\lambda x_1 : A_1 ... \lambda x_n : A_n.\{y : B \mid Q\}$. We choose $x_1, ..., x_n, y$ pairwise distinct, different from x, and outside $DV(\Gamma)$. In the following, we use the notation $P' = (\Pi(\textsc{Axioms}(x : A)_{\geq k+1}).[\![P]\!])[[\![t]\!]/x]$. The proof variable $h(x, k)$ is distinct from the expression variable x, and cannot occur free in $[\![t]\!]$ by Proposition 8.3.3. Hence, the expression $(\Pi(\textsc{Axioms}(x : A)_{\geq k}).[\![P]\!])[[\![t]\!]/x]$ corresponds to the following one: $\Pi h(x, k) : (\Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![Q]\!][xx_1...x_n/y])[[\![t]\!]/x].P'$.

As the variables $x_1, ..., x_n, y$ are pairwise distinct and outside $DV(\Gamma)$, we can use successively Theorem 5.2.3, Proposition 8.4.3, the subderivations theorem, and the renaming theorem to conclude that $\Gamma, x_1 : A_1, ..., x_n : A_n, y : B \vdash Q : Prop$ is derivable in PVS-Cert. Hence, by Proposition 8.4.4, $[\![\Gamma, x_1 : A_1, ..., x_n : A_n, y : B]\!] \vdash [\![Q]\!] : Prop$ is derivable in λ-HOL. As x is an expression variable outside $DV(\Gamma, x_1 : A_1, ..., x_n : A_n, y : B)$, it is outside $DV([\![\Gamma, x_1 : A_1, ..., x_n : A_n, y : B]\!])$. Hence, by the free variable theorem, $x \notin FV([\![\Gamma, x_1 : A_1, ..., x_n : A_n, y : B]\!])$ and $x \notin FV([\![Q]\!])$.

As a consequence of the first of these two statements, $(\Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![Q]\!][xx_1...x_n/y])[[\![t]\!]/x] = \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![Q]\!][xx_1...x_n/y][[\![t]\!]/x]$. By Proposition 8.4.4, $[\![\Gamma]\!] \vdash [\![t]\!] : [A]$ is derivable in λ-HOL. As y is an expression variable outside $DV(\Gamma)$, it is outside $DV([\![\Gamma]\!])$. Hence, by the free variable theorem, $y \notin FV([\![t]\!])$. Moreover, the variables $x_1, ..., x_n, y$ are different from x, and we proved $x \notin FV([\![Q]\!])$, hence we conclude using Lemma 5.1.1 that $[\![Q]\!][xx_1...x_n/y][[\![t]\!]/x] = [\![Q]\!][[\![t]\!]/x][[\![t]\!]x_1...x_n/y] = [\![Q]\!][[\![t]\!]x_1...x_n/y]$.

In this setting, the judgement typing q_k can be written $[\![\Gamma]\!] \vdash q_k : \Pi h(x, k) : (\Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![Q]\!][[\![t]\!]x_1...x_n/y]).P'$. By induction hypothesis on the second premise of the original derivation, there exists a proof q' such that $[\![\Gamma]\!] \vdash q' : \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![Q]\!][[\![t]\!]x_1...x_n/y]$. Hence, applying the App rule, and using the fact that no proof variable appears free in a λ-HOL expression by Proposition 8.3.3, we conclude that $[\![\Gamma]\!] \vdash q_{k+1} : P'$ is derivable in λ-HOL with $q_{k+1} = q_k q'$.

This concludes the proof of the fact that for all k, there exists a proof q_k such that $[\![\Gamma]\!] \vdash q_k : (\Pi(\textsc{Axioms}(x : A)_{\geq k}).[\![P]\!])[[\![t]\!]/x]$ is derivable in λ-HOL. Taking $k = \textsc{Size}(A) + 1$ and using Proposition 8.5.1, we conclude that the derivation $[\![\Gamma]\!] \vdash q_k : [\![P[t/x]]\!]$ is derivable in λ-HOL.

– $\dfrac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}$ App

This case is the most complex one in this proof. By the renaming theorem, we suppose without loss of generality $x \notin DV(\Gamma)$. We consider some index i such that $\textsc{Occurrence}(B[u/x], i)$ has the form $\lambda x_1 : B_1 ... \lambda x_n : B_n.\{y : B' \mid P\}$. By Lemma 8.5.1, $\textsc{Occurrence}(B, i)$ has this form as well. We define $\textsc{Occurrence}(B, i) = \lambda x_1 : B_1 ... \lambda x_n : B_n.\{y : B' \mid P\}$, choosing the variables $x_1, ..., x_n, y$ pairwise distinct, different from x, and outside $DV(\Gamma)$. By the free variable theorem, these variables are also outside $FV(u)$. Hence, applying Lemma 8.5.1 again, $\textsc{Occurrence}(B[u/x], i) = \lambda x_1 : B_1[u/x] ... \lambda x_n : B_n[u/x].\{y : B'[u/x] \mid P[u/x]\}$. In this setting, we find the expected proof in the following way.

By definition, $\textsc{Occurrence}(\Pi x : A.B, i + 1) = \lambda x : A.\textsc{Occurrence}(B, i)$. By induction hypothesis, there exists a proof p such that $[\![\Gamma]\!] \vdash p : \Pi[\![x : A, x_1 : B_1, ..., x_n : B_n]\!].[\![P]\!][[\![t]\!]xx_1...x_n/y]$ is derivable in λ-HOL. We define $Q = \Pi[\![x_1 : B_1, ..., x_n : B_n]\!].[\![P]\!][[\![t]\!]xx_1...x_n/y]$: in this setting, the previous judgement can be written $[\![\Gamma]\!] \vdash p : \Pi x : [A].\Pi(\textsc{Axioms}(x : A)).Q$. By Proposition 8.4.4, $[\![\Gamma]\!] \vdash [\![u]\!] : [A]$ is derivable in λ-HOL. Hence, $[\![\Gamma]\!] \vdash p[\![u]\!] : (\Pi(\textsc{Axioms}(x : A)).Q)[[\![u]\!]/x]$ is derivable in λ-HOL.

Starting from this result, we prove by induction that for all k, there exists a

proof p_k such that $[\![\Gamma]\!] \vdash p_k : (\Pi(\textsc{Axioms}(x : A)_{\geq k}).Q)[[\![u]\!]/x]$ is derivable in λ-HOL. The case $k = 0$ corresponds to the previous derivation with $p_0 = p[\![u]\!]$.

Supposing that $[\![\Gamma]\!] \vdash p_k : (\Pi(\textsc{Axioms}(x : A)_{\geq k}).Q)[[\![u]\!]/x]$ is derivable in λ-HOL for some k, we find a proof p_{k+1} in the following way. We consider I the set of all indexes $j \in \{0, ..., \textsc{Size}(A)\}$ such that $\textsc{Occurrence}(A, j)$ outputs some term of the form $\lambda x'_1 : A'_1 ... \lambda x'_{n'} : A'_{n'}.\{y' : A' \mid P'\}$. If $k \notin I$, the expected result is straightforward, taking $p_{k+1} = p_k$.

If $k \in I$, $\textsc{Occurrence}(A, k)$ outputs some term of the form $\lambda x'_1 : A'_1 ... \lambda x'_{n'} : A'_{n'}.\{y' : A' \mid P'\}$. We choose $x'_1, ..., x'_{n'}, y'$ pairwise distinct, different from x, and outside $DV(\Gamma)$. In the following, we use the notation $Q' = (\Pi(\textsc{Axioms}(x : A)_{\geq k+1}).Q)[[\![u]\!]/x]$. The proof variable $h(x, k)$ is distinct from the expression variable x, and cannot occur free in $[\![u]\!]$ by Proposition 8.3.3. Hence, the expression $(\Pi(\textsc{Axioms}(x : A)_{\geq k}).Q)[[\![u]\!]/x]$ corresponds to the following one: $\Pi h(x, k) : (\Pi[\![x'_1 : A'_1, ..., x'_{n'} : A'_{n'}]\!].[\![P']\!][xx'_1 ... x'_{n'}/y'])[[\![u]\!]/x].Q'$.

As the variables $x'_1, ..., x'_{n'}, y'$ are pairwise distinct and outside $DV(\Gamma)$, we can use successively Theorem 5.2.3, Proposition 8.4.3, the subderivations theorem, and the renaming theorem to conclude that $\Gamma, x'_1 : A'_1, ..., x'_{n'} : A'_{n'}, y' : A' \vdash P' : Prop$ is derivable in PVS-Cert. Hence, by Proposition 8.4.4, $[\![\Gamma, x'_1 : A'_1, ..., x'_{n'} : A'_{n'}, y' : A']\!] \vdash [\![P']\!] : Prop$ is derivable in λ-HOL. As x is an expression variable outside $DV(\Gamma, x'_1 : A'_1, ..., x'_{n'} : A'_{n'}, y' : A')$, it is outside $DV([\![\Gamma, x'_1 : A'_1, ..., x'_{n'} : A'_{n'}, y' : A']\!])$. Hence, by the free variable theorem, $x \notin FV([\![\Gamma, x'_1 : A'_1, ..., x'_{n'} : A'_{n'}, y' : A']\!])$ and $x \notin FV([\![P']\!])$.

As a consequence of the first of these two statements, the term $(\Pi[\![x'_1 : A'_1, ..., x'_{n'} : A'_{n'}]\!].[\![P']\!][xx'_1 ... x'_{n'}/y'])[[\![u]\!]/x]$ is equal to $\Pi[\![x'_1 : A'_1, ..., x'_{n'} : A'_{n'}]\!].[\![P']\!][xx'_1 ... x'_{n'}/y'][[\![u]\!]/x]$. By Proposition 8.4.4, $[\![\Gamma]\!] \vdash [\![u]\!] : [A]$ is derivable in λ-HOL. As y' is an expression variable outside $DV(\Gamma)$, it is outside $DV([\![\Gamma]\!])$. Hence, by the free variable theorem, $y' \notin FV([\![u]\!])$. Moreover, the variables $x'_1, ..., x'_{n'}, y'$ are different from x, and we proved $x \notin FV([\![P']\!])$, hence we conclude using Lemma 5.1.1 $[\![P']\!][xx'_1 ... x'_{n'}/y'][[\![u]\!]/x] = [\![P']\!][[\![u]\!]/x][[\![u]\!]x'_1 ... x'_{n'}/y'] = [\![P']\!][[\![u]\!]x'_1 ... x'_{n'}/y']$.

In this setting, the judgement typing p_k can be written $[\![\Gamma]\!] \vdash p_k : \Pi h(x, k) : (\Pi[\![x'_1 : A'_1, ..., x'_{n'} : A'_{n'}]\!].[\![P']\!][[[\![u]\!]x'_1 ... x'_{n'}/y']).Q'$. By induction hypothesis on the second premise of the original derivation, there exists a proof q such that $[\![\Gamma]\!] \vdash q : \Pi[\![x'_1 : A'_1, ..., x'_{n'} : A'_{n'}]\!].[\![P']\!][[[\![u]\!]x'_1 ... x'_n/y']$. Hence, applying the App rule, and using the fact that no proof variable appears free in a λ-HOL expression by Proposition 8.3.3, we conclude that $[\![\Gamma]\!] \vdash p_{k+1} : (\Pi(\textsc{Axioms}(x : A)_{\geq k+1}).Q)[[\![u]\!]/x]$ is derivable in λ-HOL with $p_{k+1} = p_k q$.

This concludes the proof of the fact that for all k, there exists a proof p_k such that $[\![\Gamma]\!] \vdash p_k : (\Pi(\textsc{Axioms}(x : A)_{\geq k+1}).Q)[[\![u]\!]/x]$ is derivable in λ-HOL. Taking $k = \textsc{Size}(A) + 1$, we conclude that there exists a proof q such that $[\![\Gamma]\!] \vdash q : (\Pi[\![x_1 : B_1, ..., x_n : B_n]\!].[\![P]\!][[\![t]\!]xx_1...x_n/y])[[\![u]\!]/x]$ is derivable in λ-HOL.

As $x_1, ..., x_n, y$ are expression variables outside $DV(\Gamma)$, they are outside $DV([\![\Gamma]\!])$. Hence, by the free variable theorem, they are outside $FV([\![u]\!])$. They are also distinct from x. On the other hand, any proof variable is distinct from x and cannot occur free in $[\![u]\!]$ by Proposition 8.3.3. In this setting, expanding $[\![x_1 : B_1, ..., x_n : B_n]\!]$ to $x_1 : [B_1], \textsc{Axioms}(x_1 : B_1), ..., x_n : [B_n], \textsc{Axioms}(x_n : B_n)$, applying Lemma 8.5.1 and Proposition 8.4.1, the term $(\Pi[\![x_1 : B_1, ..., x_n : B_n]\!].[\![P]\!][[\![t]\!]xx_1...x_n/y])[[\![u]\!]/x]$ can be written as $\Pi[\![x_1 : B_1[u/x], ..., x_n : B_n[u/x]]\!].[\![P]\!][[\![t]\!]xx_1...x_n/y][[\![u]\!]/x]$.

As we proved $y \notin FV([\![u]\!])$ in the previous paragraph and as the variables $x_1, ..., x_n, y$ are different from x, by Lemma 5.1.1 and Lemma 8.5.1, the two expressions $[\![P]\!][[\![t]\!]xx_1...x_n/y][[\![u]\!]/x]$ and $[\![P[u/x]]\!][[\![t[u/x]]\!][\![u]\!]x_1...x_n/y]$ are equal. By the free variable theorem, $x \notin FV(t)$, hence this term is also equal to $[\![P[u/x]]\!][[\![t]\!][\![u]\!]x_1...x_n/y]$. Hence the judgement typing q can be written $[\![\Gamma]\!] \vdash q : \Pi[\![x_1 : B_1[u/x], ..., x_n : B_n[u/x]]\!].[\![P[u/x]]\!][[\![tu]\!]x_1...x_n/y]$, which is the expected result as $\textsc{Occurrence}(B[u/x], i) = \lambda x_1 : B_1[u/x]...\lambda x_n : B_n[u/x].\{y : B'[u/x] \mid P[u/x]\}$.

- The case Subtype cannot occur by the stratification theorem
- The case Pair is proved in the following way. Using the stratification theorem, we can suppose that the last inference step matches some instance of the following form.

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash p : P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : Type}{\Gamma \vdash \langle t, p \rangle_{\{x:A|P\}} : \{x : A \mid P\}}\ \textsc{Pair}$$

We consider some index i such that $\textsc{Occurrence}(\{x : A \mid P\}, i)$ has the form $\lambda x_1 : A_1...\lambda x_n : A_n.\{y : B \mid P'\}$. We split the two cases $i = 0$ and $i > 0$.

For $i = 0$, $\textsc{Occurrence}(\{x : A \mid P\}, i) = \{x : A \mid P\}$. By induction hypothesis applied on the second premise, there exists a proof q such that $[\![\Gamma]\!] \vdash q : [\![P[t/x]]\!]$ is derivable in λ-HOL, which is the expected result as, by Proposition 8.5.1, $[\![P[t/x]]\!] = [\![P]\!][[\![t]\!]/x]$.

For $i > 0$, by definition, $\textsc{Occurrence}(\{x : A \mid P\}, i) = \textsc{Occurrence}(A, i - 1)$.

As, moreover, $[\![t]\!] = [\![\langle t, p\rangle_{\{x:A|P\}}]\!]$, we can conclude the expected result directly by induction hypothesis on the first premise.

- The case PROJ1 is proved in the following way. Using the stratification theorem, we can suppose that the last inference step matches some instance of the following form.

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \pi_1(t) : A}\ \textsc{Proj1}$$

We consider some index i such that $\textsc{Occurrence}(\{x : A \mid P\}, i)$ has the form $\lambda x_1 : A_1 ... \lambda x_n : A_n.\{y : B \mid P'\}$. By definition, $\textsc{Occurrence}(A, i) = \textsc{Occurrence}(\{x : A \mid P\}, i + 1)$. As, moreover, $[\![t]\!] = [\![\pi_1(t)]\!]$, we can conclude the expected result directly by induction hypothesis.

- The case PROJ2 is proved in the following way. Using the stratification theorem, the last inference step matches some rule instance of the following form.

$$\frac{\Gamma \vdash t : \{y : A \mid P\}}{\Gamma \vdash \pi_2(t) : P[\pi_1(t)/y]}\ \textsc{Proj2}$$

By induction hypothesis, there exists a proof q such that $[\![\Gamma]\!] \vdash q : [\![P]\!][[\![t]\!]/y]$ is derivable in λ-HOL. This corresponds to the expected result as, by Proposition 8.5.1, $[\![P]\!][[\![t]\!]/y] = [\![P]\!][[\![\pi_1(t)]\!]/y] = [\![P[\pi_1(t)/y]]\!]$.

- The case CONVERSION is proved in the following way. Using the stratification theorem, the last inference step matches one of the following instances

  – $$\frac{\Gamma \vdash p : P \qquad \Gamma \vdash Q : Prop}{\Gamma \vdash p : Q}\ \textsc{Conversion}\ P \equiv_{\beta *} Q$$

  By induction hypothesis, there exists some proof q such that $[\![\Gamma]\!] \vdash q : [\![P]\!]$ is derivable in λ-HOL. Moreover, by Proposition 8.4.4, $[\![\Gamma]\!] \vdash [\![Q]\!] : Prop$ is derivable in λ-HOL. By Proposition 8.5.2, $[\![P]\!] \equiv_\beta [\![Q]\!]$. Hence, applying the λ-HOL conversion rule, $[\![\Gamma]\!] \vdash q : [\![Q]\!]$ is derivable in λ-HOL.

  – $$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : Type}{\Gamma \vdash t : B}\ \textsc{Conversion}\ A \equiv_{\beta *} B$$

  We consider some index i such that $\textsc{Occurrence}(B, i)$ has the form $\lambda x_1 : B_1 ... \lambda x_n : B_n.\{y : B' \mid Q\}$. By Lemma 8.5.4, $\textsc{Occurrence}(A, i)$ has the form $\lambda x_1 : A_1 ... \lambda x_n : A_n.\{y : A' \mid P\}$ with $P \equiv_{\beta *} Q$ and, for all $i \in \{1, ..., n\}$, $A_i \equiv_{\beta *} B_i$. By induction hypothesis, there exists a proof p such that $[\![\Gamma]\!] \vdash p : \Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][[\![t]\!]x_1 ... x_n/y]$ is derivable

in λ-HOL. By Proposition 8.4.1 and Proposition 8.5.2, $\Pi[\![x_1 : A_1, ..., x_n : A_n]\!].[\![P]\!][[\![t]\!]x_1...x_n/y] \equiv_\beta \Pi[\![x_1 : B_1, ..., x_n : B_n]\!].[\![Q]\!][[\![t]\!]x_1...x_n/y]$. Hence, we can conclude the expected result by conversion as long as we prove that $[\![\Gamma]\!] \vdash \Pi[\![x_1 : B_1, ..., x_n : B_n]\!].[\![Q]\!][[\![t]\!]x_1...x_n/y] : \mathit{Prop}$ is derivable in λ-HOL, which is the case by Proposition 8.5.3.

□

We conclude the conservativity of PVS-Cert over λ-HOL from the conservativity of the translation $[\![\cdot]\!]$ and Proposition 8.2.3 in the following way.

Theorem 8.5.2. *PVS-Cert is a conservative extension of λ-HOL: for every judgement of the form $\Gamma \vdash P : \mathit{Prop}$ derivable in λ-HOL, P is inhabited in Γ in PVS-Cert if and only if it is inhabited in Γ in λ-HOL.*

Proof. As λ-HOL is a subsystem of PVS-Cert, the existence of a λ-HOL term p such that $\Gamma \vdash p : P$ is derivable in λ-HOL implies that $\Gamma \vdash p : P$ is derivable in PVS-Cert.

Conversely, we suppose that there exists a PVS-Cert term p such that $\Gamma \vdash p : P$ is derivable in PVS-Cert. We consider an injective indexing $h(\cdot, \cdot)$ such that $DV(\Gamma)$ and the image of $h(\cdot, \cdot)$ are disjoint. By Theorem 8.5.1, there exists a proof q such that $[\![\Gamma]\!] \vdash q : [\![P]\!]$ is derivable in λ-HOL. Hence, by Proposition 8.2.3, $\Gamma \vdash q : P$ is derivable in λ-HOL. □

Chapter 9

Expressing PVS-Core in PVS-Cert

The final purpose of PVS-Cert is to encode PVS-Core derivations as proof terms, and to use the type-checking algorithm presented in Chapter 7 to use these proof terms as verifiable certificates. In this perspective, we define a correspondence between PVS-Core and PVS-Cert. This correspondence reflects the fact that, even though these two systems are very different at the level of terms and judgements, they are almost identical at the level of derivations. We begin the description of this correspondence with a translation from PVS-Cert to PVS-Core, which will be referred to as *erasing* in the following of this chapter.

9.1 From PVS-Cert to PVS-Core

We define the following translation from the stratified part of the PVS-Cert syntax to PVS-Core. This translation is referred to as erasing as it mainly consists in the deletion of PVS-Cert explicit coercions $\langle \cdot, M \rangle_A$ and $\pi_1(\cdot)$.

Definition 9.1.1. *We define an erasure function $[\![\cdot]\!]$ from PVS-Cert expressions, types, and Type to PVS-Core terms recursively as follows. In the case of variables, this definition relies on the fact that the sets of expression variables and type variables $\mathcal{V}_{expressions}$ and $\mathcal{V}_{types}$ are shared in PVS-Cert and PVS-Core, as mentioned in the definition of PVS-Cert.*

- $[\![Type]\!] = Type$
- $[\![X]\!] = X$
- $[\![Prop]\!] = Prop$
- $[\![\Pi x : A.B]\!] = \Pi x : [\![A]\!].[\![B]\!]$
- $[\![\{x : A \mid P\}]\!] = \{x : [\![A]\!] \mid [\![P]\!]\}$

- $[\![x]\!] = x$
- $[\![\Pi h : P.Q]\!] = [\![P]\!] \Rightarrow [\![Q]\!]$
- $[\![\Pi x : A.P]\!] = \forall x : [\![A]\!].[\![P]\!]$
- $[\![\lambda x : A.t]\!] = \lambda x : [\![A]\!].[\![t]\!]$
- $[\![t\ u]\!] = [\![t]\!][\![u]\!]$
- $[\![\langle t, M\rangle_A]\!] = [\![t]\!]$
- $[\![\pi_1(t)]\!] = [\![t]\!]$

Then, we extend this $[\![\cdot]\!]$ from PVS-Cert stratified contexts to PVS-Core contexts. Again, this definition relies on the fact that the sets of expression variables and type variables $\mathcal{V}_{expressions}$ and $\mathcal{V}_{types}$ are shared in PVS-Cert and PVS-Core.

- $[\![\varnothing]\!] = \varnothing$
- $[\![\Gamma, X : Type]\!] = [\![\Gamma]\!], X : Type$
- $[\![\Gamma, x : A]\!] = [\![\Gamma]\!], x : [\![A]\!]$
- $[\![\Gamma, h : P]\!] = [\![\Gamma]\!], [\![P]\!]$

Last, we extend $[\![\cdot]\!]$ from all PVS-Cert stratified judgements except those of the form $\Gamma \vdash Type : Kind$ to PVS-Core judgements as follows.

- $[\![\Gamma \vdash WF]\!] = [\![\Gamma]\!] \vdash WF$
- $[\![\Gamma \vdash A : Type]\!] = [\![\Gamma]\!] \vdash [\![A]\!] : Type$
- $[\![\Gamma \vdash t : A]\!] = [\![\Gamma]\!] \vdash [\![t]\!] : [\![A]\!]$
- $[\![\Gamma \vdash p : P]\!] = [\![\Gamma]\!] \vdash [\![P]\!]$

By the stratification theorem in PVS-Cert, all PVS-Cert derivable judgements are stratified judgements. Hence, unless they have the form $\Gamma \vdash Type : Kind$, their erasure in PVS-Core is well-defined. We will prove in Theorem 9.1.1 that all PVS-Core judgements obtained by erasing derivable PVS-Cert judgements are themselves derivable. This theorem relies in particular on the fact that conversion in PVS-Cert and PVS-Core are related through the erasure function $[\![\cdot]\!]$, which will be established in Proposition 9.1.2. In this purpose, we first prove the following property.

Proposition 9.1.1. *For any PVS-Cert term M which is either an expression, a type, or Type, the following statements hold.*

- *For any PVS-Cert term N and any PVS-Cert proof variable h, $[\![M[N/h]]\!] = [\![M]\!]$.*

- *For any PVS-Cert term N which is either an expression, a type, or $Type$, and any PVS-Cert expression variable x, $[\![M[N/x]]\!] = [\![M]\!][[\![N]\!]/x]$.*

Proof. The two statements are proved by induction on M. As the erasure function and substitution are stable by α-conversion, we can suppose without loss of generality that no bound variable in M is free in N nor equal to x. Using this hypothesis, all cases are straightforward. □

We conclude the following proposition relating conversion in PVS-Cert and PVS-Core.

Proposition 9.1.2. *For all terms M and N which are either expressions, types, or $Type$, whenever $M \equiv_{\beta *} N$, then $[\![M]\!] \equiv_{\beta} [\![N]\!]$.*

Proof. We first prove that for any term M which is either an expression or a type, whenever there exists a term N such that $M \to_{\beta *} N$, then N is either an expression or a type, and $[\![M]\!] \equiv_{\beta} [\![N]\!]$. The fact that N is either an expression or a type follows directly from Proposition 5.4.2. The fact that $[\![M]\!] \equiv_{\beta} [\![N]\!]$ is proved by induction on M. All cases except the case of an application are straightforward by induction hypothesis. If M is an application, we split the subcases $M \not\triangleright_{\beta *} N$ and $M \triangleright_{\beta *} N$. The first subcase is straightforward by induction hypothesis. In the second case, M has the form $(\lambda x : A.t)u$ and N has the form $t[u/x]$, hence we can conclude using Proposition 9.1.1.

In a second step, we prove that for any term M which is either an expression, a type, or $Type$, whenever there exists a term N such that $M \twoheadrightarrow_{\beta *} N$, then $[\![M]\!] \equiv_{\beta} [\![N]\!]$. This is done as follows. The case $M = Type$ is straightforward as, in this case, $M = N = Type$. The cases where M is either an expression or a type are proved together by induction on the length of the reduction, using the first result.

Last, we conclude the expected result as follows. We consider two terms M and N which are either expressions, types, or $Type$, and such that $M \equiv_{\beta *} N$. By the Church-Rosser theorem, there exists a term M' such that $M \twoheadrightarrow_{\beta *} M'$ and $N \twoheadrightarrow_{\beta *} M'$. Hence, using the previous result, $[\![M]\!] \equiv_{\beta} [\![M']\!] \equiv_{\beta} [\![N]\!]$. □

Using Proposition 9.1.2 and the stratification theorem in PVS-Cert, we conclude the following theorem, which allows to map PVS-Cert derivations to PVS-Core derivations, which will be referred to as their erasures.

Theorem 9.1.1. *Every derivable PVS-Cert judgement either has the form $\Gamma \vdash Type : Kind$ or admits an image through $[\![\cdot]\!]$. In the latter case, this image is derivable in PVS-Core.*

Proof. The first part of the proof is a direct consequence of the stratification theorem. The second part is done by strong induction on the height of PVS-Cert derivations. Most cases are straightforward. The most complex ones correspond to the PVS-Cert rules App and Conversion, and are proved using Proposition 9.1.1 and Proposition

9.1.2 respectively. However, the proof is given in details in order to show explicitly how PVS-Cert rules and PVS-Core rules are related. The possible cases are the following.

- The case EMPTY is straightforward using the PVS-Core rule EMPTY.
- In the case DECL, by the stratification theorem, the last inference step matches one of the following instances:
  - $\dfrac{\Gamma \vdash Type : Kind}{\Gamma, X : Type \vdash WF}$ DECL $X \in \mathcal{V}_{types} \backslash DV(\Gamma)$

    By the subderivations theorem, $\Gamma \vdash WF$ is a strict subderivation of the original one. Hence, by induction hypothesis, $[\![\Gamma]\!] \vdash WF$ is derivable in PVS-Core. As $X \notin \Gamma$, $X \notin [\![\Gamma]\!]$, hence we can conclude applying the PVS-Core rule TYPEDECL that $[\![\Gamma]\!], X : Type \vdash WF$ is derivable.

  - $\dfrac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash WF}$ DECL $x \in \mathcal{V}_{expressions} \backslash DV(\Gamma)$

    By induction hypothesis, $[\![\Gamma]\!] \vdash [\![A]\!] : Type$ is derivable in PVS-Core. As $x \notin \Gamma$, $x \notin [\![\Gamma]\!]$, hence we can conclude applying the PVS-Core rule ELTDECL that $[\![\Gamma]\!], x : [\![A]\!] \vdash WF$ is derivable.

  - $\dfrac{\Gamma \vdash P : Prop}{\Gamma, h : P \vdash WF}$ DECL $h \in \mathcal{V}_{proofs} \backslash DV(\Gamma)$

    By induction hypothesis, $[\![\Gamma]\!] \vdash [\![P]\!] : Prop$ is derivable in PVS-Core. Hence, we can conclude applying the PVS-Core rule ASSUMPTION that $[\![\Gamma]\!], [\![P]\!] \vdash WF$ is derivable.

- In the case SORT, the last inference step matches one of the following PVS-Cert instances.
  - $\dfrac{\Gamma \vdash WF}{\Gamma \vdash Type : Kind}$ SORT

    No proof is expected in this case.

  - $\dfrac{\Gamma \vdash WF}{\Gamma \vdash Prop : Type}$ SORT

    The expected result follows by induction hypothesis, applying the PVS-Core rule PROP.

- In the case VAR, by the stratification theorem, the last inference step matches one of the following instances:

– $\dfrac{\Gamma \vdash WF}{\Gamma \vdash h : P}$ VAR $(h : P) \in \Gamma$

By induction hypothesis, $[\![\Gamma]\!] \vdash WF$ is derivable in PVS-Core. As $(h : P) \in \Gamma$, $[\![P]\!] \in [\![\Gamma]\!]$, hence we can conclude applying the PVS-Core rule AXIOM that $[\![\Gamma]\!] \vdash [\![P]\!]$ is derivable.

– $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ VAR $(x : A) \in \Gamma$

By induction hypothesis, $[\![\Gamma]\!] \vdash WF$ is derivable in PVS-Core. As $(x : A) \in \Gamma$, $(x : [\![A]\!]) \in [\![\Gamma]\!]$, hence we can conclude applying the PVS-Core rule ELTVAR that $[\![\Gamma]\!] \vdash x : [\![A]\!]$ is derivable.

– $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ VAR $(X : Type) \in \Gamma$

By induction hypothesis, $[\![\Gamma]\!] \vdash WF$ is derivable in PVS-Core. As $(X : Type) \in \Gamma$, $(X : Type) \in [\![\Gamma]\!]$, hence we can conclude applying the PVS-Core rule TYPEVAR that $[\![\Gamma]\!] \vdash X : Type$ is derivable.

- In the case PROD, by the stratification theorem, the last inference step matches one of the following instances:

  – $\dfrac{\Gamma \vdash P : Prop \qquad \Gamma, h : P \vdash Q : Prop}{\Gamma \vdash \Pi h : P.Q : Prop}$ PROD

  The expected result follows by induction hypothesis on the second premise, applying the PVS-Core rule IMPLY.

  – $\dfrac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \Pi x : A.P : Prop}$ PROD

  The expected result follows by induction hypothesis on the second premise, applying the PVS-Core rule FORALL.

  – $\dfrac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}$ PROD

  The expected result follows by induction hypothesis on the second premise, applying the PVS-Core rule PI.

- In the case SUBTYPE, by the stratification theorem, the last inference step matches some instance of the form

$$\frac{\Gamma \vdash A : Type \qquad \Gamma, x : A \vdash P : Prop}{\Gamma \vdash \{x : A \mid P\} : Type}\ \textsc{Subtype}$$

In this setting, the expected result follows by induction hypothesis on the second premise, applying the PVS-Core rule Subtype.

- In the case Lam, by the stratification theorem, the last inference step matches one of the following instances:

  – $$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x : A.B : Type}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}\ \textsc{Lam}$$

  The expected result follows by induction hypothesis on the first premise, applying the PVS-Core rule Lam.

  – $$\frac{\Gamma, x : A \vdash M : P \qquad \Gamma \vdash \Pi x : A.P : Prop}{\Gamma \vdash \lambda x : A.M : \Pi x : A.P}\ \textsc{Lam}$$

  The expected result follows by induction hypothesis on the first premise, applying the PVS-Core rule ForallIntro.

  – $$\frac{\Gamma, h : P \vdash M : Q \qquad \Gamma \vdash \Pi h : P.Q : Prop}{\Gamma \vdash \lambda h : P.M : \Pi h : P.Q}\ \textsc{Lam}$$

  The expected result follows by induction hypothesis on the first premise, applying the PVS-Core rule ImplyIntro.

- In the case App, by the stratification theorem, the last inference step matches one of the following instances:

  – $$\frac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}\ \textsc{App}$$

  By induction hypothesis, applying the PVS-Core rule App, $[\![\Gamma]\!] \vdash [\![t]\!][\![u]\!] : [\![B]\!][[\![u]\!]/x]$ is derivable in PVS-Core. By Proposition 9.1.1, this corresponds the expected result as $[\![B]\!][[\![u]\!]/x] = [\![B[u/x]]\!]$.

  – $$\frac{\Gamma \vdash M : \Pi x : A.P \qquad \Gamma \vdash t : A}{\Gamma \vdash Mt : P[t/x]}\ \textsc{App}$$

  By induction hypothesis, applying the PVS-Core rule ForallElim, $[\![\Gamma]\!] \vdash [\![P]\!][[\![t]\!]/x]$ is derivable in PVS-Core. By Proposition 9.1.1, this corresponds the expected result as $[\![P]\!][[\![t]\!]/x] = [\![P[t/x]]\!]$.

  – $$\frac{\Gamma \vdash p : \Pi h : P.Q \qquad \Gamma \vdash q : P}{\Gamma \vdash pq : Q[q/h]}\ \textsc{App}$$

By induction hypothesis, applying the PVS-Core rule ImplyElim, $[\![\Gamma]\!] \vdash [\![Q]\!]$ is derivable in PVS-Core. By Proposition 9.1.1, this corresponds the expected result as $[\![Q]\!] = [\![Q[q/h]]\!]$.

- In the case Pair, by the stratification theorem, the last inference step matches some instance of the form

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash M : P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : Type}{\Gamma \vdash \langle t, M \rangle_{\{x:A|P\}} : \{x : A \mid P\}}\ \textsc{Pair}$$

By induction hypothesis, the judgement $[\![\Gamma]\!] \vdash [\![t]\!] : [\![A]\!]$, the judgement $[\![\Gamma]\!] \vdash [\![P[t/x]]\!]$, and the judgement $[\![\Gamma]\!] \vdash \{x : [\![A]\!] \mid [\![P]\!]\} : Type$ are derivable in PVS-Core. By Proposition 9.1.1, the second one corresponds to $[\![\Gamma]\!] \vdash [\![P]\!][[\![t]\!]/x]$ as $[\![P[t/x]]\!] = [\![P]\!][[\![t]\!]/x]$. Hence, we can conclude the expected result applying the PVS-Core rule SubtypeIntro.

- In the case Proj1, by the stratification theorem, the last inference step matches some instance of the form

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \pi_1(t) : A}\ \textsc{Proj1}$$

The expected result follows by induction hypothesis, applying the PVS-Core rule SubtypeElim1.

- In the case Proj2, by the stratification theorem, the last inference step matches some instance of the form

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \pi_2(t) : P[\pi_1(t)/x]}\ \textsc{Proj2}$$

By induction hypothesis, applying the PVS-Core rule SubtypeElim2, $[\![\Gamma]\!] \vdash [\![P]\!][[\![t]\!]/x]$ is derivable in PVS-Core. By Proposition 9.1.1, this corresponds the expected result as $[\![P]\!][[\![t]\!]/x] = [\![P]\!][[\![\pi_1(t)]\!]/x] = [\![P[\pi_1(t)/x]]\!]$.

- In the case Conversion, by the stratification theorem, the last inference step matches one of the following instances:
  - $\dfrac{\Gamma \vdash A : Type \qquad \Gamma \vdash Type : Kind}{\Gamma \vdash A : Type}\ \textsc{Conversion}\ Type \equiv_{\beta *} Type$

The expected result is straightforward by induction hypothesis on the first premise.

– $\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash B : Type}{\Gamma \vdash t : B}$ Conversion $A \equiv_{\beta *} B$

By Proposition 9.1.2, $[\![A]\!] \equiv_\beta [\![B]\!]$. In this setting, the expected result follows by induction hypothesis, applying the PVS-Core rule TypeConversion.

– $\dfrac{\Gamma \vdash p : P \qquad \Gamma \vdash Q : Prop}{\Gamma \vdash p : Q}$ Conversion $P \equiv_{\beta *} Q$

By Proposition 9.1.2, $[\![P]\!] \equiv_\beta [\![Q]\!]$. In this setting, the expected result follows by induction hypothesis, applying the PVS-Core rule PropConversion.

□

9.2 Expressing PVS-Core derivations as PVS-Cert judgement

Theorem 9.1.1 shows that a PVS-Cert derivable judgement can testify to the PVS-Core derivability of another judgement, its erasure. In this section, we show conversely that, given any PVS-Core derivation, we can build such a PVS-Cert judgement. For this purpose, we first present an algorithm Certificate, which translates a PVS-Core derivation into a PVS-Cert judgement. In a second step, we will prove that such PVS-Cert judgements are always derivable in PVS-Cert.

The following definition will be useful to define the algorithm Certificate.

Definition 9.2.1. *We define some injective function $h(\cdot)$ mapping natural numbers to PVS-Cert proof variables, and referred to as an injective indexing (as in Definition 8.2.3).*

The definition of the algorithm Certificate also relies on some properties of the erasure function $[\![\cdot]\!]$. More precisely, we will need to infer, in some specific cases, the form of PVS-Cert pre-images through $[\![\cdot]\!]$ from the form of their PVS-Core images. We split these properties in two propositions. The first, Proposition 9.2.1, contains straightforward properties, and will be used implicitly in the following. The second, Proposition 9.2.2, contains more complex properties, and will be mentioned explicitly in the following.

Proposition 9.2.1. *The erasure function $[\![\cdot]\!]$ respects the following specifications.*

- *Whenever a PVS-Cert term M is either an expression, a type, or Type, it is an expression (resp. a type, Type) if and only if $[\![M]\!]$ is an expression (resp. a type, Type).*

- *The erasure of a PVS-Cert stratified context* Γ *is a context of same length. Moreover, for every index* n*, the* n*-th declaration of* Γ *has the form* $(h : P)$ *(resp.* $(x : A)$*,* $(X : Type)$*) if and only if the* n*-th declaration of* $[\![\Gamma]\!]$ *has the form* P' *(resp.* $(x : A')$*,* $(X : Type)$*).*

- *For every PVS-Cert stratified judgement which doesn't have the form* $\Gamma \vdash Type : Kind$*, this judgement has the form* $\Gamma \vdash WF$ *(resp.* $\Gamma \vdash p : P$*,* $\Gamma \vdash t : A$*,* $\Gamma \vdash A : Type$*) if and only if its erasure has the form* $\Gamma \vdash WF$ *(resp.* $\Gamma \vdash p : P$*,* $\Gamma \vdash t : A$*,* $\Gamma \vdash A : Type$*).*

Proof. The proof is straightforward by case analysis. □

More complex properties of the translation $[\![\cdot]\!]$ are presented in the following proposition.

Proposition 9.2.2. *For any PVS-Cert type* A*, the following statements hold.*

- *Whenever* $[\![A]\!] = Prop$*,* $A = Prop$
- *Whenever* $[\![A]\!]$ *has the form* $\Pi x : B_1.B_2$*,* A *has the form* $\Pi x : B'_1.B'_2$
- *Whenever* $[\![A]\!]$ *has the form* $\{x : B \mid P\}$*,* A *has the form* $\{x : B' \mid P'\}$

For any PVS-Cert expression t*, the following statements hold.*

- *Whenever* $[\![t]\!]$ *has the form* $P \Rightarrow Q$*,* t *admits a normal form with respect to* $\triangleright_*$*, which has the form* $\Pi h : P'.Q'$
- *Whenever* $[\![t]\!]$ *has the form* $\forall x : A.P$*,* t *admits a normal form with respect to* $\triangleright_*$*, which has the form* $\Pi x : A'.P'$

Proof. The statements dealing with types are straightforward by case analysis on the possible forms of a type. The two statements dealing with expressions are proved independently by straightforward induction on expressions. □

The expression of PVS-Core derivations as PVS-Cert judgements is defined as follows.

Definition 9.2.2. *For any PVS-Core derivation* D*, we define recursively the PVS-Cert stratified judgement* Certificate(D) *such that* $[\![\textsc{Certificate}(D)]\!]$ *corresponds to the conclusion of* D*. In the following, we provide a proof of the fact that* $[\![\textsc{Certificate}(D)]\!]$ *corresponds to the conclusion of* D *only when it is not straightforward by induction hypothesis, i.e. in the cases corresponding to the rules* App*,* ImplyElim*, and* ForallElim*. The possible cases are the following.*

- $\dfrac{}{\varnothing \vdash WF}$ Empty

  We define Certificate$(D) = \varnothing \vdash WF$*.*

- $\dfrac{\Gamma \vdash WF}{\Gamma, X : Type \vdash WF}$ TypeDecl $X \in \mathcal{V}_{types} \backslash DV(\Gamma)$

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash WF$. We define* Certificate$(D) = \Gamma_1, X : Type \vdash WF$.

- $\dfrac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash WF}$ EltDecl $x \in \mathcal{V}_{expressions} \backslash DV(\Gamma)$

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash A_1 : Type$. We define* Certificate$(D) = \Gamma_1, x : A_1 \vdash WF$.

- $\dfrac{\Gamma \vdash P : Prop}{\Gamma, P \vdash WF}$ Assumption

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash P_1 : Prop$ by Proposition 9.2.2. We consider n the number of declarations of the form $(h : Q)$ in Γ_1, and we define* Certificate$(D) = \Gamma_1, h(n) : P_1 \vdash WF$.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ TypeVar $(X : Type) \in \Gamma$

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash WF$. We define* Certificate$(D) = \Gamma_1 \vdash X : Type$.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash Prop : Type}$ Prop

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash WF$. We define* Certificate$(D) = \Gamma_1 \vdash Prop : Type$.

- $\dfrac{\Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}$ Pi

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1, x : A_1 \vdash B_1 : Type$. We define* Certificate$(D) = \Gamma_1 \vdash \Pi x : A_1.B_1 : Type$.

- $\dfrac{\Gamma, x : A \vdash P : Prop}{\Gamma \vdash \{x : A \mid P\} : Type}$ Subtype

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1, x : A_1 \vdash P_1 : Prop$ by Proposition 9.2.2. We define* Certificate$(D) = \Gamma_1 \vdash \{x : A_1 : P_1\} : Type$.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ EltVar $(x : A) \in \Gamma$

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash WF$, and there exists at least one declaration of the form $(x : A_1) \in \Gamma_1$*

such that $[\![A_1]\!] = A$. *We consider the first declaration* $(x : A_1) \in \Gamma_1$ *having this property, and define* $\textsc{Certificate}(D) = \Gamma_1 \vdash x : A_1$.

- $\dfrac{\Gamma, x : A \vdash P : Prop}{\Gamma \vdash \forall x : A.P : Prop}\ \textsc{Forall}$

  We consider D_1 *the direct subderivation of* D. $\textsc{Certificate}(D_1)$ *has the form* $\Gamma_1, x : A_1 \vdash P_1 : Prop$ *by Proposition 9.2.2. We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash \Pi x : A_1.P_1 : Prop$.

- $\dfrac{\Gamma, P \vdash Q : Prop}{\Gamma \vdash P \Rightarrow Q : Prop}\ \textsc{Imply}$

  We consider D_1 *the direct subderivation of* D. $\textsc{Certificate}(D_1)$ *has the form* $\Gamma_1, h_1 : P_1 \vdash Q_1 : Prop$ *by Proposition 9.2.2. We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash \Pi h_1 : P_1.Q_1 : Prop$.

- $\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}\ \textsc{Lam}$

  We consider D_1 *the direct subderivation of* D. $\textsc{Certificate}(D_1)$ *has the form* $\Gamma_1, x : A_1 \vdash t_1 : B_1$. *We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash \lambda x : A_1.t_1 : \Pi x : A_1.B_1$.

- $\dfrac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}\ \textsc{App}$

  We consider D_1 *and* D_2 *the direct subderivations of* D. $\textsc{Certificate}(D_2)$ *has the form* $\Gamma_2 \vdash u_2 : A_2$ *and, by Proposition 9.2.2,* $\textsc{Certificate}(D_1)$ *has the form* $\Gamma_1 \vdash t_1 : \Pi x : A_1.B_1$. *We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash t_1 u_2 : B_1[u_2/x]$. *By induction hypothesis and Proposition 9.1.1,* $[\![B_1[u_2/x]]\!] = [\![B_1]\!][[\![u_2]\!]/x] = B[u/x]$, *hence the erasure of this judgements is* $\Gamma \vdash tu : B[u/x]$, *as expected.*

- $\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : Type}{\Gamma \vdash t : \{x : A \mid P\}}\ \textsc{SubtypeIntro}$

  We consider D_1, D_2, *and* D_3 *the direct subderivations of* D. $\textsc{Certificate}(D_1)$ *has the form* $\Gamma_1 \vdash t_1 : A_1$, $\textsc{Certificate}(D_2)$ *has the form* $\Gamma_2 \vdash p_2 : P_2'$, *and, by Proposition 9.2.2,* $\textsc{Certificate}(D_3)$ *has the form* $\Gamma_3 \vdash \{x : A_3 \mid P_3\} : Type$. *We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash \langle t_1, p_2 \rangle_{\{x:A_3|P_3\}} : \{x : A_3 \mid P_3\}$.

- $\dfrac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash t : A}\ \textsc{SubtypeElim1}$

  We consider D_1 *the direct subderivation of* D. $\textsc{Certificate}(D_1)$ *has the form* $\Gamma_1 \vdash t_1 : \{x : A_1 \mid P_1\}$ *by Proposition 9.2.2. We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash \pi_1(t_1) : A_1$.

- $\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash B : Type}{\Gamma \vdash t : B}$ TypeConversion $A \equiv_\beta B$

  We consider D_1 and D_2 the direct subderivations of D. Certificate(D_1) *has the form $\Gamma_1 \vdash t_1 : A_1$ and* Certificate(D_1) *has the form $\Gamma_2 \vdash B_2 : Type$. We define* Certificate$(D) = \Gamma_1 \vdash t_1 : B_2$.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash P}$ Axiom $P \in \Gamma$

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash WF$. As $[\![\Gamma_1]\!] = \Gamma$, there exists some declaration of the form $(h_1 : P_1) \in \Gamma_1$ such that $[\![P_1]\!] = P$. We consider $(h_1 : P_1) \in \Gamma_1$ the first declaration satisfying this property and define* Certificate$(D) = \Gamma_1 \vdash h_1 : P_1$.

- $\dfrac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q}$ ImplyIntro

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1, h_1 : P_1 \vdash q_1 : Q_1$. We define* Certificate$(D) = \Gamma_1 \vdash \lambda h_1 : P_1.q_1 : \Pi h_1 : P_1.Q_1$.

- $\dfrac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}$ ImplyElim

  We consider D_1 and D_2 the direct subderivations of D. Certificate(D_2) *has the form $\Gamma_2 \vdash p_2 : P_2$ and, by Proposition 9.2.2,* Certificate(D_1) *has the form $\Gamma_1 \vdash p_1 : Q_1'$ where Q_1' admits a normal form with respect to $\triangleright_*$ which has the form $\Pi h : P_1.Q_1$. We define* Certificate$(D) = \Gamma_1 \vdash p_1 p_2 : Q_1[p_2/h]$. *By induction hypothesis and Proposition 9.1.1, $[\![Q_1[p_2/h]]\!] = [\![Q_1]\!] = Q$, hence the erasure of this judgement is $\Gamma \vdash Q$, as expected.*

- $\dfrac{\Gamma, x : A \vdash P}{\Gamma \vdash \forall x : A.P}$ ForallIntro

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1, x : A_1 \vdash p_1 : P_1$. We define* Certificate$(D) = \Gamma_1 \vdash \lambda x : A_1.p_1 : \Pi x : A_1.P_1$.

- $\dfrac{\Gamma \vdash \forall x : A.P \qquad \Gamma \vdash t : A}{\Gamma \vdash P[t/x]}$ ForallElim

  We consider D_1 and D_2 the direct subderivations of D. Certificate(D_2) *has the form $\Gamma_2 \vdash t_2 : A_2$ and, by Proposition 9.2.2,* Certificate(D_1) *has the form $\Gamma_1 \vdash p_1 : P_1'$ where P_1' admits a normal form with respect to $\triangleright_*$ which has the form $\Pi x : A_1.P_1$. We define* Certificate$(D) = \Gamma_1 \vdash p_1 t_2 : P_1[t_2/x]$. *By induction hypothesis and Proposition 9.1.1, $[\![P_1[t_2/x]]\!] = [\![P_1]\!][[\![t_2]\!]/x] = P[t/x]$, hence the erasure of this judgement is $\Gamma \vdash P[t/x]$, as expected.*

- $\dfrac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash P[t/x]}$ SubtypeElim2

  We consider D_1 the direct subderivation of D. Certificate(D_1) *has the form $\Gamma_1 \vdash t_1 : \{x : A_1 \mid P_1\}$ by Proposition 9.2.2. We define* Certificate$(D) = \Gamma_1 \vdash \pi_2(t_1) : P_1[\pi_1(t_1)/x]$.

- $\dfrac{\Gamma \vdash P \qquad \Gamma \vdash Q : Prop}{\Gamma \vdash Q}$ PropConversion $P \equiv_\beta Q$

  We consider D_1 and D_2 the direct subderivations of D. Certificate(D_1) *has the form $\Gamma_1 \vdash p_1 : P_1$ and, by Proposition 9.2.2,* Certificate(D_1) *has the form $\Gamma_2 \vdash Q_2 : Prop$. We define* Certificate$(D) = \Gamma_1 \vdash p_1 : Q_2$.

9.3 Soundness of the synthesis of certificates

This section is dedicated to the proof of soundness of the algorithm Certificate: for any PVS-Cert derivation D, Certificate(D) is derivable in PVS-Cert. This soundness property follows from several propositions. The first important one is Proposition 9.3.2, which expresses in particular that for any terms M and N which are either expressions, types, or $Type$, whenever $[\![M]\!] = [\![N]\!]$, then $M \equiv_* N$. This statement is presented together with a similar property dealing with contexts instead of terms. This latter case is more complex in the general case, as the equality of the erasures of two stratified contexts does not imply that they are related through $\equiv_*$. Nevertheless, we prove that it is sufficient whenever these two contexts belong respectively to some PVS-Cert judgement in the range of Certificate.

We first present and prove the following property of contexts of PVS-Cert judgements in the range of Certificate, which is the main lemma of Proposition 9.3.2.

Proposition 9.3.1. *For any PVS-Core derivation D, we consider Γ the context of* Certificate(D) *and $h_1 : P_1, ..., h_n : P_n$ the context obtained from Γ by selecting only declarations of the form $(h : P)$ for some proof variable h. In this setting, $h_1 = h(1), h_2 = h(2), ..., h_n = h(n)$.*

Proof. The proof is straightforward by induction on the derivation D. □

We conclude the expected proposition 9.3.2 as follows.

Proposition 9.3.2. *The two following statements hold.*

- *For all terms M and N which are either expressions, types, or $Type$, whenever $[\![M]\!] = [\![N]\!]$, then $M \equiv_* N$.*
- *For all PVS-Core derivations D_1 and D_2, we consider Γ_1 and Γ_2 the respective contexts of* Certificate(D_1) *and* Certificate(D_2). *Whenever $[\![\Gamma_1]\!] = [\![\Gamma_2]\!]$, then $\Gamma_1 \equiv_* \Gamma_2$.*

Proof. The first statement is straightforward by induction on M and N. The second statement is proved by induction on Γ_1, using the first statement together with Proposition 9.3.1. □

The next important proposition needed in the proof of soundness of CERTIFICATE is Proposition 9.3.4, which is the converse of Proposition 9.1.2: for any terms M and N which are either expressions, types, or $Type$, whenever $[\![M]\!] \equiv_\beta [\![N]\!]$, then $M \equiv_{\beta *} N$.

This proposition will be established from Proposition 9.3.2 as well as the definition of a new function from PVS-Core terms to PVS-Cert terms, $[\cdot]$, which will be proved to be a right inverse for $[\![\cdot]\!]$. the definition of this new function is the following.

Definition 9.3.1. *We define the following function $[\cdot]$ from PVS-Core terms to PVS-Cert expressions, types, or Type recursively. As in the definition of $[\![\cdot]\!]$, this new definition relies on the fact that the sets of expression variables and type variables $\mathcal{V}_{expressions}$ and $\mathcal{V}_{types}$ are shared in PVS-Cert and PVS-Core.*

- $[Type] = Type$
- $[X] = X$
- $[Prop] = Prop$
- $[\Pi x : A.B] = \Pi x : [A].[B]$
- $[\{x : A \mid P\}] = \{x : [A] \mid [P]\}$
- $[x] = x$
- $[P \Rightarrow Q] = \Pi h : [P].[Q]$
- $[\forall x : A.P] = \Pi x : [A].[P]$
- $[\lambda x : A.t] = \lambda x : [A].[t]$
- $[t\ u] = [t][u]$

Proposition 9.3.3. *The translation $[\cdot]$ is a right inverse of $[\![\cdot]\!]$: for any PVS-Core term M, $[\![[M]]\!] = M$.*

Proof. The proof is straightforward by induction on M. □

The following property of $[\cdot]$ will be the main lemma used in the proof of .

Lemma 9.3.1. *The following statements hold.*

- *For any PVS-Core expressions t and u and any expression variable x, $[t[u/x]] = [t][[u]/x]$.*
- *For any PVS-Core terms M and N, whenever $M \to_\beta N$, then $[M] \to_\beta [N]$.*

- *For any PVS-Core terms M and N, whenever $M \equiv_\beta N$, then $[M] \equiv_\beta [N]$.*

Proof. The first statement is proved by induction on t. As $[\cdot]$ and substitution are stable by α-conversion, we can suppose without loss of generality that no bound variable in t is free in u nor equal to x. In this setting, all cases are straightforward.

The second statement is proved by induction on M. The only difficult case corresponds to the case of an application, when $M \triangleright_\beta N$. In this setting, M has the form $(\lambda x : A.t)u$ and $N = t[u/x]$, and we can conclude using the first statement.

The last statement is straightforward by induction on the length of the conversion. □

We finally conclude the expected converse of Proposition 9.1.2 as follows.

Proposition 9.3.4. *For all terms M and N which are either expressions, types, or Type, whenever $[\![M]\!] \equiv_\beta [\![N]\!]$, then $M \equiv_{\beta *} N$.*

Proof. By Proposition 9.3.3 followed by Proposition 9.3.2, $[[\![M]\!]] \equiv_* M$ and $[[\![N]\!]] \equiv_* N$. Hence, it is sufficient to prove $[[\![M]\!]] \equiv_\beta [[\![N]\!]]$. This last requirement holds by Proposition 9.3.1, as $[\![M]\!] \equiv_\beta [\![N]\!]$. □

The last proposition needed to prove the soundness of CERTIFICATE is the following. It shows that, in specific situations, the operation of normalization through $\triangleright_*$ (which erases the coercions $\pi_1(\cdot)$ and $\langle \cdot, M \rangle_T$ the head of terms exclusively) is type preserving.

Proposition 9.3.5. *For any derivable PVS-Cert judgement of the form $\Gamma \vdash P : Prop$, if P admits a normal form with respect to $\triangleright_*$ which has the form $\Pi v : M.T$, then $\Gamma \vdash \Pi v : M.T : Prop$ is derivable.*

Proof. This result is a direct consequence of the following stronger statement: for any derivable PVS-Cert judgement of the form $\Gamma \vdash t : \{x_n ... \{x_1 : Prop \mid Q_1\} ... \mid Q_n\}$, if P admits a normal form with respect to $\triangleright_*$ which has the form $\Pi v : M.T$, then $\Gamma \vdash \Pi v : M.T : Prop$ is derivable. The proof is done by induction on t. As t admits a normal form with respect to $\triangleright_*$ which has the form $\Pi v : M.T$, the only possible cases are the following.

- t has the form $\pi_1(u)$. In this case, by the subderivations theorem, there exists some derivable judgement of the form $\Gamma \vdash u : \{x_{n+1} : A \mid Q_{n+1}\}$, where $A \equiv_{\beta *} \{x_n ... \{x_1 : Prop \mid Q_1\} ... \mid Q_n\}$. By the stratification theorem, A is a type. By straightforward induction on n, using Theorem 5.3.2 and the fact that A is a type, A has the form $\{x'_n ... \{x'_1 : Prop \mid Q'_1\} ... \mid Q'_n\}$. Hence, we can conclude the expected result by induction hypothesis.

- t has the form $\langle u, p \rangle_A$. In this case, by the subderivations theorem, A has the form $\{x : B \mid Q\}$ with $\{x : B \mid Q\} \equiv_{\beta *} \{x_n ... \{x_1 : Prop \mid Q_1\} ... \mid Q_n\}$ and $\Gamma \vdash u : B$ is derivable. By Theorem 5.3.2, $n \geq 1$, and $B \equiv_{\beta *} \{x_{n-1} ... \{x_1 : Prop \mid$

$Q_1\}... \mid Q_{n-1}\}$. By the stratification theorem, B is a type. By straightforward induction on n, using Theorem 5.3.2 and the fact that B is a type, B has the form $\{x'_{n-1}...\{x'_1 : Prop \mid Q'_1\}... \mid Q'_{n-1}\}$. Hence, we can conclude the expected result by induction hypothesis.

- $t = \Pi v : M.T$. In this case, by the subderivations theorem and the stratification theorem, $Prop \equiv_{\beta *} \{x_n...\{x_1 : Prop \mid Q_1\}... \mid Q_n\}$. Hence, by Theorem 5.3.2, $n = O$. In this setting, by hypothesis, $\Gamma \vdash \Pi v : M.T : Prop$ is derivable.

□

The following theorem is the expected soundness property for CERTIFICATE. Most of the proof is straightforward. The most complex parts of this proof correspond, on the one hand, to PVS-Core with more than one premise. In those cases, we use the notion of conversion of contexts defined in Definition 5.7.1 and the admissibility of rules for context conversions presented in Proposition 5.7.1. On the other hand, the cases IMPLYELIM and FORALLELIM involve, by definition of CERTIFICATE, some normalization with respect to $\triangleright_*$. The specific difficulties related to this normalization are handled using Proposition will be used.

As in the definition of the CERTIFICATE, Proposition 9.2.1 will be used implicitly in this proof.

Theorem 9.3.1. *For any PVS-Core derivation D,* CERTIFICATE(D) *is derivable in PVS-Cert.*

Proof. The proof is done by induction on D. The possible cases for the rule instances matching the last inference step of D are the following.

- $\dfrac{}{\varnothing \vdash WF}$ EMPTY

  This case is straightforward using the PVS-Cert rule EMPTY.

- $\dfrac{\Gamma \vdash WF}{\Gamma, X : Type \vdash WF}$ TYPEDECL $X \in \mathcal{V}_{types} \backslash DV(\Gamma)$

  We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash WF$. In this setting, CERTIFICATE(D) $= \Gamma_1, X : Type \vdash WF$. By induction hypothesis followed by the rule SORT, $\Gamma_1 \vdash Type : Kind$ is derivable in PVS-Cert. Moreover, as $X \notin DV(\Gamma)$, $X \notin DV(\Gamma_1)$. Hence, applying the rule DECL, $\Gamma_1, X : Type \vdash WF$ is derivable in PVS-Cert.

- $\dfrac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash WF}$ ELTDECL $x \in \mathcal{V}_{expressions} \backslash DV(\Gamma)$

We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash A_1 : Type$. In this setting, CERTIFICATE(D) $= \Gamma_1, x : A_1 \vdash WF$. By induction hypothesis, $\Gamma_1 \vdash A_1 : Type$ is derivable in PVS-Cert. Moreover, as $x \notin DV(\Gamma)$, $x \notin DV(\Gamma_1)$. Hence, applying the rule DECL, $\Gamma_1, x : A_1 \vdash WF$ is derivable in PVS-Cert.

- $\dfrac{\Gamma \vdash P : Prop}{\Gamma, P \vdash WF}$ ASSUMPTION

We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash P_1 : Prop$ by Proposition 9.2.2. We consider n the number of declarations of the form $(h : Q)$ in Γ_1. In this setting, CERTIFICATE(D) $= \Gamma_1, h(n) : P \vdash WF$. By induction hypothesis, $\Gamma_1 \vdash P_1 : Prop$ is derivable in PVS-Cert. Moreover, by Proposition 9.3.1, $h(n) \notin DV(\Gamma_1)$. Hence, applying the rule DECL, $\Gamma_1, h(n) : P_1 \vdash WF$ is derivable in PVS-Cert.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ TYPEVAR $(X : Type) \in \Gamma$

We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash WF$, and there exists at least one declaration $(X : Type) \in \Gamma$. In this setting, CERTIFICATE(D) $= \Gamma_1 \vdash X : Type$. By induction hypothesis, $\Gamma_1 \vdash WF$ is derivable in PVS-Cert. Hence, applying the rule VAR, $\Gamma_1 \vdash X : Type$ is derivable.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash Prop : Type}$ PROP

The expected result follows by induction hypothesis, applying the PVS-Cert rule SORT.

- $\dfrac{\Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}$ PI

We consider D_1 the second direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1, x : A_1 \vdash B_1 : Type$. In this setting, CERTIFICATE(D) $= \Gamma_1 \vdash \Pi x : A_1.B_1 : Type$. By induction hypothesis, $\Gamma_1, x : A_1 \vdash B_1 : Type$ is derivable in PVS-Cert. Hence, by the subderivations theorem and the stratification theorem, $\Gamma_1 \vdash A_1 : Type$ is also derivable. Thus, applying the rule PROD, $\Gamma_1 \vdash \Pi x : A_1.B_1 : Type$ is derivable in PVS-Cert.

- $\dfrac{\Gamma, x : A \vdash P : Prop}{\Gamma \vdash \{x : A \mid P\} : Type}$ SUBTYPE

We consider D_1 the second direct subderivation of D. By Proposition 9.2.2, CERTIFICATE(D_1) has the form $\Gamma_1, x : A_1 \vdash P_1 : Prop$. Hence, CERTIFICATE($D$) = $\Gamma_1 \vdash \{x : A_1 : P_1\} : Type$. By induction hypothesis, $\Gamma_1, x : A_1 \vdash P_1 : Prop$ is derivable in PVS-Cert. Hence, by the subderivations theorem and the stratification theorem, $\Gamma_1 \vdash A_1 : Type$ is also derivable. Thus, applying the rule SUBTYPE, $\Gamma_1 \vdash \{x : A_1 : P_1\} : Type$ is derivable in PVS-Cert.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash x : A}$ ELTVAR $(x : A) \in \Gamma$

  We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash WF$, and there exists at least one declaration of the form $(x : A_1) \in \Gamma_1$ such that $[\![A_1]\!] = A$. We consider the first declaration $(x : A_1) \in \Gamma_1$ having this property. In this setting, CERTIFICATE(D) = $\Gamma_1 \vdash x : A_1$. By induction hypothesis, $\Gamma_1 \vdash WF$ is derivable in PVS-Cert. Hence, applying the rule VAR, $\Gamma_1 \vdash X : Type$ is derivable.

- $\dfrac{\Gamma, x : A \vdash P : Prop}{\Gamma \vdash \forall x : A.P : Prop}$ FORALL

  We consider D_1 the second direct subderivation of D. By Proposition 9.2.2, CERTIFICATE(D_1) has the form $\Gamma_1, x : A_1 \vdash P_1 : Prop$. Hence, CERTIFICATE($D$) = $\Gamma_1 \vdash \Pi x : A_1.P_2 : Prop$. By induction hypothesis, $\Gamma_1, x : A_1 \vdash P_1 : Prop$ is derivable in PVS-Cert. Hence, by the subderivations theorem and the stratification theorem, $\Gamma_1 \vdash A_1 : Type$ is also derivable. Thus, applying the rule PROD, $\Gamma_1 \vdash \Pi x : A_1.P_1 : Prop$ is derivable in PVS-Cert.

- $\dfrac{\Gamma, P \vdash Q : Prop}{\Gamma \vdash P \Rightarrow Q : Prop}$ IMPLY

  We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1, h_1 : P_1 \vdash Q_2 : Prop$ by Proposition 9.2.2. In this setting, CERTIFICATE(D) = $\Gamma_1 \vdash \Pi h_1 : P_1.Q_1 : Prop$. By induction hypothesis, $\Gamma_1, h_1 : P_1 \vdash Q_1 : Prop$ is derivable in PVS-Cert. Hence, by the subderivations theorem and the stratification theorem, $\Gamma_1 \vdash P_1 : Prop$ is also derivable. Thus, applying the rule PROD, $\Gamma_1 \vdash \Pi h_1 : P_1.Q_1 : Prop$ is derivable in PVS-Cert.

- $\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}$ LAM

  We consider D_1 the first direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1, x : A_1 \vdash t_1 : B_1$. In this setting, CERTIFICATE(D) = $\Gamma_1 \vdash \lambda x : A_1.t_1 : \Pi x : A_1.B_1$. By induction hypothesis, $\Gamma_1, x : A_1 \vdash t_1 : B_1$ is derivable in PVS-Cert. Hence, by the subderivations theorem, Theorem 5.2.3, and the stratification

theorem, $\Gamma_1 \vdash A_1 : Type$ and $\Gamma_1, x : A_1 \vdash B_1 : Type$ are derivable in PVS-Cert. Thus, applying the rule Prod and the rule Lam, $\Gamma_1 \vdash \lambda x : A_1.t_1 : \Pi x : A_1.B_1$ is derivable in PVS-Cert.

- $\dfrac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}$ App

  We consider D_1 and D_2 the direct subderivations of D. Certificate(D_2) has the form $\Gamma_2 \vdash u_2 : A_2$ and, by Proposition 9.2.2, Certificate(D_1) has the form $\Gamma_1 \vdash t_1 : \Pi x : A_1.B_1$. In this setting, Certificate(D) $= \Gamma_1 \vdash t_1 u_2 : B_1[u_2/x]$. By induction hypothesis, $\Gamma_1 \vdash t_1 : \Pi x : A_1.B_1$ and $\Gamma_2 \vdash u_2 : A_2$ are derivable in PVS-Cert. By Theorem 5.2.3 followed by the subderivations theorem and the stratification theorem, $\Gamma_1 \vdash A_1 : Type$ is derivable. Therefore, by Proposition 9.3.2 followed by Proposition 5.7.1, $\Gamma_1 \vdash u_2 : A_1$ is derivable in PVS-Cert. Hence, applying the rule App, $\Gamma_1 \vdash t_1 u_2 : B_1[u_2/x]$ is derivable as expected.

- $\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : Type}{\Gamma \vdash t : \{x : A \mid P\}}$ SubtypeIntro

  We consider D_1, D_2, and D_3 the direct subderivations of D. Certificate(D_1) has the form $\Gamma_1 \vdash t_1 : A_1$, Certificate($D_2$) has the form $\Gamma_2 \vdash p_2 : P_2'$, and, by Proposition 9.2.2, Certificate(D_3) has the form $\Gamma_3 \vdash \{x : A_3 \mid P_3\} : Type$. In this setting, Certificate(D) $= \Gamma_1 \vdash \langle t_1, p_2 \rangle_{\{x:A_3|P_3\}} : \{x : A_3 \mid P_3\}$. By induction hypothesis, $\Gamma_1 \vdash t_1 : A_1$, $\Gamma_2 \vdash p_2 : P_2'$, and $\Gamma_3 \vdash \{x : A_3 \mid P_3\} : Type$ are derivable in PVS-Cert. We first prove that $\Gamma_1 \vdash P_3[t_1/x] : Prop$ is derivable in the following way.

  By the subderivations theorem, $\Gamma_1 \vdash WF$ is derivable. Therefore, by Proposition 9.3.2 followed by Proposition 5.7.1, $\Gamma_1 \vdash \{x : A_3 \mid P_3\} : Type$ is derivable in PVS-Cert. As a consequence, by the subderivations theorem, $\Gamma_1, x : A_3 \vdash P_3 : Prop$ is derivable. By the subderivations theorem again, $\Gamma_1 \vdash A_3 : Type$ is also derivable. Hence, by Proposition 9.3.2, we can apply conversion to the first premise to obtain a derivation of $\Gamma_1 \vdash t_1 : A_3$. We conclude by the substitution theorem that $\Gamma_1 \vdash P_3[t_1/x] : Prop$ is derivable.

  On the other hand, by Proposition 9.1.1, $[\![P_3[t_1/x]]\!] = [\![P_3]\!][[\![t_1]\!]/x] = P[t/x] = [\![P_2]\!]$. Therefore, by Proposition 9.3.2 followed by Proposition 5.7.1, $\Gamma_1 \vdash p_2 : P_3[t_1/x]$ is derivable in PVS-Cert. Finally, applying the rule Pair, the judgement $\Gamma_1 \vdash \langle t_1, p_2 \rangle_{\{x:A_3|P_3\}} : \{x : A_3 \mid P_3\}$ is derivable as expected.

- $\dfrac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash t : A}$ SubtypeElim1

The expected result follows by induction hypothesis, applying the PVS-Cert rule PROJ1.

- $\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash B : Type}{\Gamma \vdash t : B}$ TYPECONVERSION $A \equiv_\beta B$

We consider D_1 and D_2 the direct subderivations of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash t_1 : A_1$ and CERTIFICATE(D_1) has the form $\Gamma_2 \vdash B_2 : Type$. In this setting, CERTIFICATE(D) $= \Gamma_1 \vdash t_1 : B_2$. By induction hypothesis, $\Gamma_1 \vdash t_1 : A_1$ and $\Gamma_2 \vdash B_2 : Type$ are derivable.

By the subderivations theorem, $\Gamma_1 \vdash WF$ is derivable. Hence, by Proposition 9.3.2 followed by Proposition 5.7.1, $\Gamma_1 \vdash B_2 : Type$ is derivable in PVS-Cert. On the other hand, by Proposition 9.3.4, $A_1 \equiv_{\beta *} B_2$. Hence, applying conversion, $\Gamma_1 \vdash t_1 : B_2$ is derivable.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash P}$ AXIOM $P \in \Gamma$

We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash WF$, and there exists at least one declaration of the form $(h_1 : P_1) \in \Gamma_1$ such that $[\![P_1]\!] = P$. We consider $(h_1 : P_1) \in \Gamma_1$ the first declaration satisfying this property. In this setting, CERTIFICATE(D) $= \Gamma_1 \vdash h_1 : P_1$. By induction hypothesis, $\Gamma_1 \vdash WF$ is derivable in PVS-Cert. As $(h_1 : P_1) \in \Gamma_1$, we conclude the expected result applying the rule VAR.

- $\dfrac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q}$ IMPLYINTRO

We consider D_1 the direct subderivation of D. CERTIFICATE(D_1) has the form $\Gamma_1, h_1 : P_1 \vdash q_1 : Q_1$. In this setting, CERTIFICATE(D) $= \Gamma_1 \vdash \lambda h_1 : P_1.q_1 : \Pi h_1 : P_1.Q_1$. By induction hypothesis, $\Gamma_1, h_1 : P_1 \vdash q_1 : Q_1$ is derivable in PVS-Cert. Hence, by the subderivations theorem, Theorem 5.2.3, and the stratification theorem, $\Gamma_1 \vdash P_1 : Prop$ and $\Gamma_1, h_1 : P_1 \vdash Q_1 : Prop$ are derivable in PVS-Cert. Thus, applying the rule PROD and the rule LAM, $\Gamma_1 \vdash \lambda h_1 : P_1.q_1 : \Pi h_1 : P_1.Q_1$ is derivable in PVS-Cert.

- $\dfrac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}$ IMPLYELIM

We consider D_1 and D_2 the direct subderivations of D. CERTIFICATE(D_2) has the form $\Gamma_2 \vdash p_2 : P_2$ and, by Proposition 9.2.2, CERTIFICATE(D_1) has the form $\Gamma_1 \vdash p_1 : Q_1'$ where Q_1' admits a normal form with respect to $\triangleright_*$ which has the

form $\Pi h : P_1.Q_1$. In this setting, $\textsc{Certificate}(D) = \Gamma_1 \vdash p_1 p_2 : Q_1[p_2/h]$. By induction hypothesis, $\Gamma_1 \vdash p_1 : Q_1'$ and $\Gamma_2 \vdash p_2 : P_2$ are derivable in PVS-Cert. By Proposition 5.2.3 and the stratification theorem, $\Gamma_1 \vdash Q_1' : \mathit{Prop}$ is derivable in PVS-Cert. Hence, by Proposition 9.3, $\Gamma_1 \vdash \Pi h : P_1.Q_1 : \mathit{Prop}$ is derivable as well. As $Q_1' \equiv_{\beta *} \Pi h : P_1.Q_1$, we conclude by conversion that $\Gamma_1 \vdash p_1 : \Pi h : P_1.Q_1$ is derivable.

By Theorem 5.2.3 followed by the subderivations theorem and the stratification theorem, $\Gamma_1 \vdash P_1 : \mathit{Prop}$ is derivable. Therefore, by Proposition 9.3.2 followed by Proposition 5.7.1, $\Gamma_1 \vdash p_2 : P_1$ is derivable in PVS-Cert. Hence, applying the rule App, $\Gamma_1 \vdash p_1 p_2 : Q_1[p_2/h]$ is derivable as expected.

- $\dfrac{\Gamma, x : A \vdash P}{\Gamma \vdash \forall x : A.P}$ ForallIntro

  We consider D_1 the direct subderivation of D. $\textsc{Certificate}(D_1)$ has the form $\Gamma_1, x : A_1 \vdash p_1 : P_1$. In this setting, $\textsc{Certificate}(D) = \Gamma_1 \vdash \lambda x : A_1.p_1 : \Pi x : A_1.P_1$. By induction hypothesis, $\Gamma_1, x : A_1 \vdash p_1 : P_1$ is derivable in PVS-Cert. Hence, by the subderivations theorem, Theorem 5.2.3, and the stratification theorem, $\Gamma_1 \vdash A_1 : \mathit{Type}$ and $\Gamma_1, x_1 : A_1 \vdash P_1 : \mathit{Prop}$ are derivable in PVS-Cert. Thus, applying the rule Prod and the rule Lam, $\Gamma_1 \vdash \lambda x_1 : A_1.p_1 : \Pi x_1 : A_1.P_1$ is derivable in PVS-Cert.

- $\dfrac{\Gamma \vdash \forall x : A.P \qquad \Gamma \vdash t : A}{\Gamma \vdash P[t/x]}$ ForallElim

  We consider D_1 and D_2 the direct subderivations of D. $\textsc{Certificate}(D_2)$ has the form $\Gamma_2 \vdash t_2 : A_2$ and, by Proposition 9.2.2, $\textsc{Certificate}(D_1)$ has the form $\Gamma_1 \vdash p_1 : P_1'$ where P_1' admits a normal form with respect to $\triangleright_*$ which has the form $\Pi x : A_1.P_1$. In this setting, $\textsc{Certificate}(D) = \Gamma_1 \vdash p_1 t_2 : P_1[t_2/x]$. By induction hypothesis, $\Gamma_1 \vdash p_1 : P_1'$ and $\Gamma_2 \vdash t_2 : A_2$ are derivable in PVS-Cert. By Proposition 5.2.3 and the stratification theorem, $\Gamma_1 \vdash P_1' : \mathit{Prop}$ is derivable. Hence, by Proposition 9.3, $\Gamma_1 \vdash \Pi x : A_1.P_1 : \mathit{Prop}$ is derivable as well. As $P_1' \equiv_{\beta *} \Pi x : A_1.P_1$, we conclude by conversion that $\Gamma_1 \vdash p_1 : \Pi x : A_1.P_1$ is derivable.

  By Theorem 5.2.3 followed by the subderivations theorem and the stratification theorem, $\Gamma_1 \vdash A_1 : \mathit{Type}$ is derivable. Therefore, by Proposition 9.3.2 followed by Proposition 5.7.1, $\Gamma_1 \vdash t_2 : A_1$ is derivable in PVS-Cert. Hence, applying the rule App, $\Gamma_1 \vdash p_1 t_2 : P_1[t_2/x]$ is derivable as expected.

- $\dfrac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash P[t/x]}$ SubtypeElim2

The expected result follows by induction hypothesis, applying the PVS-Cert rule PROJ2.

- $\dfrac{\Gamma \vdash P \qquad \Gamma \vdash Q : Prop}{\Gamma \vdash Q}$ PROPCONVERSION $P \equiv_\beta Q$

We consider D_1 and D_2 the direct subderivations of D. CERTIFICATE(D_1) has the form $\Gamma_1 \vdash p_1 : P_1$ and, by Proposition 9.2.2, CERTIFICATE(D_1) has the form $\Gamma_2 \vdash Q_2 : Prop$. In this setting, CERTIFICATE(D) $= \Gamma_1 \vdash p_1 : Q_2$. By induction hypothesis, $\Gamma_1 \vdash p_1 : P_1$ and $\Gamma_2 \vdash Q_2 : Prop$ are derivable.

By the subderivations theorem, $\Gamma_1 \vdash WF$ is derivable. Hence, by Proposition 9.3.2 followed by Proposition 5.7.1, $\Gamma_1 \vdash Q_2 : Prop$ is derivable in PVS-Cert. On the other hand, by Proposition 9.3.4, $P_1 \equiv_{\beta *} Q_2$. Hence, applying conversion, $\Gamma_1 \vdash t_1 : B_2$ is derivable.

□

Chapter 10

Transposing PVS-Cert results in PVS-Core

We end the first part of this work showing the consequences, in PVS-Core, of the different results obtained for PVS-Cert, such as the theorem of decidability of type-checking in PVS-Cert (7.1.1), the type preservation theorem 5.5.2, the conservativity theorem 8.5.2 over higher-order logic, but also the cut elimination theorem 6.5.3 and its corollaries. This analysis is done through the correspondence established in the previous chapter between PVS-Core and PVS-Cert.

This overview begins with an answer to the first question addressed in the current work: defining a system of verifiable certificates for PVS-Core.

10.1 Using PVS-Cert as a system of verifiable certificates for PVS-Core

The type-checking algorithm for PVS-Cert presented in Chapter 7 can be combined in the following way with the encoding of PVS-Core derivations into PVS-Cert presented in Chapter 9 to use PVS-Cert proof judgements as proof certificates for PVS-Core.

Definition 10.1.1 (PVS-Cert proof judgements as PVS-Core certificates)**.** *A PVS-Cert judgement* $\Gamma \vdash p : P$ *can be used as a certificate for its PVS-Core erasure* $[\![\Gamma]\!] \vdash [\![P]\!]$ *(defined in Definition 9.1.1), to be checked using the type-checking algorithm* CHECK-TYPE$(\Gamma \mid p \mid P)$ *(defined in Definition 7.1.3).*

On the one hand, this way of using PVS-Cert judgements as certificates is sound: whenever CHECK-TYPE$(\Gamma \mid p \mid P)$ *succeeds, then, by Proposition 7.2.1, the judgement* $\Gamma \vdash p : P$ *is derivable, and, by Theorem 9.1.1,* $[\![\Gamma]\!] \vdash [\![P]\!]$ *is derivable in PVS-Core.*

On the other hand, valid certificates can be generated for arbitrary PVS-Core theorems in the following way. Given some PVS-Core judgement $\Delta \vdash Q$ *derivable through*

some derivation D, we can use the PVS-Cert judgement CERTIFICATE(D) *as a certificate of $\Delta \vdash Q$: indeed, using the notations $\Gamma \vdash p : P$ for* CERTIFICATE(D)*, the following statements hold.*

- *By definition of* CERTIFICATE, $[\![\Gamma]\!] = \Delta$ *and* $[\![P]\!] = Q$*, hence this judgement is a certificate for $\Delta \vdash Q$.*
- *By Theorem 9.3.1, $\Gamma \vdash p : P$ is derivable in PVS-Cert. Hence, by Proposition 7.4.1,* CHECK-TYPE($\Gamma \mid p \mid P$) *succeeds: this certificate is valid.*

10.2 Transposing PVS-Cert properties in PVS-Core

Several properties proved for PVS-Cert can be transposed directly for PVS-Core through the correspondence described in Chapter 9. We present four examples of such transpositions, corresponding, respectively, to Theorem 5.2.3, to the type preservation theorem 5.5.2, to the strong normalization theorem 6.5.2, and to the conservativity theorem 8.5.2 over higher-order logic.

10.2.1 A provable proposition is well-typed

The adaptation of Theorem 5.2.3 in PVS-Core has two forms: on the one hand, a proved proposition is well-typed by *Prop*, and, on the other hand, the type of a well-typed expression is well-formed. More precisely, we formalize this adaptation as follows.

Theorem 10.2.1. *The two following statements hold.*

- *For any derivable PVS-Core judgement $\Gamma \vdash P$, the judgement $\Gamma \vdash P : Prop$ is derivable too.*
- *For any derivable PVS-Core judgement $\Gamma \vdash t : A$, the judgement $\Gamma \vdash A : Type$ is derivable too.*

Proof. The two cases are similar. We present the first one as an illustration. We define $\Gamma' \vdash p : P'$ the image of some derivation of $\Gamma \vdash P$ through the algorithm CERTIFICATE. By definition, $[\![\Gamma']\!] = \Gamma$ and $[\![P']\!] = P$. By Theorem 9.3.1, $\Gamma' \vdash p : P'$ is derivable in PVS-Cert. Hence, by Theorem 5.2.3 and the stratification theorem, $\Gamma' \vdash P' : Prop$ is derivable as well in PVS-Cert. As a consequence, by Theorem 9.1.1, $\Gamma \vdash P : Prop$ is derivable in PVS-Core. □

10.2.2 Type preservation in PVS-Core

We present the following transposition to PVS-Core of the PVS-Cert type preservation theorem (5.5.2).

Theorem 10.2.2 (Type preservation for $\twoheadrightarrow_\beta$)**.** *Given any derivable PVS-Core judgement $\Gamma \vdash P$ (resp. $\Gamma \vdash t : A$, $\Gamma \vdash A : Type$), the following statements hold.*

- *For any derivable PVS-Core judgement* $\Gamma \vdash P$, *whenever* $P \twoheadrightarrow_\beta Q$, $\Gamma \vdash Q$ *is derivable.*

- *For any derivable PVS-Core judgement* $\Gamma \vdash t : A$, *whenever* $t \twoheadrightarrow_\beta u$ *and* $A \twoheadrightarrow_\beta B$, $\Gamma \vdash u : B$ *is derivable*

- *For any derivable PVS-Core judgement* $\Gamma \vdash A : Type$, *whenever* $A \twoheadrightarrow_\beta B$, $\Gamma \vdash B : Type$ *is derivable.*

The proof of this adaptation of the PVS-Cert type preservation theorem requires the following lemma.

Lemma 10.2.1. *The following statements hold.*

- *For any PVS-Core types* $A \to_\beta B$, *whenever there exists some PVS-Cert derivable judgement* $\Gamma \vdash A' : Type$ *with* $[\![A']\!] = A$, *then there exists some PVS-Cert type* B' *such that* $A' \twoheadrightarrow_{\beta\sigma} B'$ *and* $[\![B']\!] = B$.

- *For any PVS-Core expressions* $t \to_\beta u$, *whenever there exists some PVS-Cert derivable judgement* $\Gamma \vdash t' : A'$ *with* $[\![t']\!] = t$, *then there exists some PVS-Cert expression* u' *such that* $t' \twoheadrightarrow_{\beta\sigma} u'$ *and* $[\![u']\!] = u$.

The hypotheses of derivability of $\Gamma \vdash A' : Type$ in the first statement and $\Gamma \vdash t' : A'$ in the second statement are the most important requirements to make these statements hold. Indeed, a straightforward counter-example without this constraint would be to take some PVS-Core expression of the form $\pi_1(\lambda x : A.x)y$: on the one hand, $[\![\pi_1(\lambda x : A.x)y]\!] = (\lambda x : A.x)y \to_\beta y$, but $\pi_1(\lambda x : A.x)y$ is in normal form with respect to the reduction $\to_{\sigma\beta}$.

The proof of the lemma uses these derivability constraints as follows.

Proof. The two statements are proved together by induction on the type A in the first statement and the expression t in the second statement.

In the first statement, the possible cases for A are the following.

- The cases $Prop$ and X cannot occur.

- In the case $\Pi x : A_1.A_2$, one of the following holds.

  - $B = \Pi x : B_1.A_2$, where $A_1 \to_\beta B_1$. By Proposition 9.2.2, A' has the form $\Pi x : A'_1.A'_2$ with $[\![A'_1]\!] = A_1$ and $[\![A'_2]\!] = A_2$. By the subderivations theorem and the stratification theorem, $\Gamma \vdash A'_1 : Type$ is derivable. Hence, by induction hypothesis, there exists some PVS-Cert type B'_1 such that $\Pi x : A'_1.A'_2 \twoheadrightarrow_{\beta\sigma} \Pi x : B'_1.A'_2$ and $[\![B'_1]\!] = B_1$.

– $B = \Pi x : A_1.B_2$, where $A_2 \rightarrow_\beta B_2$. By Proposition 9.2.2, A' has the form $\Pi x : A'_1.A'_2$ with $[\![A'_1]\!] = A_1$ and $[\![A'_2]\!] = A_2$. By the subderivations theorem and the stratification theorem, $\Gamma, x : A'_1 \vdash A'_2 : Type$ is derivable. Hence, by induction hypothesis, there exists some PVS-Cert type B'_2 such that $\Pi x : A'_1.A'_2 \twoheadrightarrow_{\beta\sigma} \Pi x : A'_1.B'_2$ and $[\![B'_2]\!] = B_2$.

In the second statement, we first reduce the general case to the case where t' does not begin with the constructions $\langle \cdot, M \rangle_T$ nor $\pi_1(\cdot)$ as follows. We consider t'' the subterm of t' obtained by removing iteratively all possible constructions $\langle \cdot, M \rangle_T$ nor $\pi_1(\cdot)$ at the head of t': in this setting, t'' doesn't begin with one of these constructions. By straightforward induction on the depth of the occurrence of t'' in t', there exists a derivable PVS-Cert judgement of the form $\Gamma \vdash t'' : A''$ for some type A''. In the restricted case, we suppose that the expected result holds: there exists a term u'' such that $t'' \rightarrow_{\beta\sigma} u''$ and $[\![u'']\!] = u$. In this setting, we consider the term u' obtained from t' by replacing the subterm t'' by u''. On the one hand, $t' \twoheadrightarrow_{\beta\sigma} u'$. On the other hand, $[\![u']\!] = [\![u'']\!] = u$.

It remains to be proved that the expected result holds in this restricted case, which is done by case analysis on t.

- The case x cannot occur.
- In the cases $\Pi x : A.P$ (resp. $\Pi x : P.Q$ or $\lambda x : A.t_1$), we first use the fact that the head t' does not correspond to the constructions $\langle \cdot, M \rangle_T$ nor $\pi_1(\cdot)$ to conclude that t' must have the form $\Pi x : A'.P'$ (resp. $\Pi x : P'.Q'$ or $\lambda x : A'.t'_1$). Using this fact, the remaining of the proof follows exactly what was done in the case $\Pi x : A_1.A_2$ in the analysis of the first statement.
- The case $t_1 t_2$ is proved in the same way if $t \not\triangleright_\beta u$.
- The case $t_1 t_2$ is proved in the following way if $t \triangleright_\beta u$. In this case, t_1 has the form $\lambda x : B_0.t_0$ and $u = t_0[t_2/x]$. If necessary, we first α-rename t_1 to ensure $x \notin DV(\Gamma)$. As $[\![t']\!] = t$ and as t' doesn't have the form $\langle t'_0, M \rangle_T$ nor $\pi_1(t'_0)$, the only possible form of t' is an application $t'_1 t'_2$, where $[\![t'_1]\!] = \lambda x : B.t_0$ and $[\![t'_2]\!] = t_2$. By the subderivations theorem, there exists two derivable judgements of the form $\Gamma \vdash t'_1 : \Pi x : B'.B''$ (as $x \notin DV(\Gamma)$, the free variable theorem ensures that the variable x can be used as a bound variable in $\Pi x : B'.B''$) and $\Gamma \vdash t'_2 : B'$.

  We consider t''_1 the normal form of t'_1 with respect to the reduction $\twoheadrightarrow_\sigma$, which exists by the strong normalization theorem. By the type preservation theorem, as the judgement $\Gamma \vdash t'_1 : \Pi x : B'.B''$ is derivable, so is the judgement $\Gamma \vdash t''_1 : \Pi x : B'.B''$. We consider the unique way to write $t''_1 = e[t'''_1]$ where the elimination context e (referring to Definition 6.1.1) contains only the projections $\pi_1(\cdot)$ and where the head of t'''_1 is not $\pi_1(\cdot)$. We consider the possible forms for t'''_1. As $[\![t'''_1]\!] = [\![t''_1]\!] = [\![t'_1]\!] = \lambda x : B_0.t_0$, t'''_1 can only have the form $\lambda x : B'_0.t'_0$ or $\langle t'_0, p \rangle_{B'_0}$.

We first prove that the second case is impossible. In this second case, as t_1'' is in normal form with respect to $\twoheadrightarrow_\sigma$, the elimination context e cannot contain any projection $\pi_1(\cdot)$: hence, $t_1'' = t_1''' = \langle t_0', p\rangle_{B_0'}$, hence $\Gamma \vdash \langle t_0', p\rangle_{B_0'} : \Pi x : B'.B''$ is derivable. By the subderivations theorem, B_0' has the form $\{y : B_0'' \mid P'\}$ and $\{y : B_0'' \mid P'\} \equiv_{\beta *} \Pi x : B'.B''$, which is impossible by Theorem 5.3.2.

As a consequence, t_1''' has the form $\lambda x : B_0'.t_0'$ and, as $[\![t_1''']\!] = \lambda x : B_0.t_0$, $[\![B_0']\!] = B_0$ and $[\![t_0']\!] = t_0$. Using these facts, we prove that the elimination context e contains no projection $\pi_1(\cdot)$ as follows. We suppose that e does contain at least one projection $\pi_1(\cdot)$. In this case, $\pi_1(\lambda x : B_0'.t_0')$ is a subterm of t_1'': by straightforward induction on the depth of e, some judgement of the form $\Gamma \vdash \pi_1(\lambda x : B_0'.t_0') : B_1'$ is derivable for some type B_1'. Hence, by the subderivations theorem, some judgement of the form $\Gamma \vdash \lambda x : B_0'.t_0' : \{y : B_0'' \mid P'\}$ is derivable. This is impossible by the subderivations theorem followed by Theorem 5.3.2. As a consequence, e contains no projection $\pi_1(\cdot)$, hence $t_1'' = t_1''' = \lambda x : B_0'.t_0'$.

In this setting, $t_1' \twoheadrightarrow_\sigma \lambda x : B_0'.t_0'$, and $t' = t_1't_2' \twoheadrightarrow_{\beta\sigma} t_0'[t_2'/x]$. We consider the term $u' = t_0'[t_2'/x]$. On the one hand, $t' \twoheadrightarrow_{\beta\sigma} u'$. On the other hand, by Theorem 9.1.1, $[\![u']\!] = [\![t_0']\!][[\![t_2']\!]/x] = t_0[t_2/x] = u$.

□

Using this lemma, the proof of Theorem 10.2.2 is the following.

Proof. [Theorem 10.2.2] All cases are similar. We present the second one as an illustration. We define $\Gamma' \vdash t' : A'$ the image of some derivation of $\Gamma \vdash t : A$ through the algorithm CERTIFICATE. By definition of CERTIFICATE, $[\![\Gamma']\!] = \Gamma$, $[\![t']\!] = t$, and $[\![A']\!] = A$. By Theorem 9.3.1, $\Gamma' \vdash t' : A'$ is derivable in PVS-Cert. By straightforward induction, using Lemma 10.2.1, there exists some expression u' such that $t' \twoheadrightarrow u'$ and $[\![u']\!] = u$. On the other hand, by Theorem 5.2.3 and the stratification theorem, $\Gamma' \vdash A' : Type$ is derivable: as previously, by straightforward induction, using Lemma 10.2.1, there exists some type B' such that $A' \twoheadrightarrow B'$ and $[\![B']\!] = B$.

By the PVS-Cert type preservation theorem 5.5.2 applied twice, $\Gamma' \vdash u' : B'$ is derivable in PVS-Cert. Hence, by Theorem 9.1.1, $\Gamma \vdash u : B$ is derivable in PVS-Core. □

10.2.3 Strong normalization in PVS-Core

We present the following transposition to PVS-Core of the PVS-Cert strong normalization theorem 6.5.2.

Theorem 10.2.3 (Strong normalization for $\twoheadrightarrow_\beta$)**.** *The types and expressions appearing in the conclusion of a derivable PVS-Core judgement are strongly normalizing under* $\rightarrow_\beta$*:*

- *For any derivable PVS-Core judgement $\Gamma \vdash P$, the expression P is strongly normalizing under $\rightarrow_\beta$.*
- *For any derivable PVS-Core judgement $\Gamma \vdash t : A$, the expression t and the type A are strongly normalizing under $\rightarrow_\beta$.*
- *For any derivable PVS-Core judgement $\Gamma \vdash A : Type$, the type A is strongly normalizing under $\rightarrow_\beta$.*

Proof. All cases are similar. We present the second one as an illustration. We define $\Gamma' \vdash t' : A'$ the image of some derivation of $\Gamma \vdash t : A$ through the algorithm CERTIFICATE. By definition of CERTIFICATE, $[\![\Gamma']\!] = \Gamma$, $[\![t']\!] = t$, and $[\![A']\!] = A$. By Theorem 9.3.1, $\Gamma' \vdash t' : A'$ is derivable in PVS-Cert. By the strong normalization theorem 6.5.2, the terms t' and A' are strongly normalizing under $\rightarrow_{\beta *}$. We consider the translation $[\cdot]$ presented in Definition 9.3.1. By straightforward induction on types and expressions, $t' \twoheadrightarrow_* [t]$ and $A' \twoheadrightarrow_* [A]$, hence $[t]$ and $[A]$ are strongly normalizing under $\rightarrow_{\beta *}$. In particular, they are strongly normalizing under $\rightarrow_\beta$. By Lemma 9.3.1 and by straightforward induction on the length of reductions, for any reduction of the form $M_1 \rightarrow_\beta M_2 \rightarrow_\beta ... \rightarrow_\beta M_n$ in PVS-Core, there exists a corresponding reduction of the form $[M_1] \rightarrow_\beta [M_2] \rightarrow_\beta ... \rightarrow_\beta [M_n]$ in PVS-Cert. Hence, t and A are strongly normalizing under $\rightarrow_\beta$. □

10.2.4 PVS-Core is a conservative extension of higher-order logic

We present a transposition to PVS-Core of the PVS-Cert conservativity theorem 8.5.2 over higher-order logic. In the same way as PVS-Cert is a conservative extension of λ-HOL, presented in Definition 4.2.1 as a subsystem of PVS-Cert, PVS-Core is a conservative extension of higher-order logic, presented in Definition 3.3.1 as a subsystem of PVS-Core.

In this transposition, we will use the fact that the erasure function $[\![\cdot]\!]$ from PVS-Cert to PVS-Core can be restricted to translate λ-HOL judgements to higher-order logic. This is formalized and proved as follows.

Proposition 10.2.1. *Every derivable λ-HOL judgement either has the form $\Gamma \vdash Type : Kind$ or admits an image through $[\![\cdot]\!]$ which belongs to higher-order logic. In this latter case, this judgement is derivable in higher-order logic.*

Proof. The proof is analogous to the proof of Theorem 9.1.1. The first part of the theorem is a consequence of the stratification theorem and the fact that, by straightforward induction on PVS-Cert types and expressions, the image of a type or expression belonging to the syntax of λ-HOL through $[\![\cdot]\!]$ belongs to the syntax of higher-order logic.

The second part is done by strong induction on the height of λ-HOL derivations. All cases are treated as in the proof Theorem 9.1.1. The only required adaptation is the modification of the induction hypotheses and the conclusions, replacing derivability in PVS-Core by derivability in higher-order logic. □

In a second step, we prove the following proposition relating the erasure function $[\![\cdot]\!]$ from PVS-Cert to PVS-Core and the translation from PVS-Cert to λ-HOL presented in Definition 8.2.5. This latter translation was referred to as $[\![\cdot]\!]$ in Chapter 8 where it is presented and used, but it will be denoted $[\![\cdot]\!]_{\lambda HOL}$ in the following of this section to disambiguate it from the erasure function $[\![\cdot]\!]$ from PVS-Cert to PVS-Core.

Proposition 10.2.2. *The following statements hold.*

- *For every PVS-Cert type A such that $[\![A]\!]$ is a higher-order logic type, $[\![[\![A]\!]_{\lambda HOL}]\!] = [\![A]\!]$.*
- *For every PVS-Cert expression t such that $[\![t]\!]$ is a higher-order logic expression, $[\![[\![t]\!]_{\lambda HOL}]\!] = [\![t]\!]$.*
- *For every PVS-Cert stratified context of the form Γ such that $[\![\Gamma]\!]$ is a higher-order logic context, $[\![[\![\Gamma]\!]_{\lambda HOL}]\!] = [\![\Gamma]\!]$.*

This proposition is a useful property for PVS-Cert terms that are not λ-HOL terms, but that have a higher-order logic erasure: Proposition 10.2.2 shows that, although the image of such a term through the translation $[\![\cdot]\!]_{\lambda HOL}$ is not identical to the original term, the two have the same PVS-Core erasures. This proposition is proved as follows.

Proof. The two first statements are proved together by induction on PVS-Cert types and expressions. In the first statement, the possible cases are the following.

- The case X, $Prop$ are straightforward
- The case $\Pi x : A.B$ is straightforward by induction hypothesis
- The case $\{x : A \mid P\}$ cannot occur as $[\![A]\!]$ is a higher-order logic type.

In the second statement, the possible cases are the following.

- The cases x is straightforward
- The cases $\Pi x : A.P$, $\Pi h : P.Q$, $\lambda x : A.t$, and $t\ u$ are straightforward by induction hypothesis
- In the case $\langle t, M\rangle_A$, by induction hypothesis, $[\![[\![t]\!]_{\lambda HOL}]\!] = [\![t]\!]$, hence $[\![[\![\langle t, M\rangle_A]\!]_{\lambda HOL}]\!] = [\![[\![t]\!]_{\lambda HOL}]\!] = [\![t]\!] = [\![\langle t, M\rangle_A]\!]$.
- In the case $\langle \pi_1(t)$, by induction hypothesis, $[\![[\![t]\!]_{\lambda HOL}]\!] = [\![t]\!]$, hence $[\![[\![\pi_1(t)]\!]_{\lambda HOL}]\!] = [\![[\![t]\!]_{\lambda HOL}]\!] = [\![t]\!] = [\![\pi_1(t)]\!]$.

Using the two first statements, the third one is proved straightforwardly by induction on the length of the context Γ. □

Finally, we present the following transposition from PVS-Cert to PVS-Core of the conservativity theorem proved in Theorem 8.5.2.

Theorem 10.2.4. *PVS-Core is a conservative extension of higher-order logic: for every judgement of the form $\Gamma \vdash P : Prop$ derivable in higher-order logic, P is provable in Γ in PVS-Core if and only if P is provable in Γ in higher-order logic.*

Proof. As higher-order logic is a subsystem of PVS-Core, the derivability of $\Gamma \vdash P$ in higher-order logic implies its derivability in PVS-Core.

Conversely, we suppose that the judgement $\Gamma \vdash P$ admits some PVS-Core derivation D. We define $\Gamma' \vdash p : P' = \textsc{Certificate}(D)$. By Theorem 9.3.1, $\Gamma' \vdash p : P'$ is derivable in PVS-Cert. Hence, by Theorem 8.5.1, there exists a proof q such that $[\![\Gamma']\!]_{\lambda HOL} \vdash q : [\![P']\!]_{\lambda HOL}$ is derivable in λ-HOL. As a consequence, by Proposition 10.2.1, $[\![[\![\Gamma']\!]_{\lambda HOL}]\!] \vdash [\![[\![P']\!]_{\lambda HOL}]\!]$ is derivable in higher-order logic. In this setting, we can conclude the expected result by proving that $[\![[\![\Gamma']\!]_{\lambda HOL}]\!] = \Gamma$ and $[\![[\![P']\!]_{\lambda HOL}]\!] = P$. This is done as follows.

By definition of the function Certificate, $[\![\Gamma']\!] = \Gamma$, which is a higher-order logic context, and $[\![P']\!] = P$, which is a higher-order logic expression. Hence, by Proposition 10.2.2, $[\![[\![\Gamma']\!]_{\lambda HOL}]\!] = \Gamma$, which is a higher-order logic context, and $[\![[\![P']\!]_{\lambda HOL}]\!] = P$. Hence, $\Gamma \vdash P$ is derivable in higher-order logic, as expected. □

10.3 Using cut elimination in PVS-Cert to study PVS-Core logical properties

The logical properties established for PVS-Cert through cut elimination can be transposed to PVS-Core. The simplest of these properties is the consistency of PVS-Core.

Theorem 10.3.1. *The system PVS-Core is consistent: the judgement $\vdash \forall x : Prop.x$ is not derivable.*

Proof. If the judgement $\vdash \forall x : Prop.x$ admits a PVS-Core derivation D, we define $\vdash p : P = \textsc{Certificate}(D)$. By definition, $[\![P]\!] = \forall x : Prop.x = [\![\Pi x : Prop.x]\!]$. Hence, by proposition 9.3.4, $P \equiv_{\beta *} \Pi x : Prop.x$. As $\vdash \Pi x : Prop.x : Prop$ is derivable in PVS-Cert, we can apply the conversion rule to conclude that $\vdash p : \Pi x : Prop.x$ is derivable in PVS-Cert, which is impossible by Theorem 6.7.1. □

The more complex example of the analysis of Leibniz's definition of equality was presented in Theorem 6.7.2 as another corollary of cut elimination in PVS-Cert cut elimination. It is transposed to PVS-Core in the following way.

Theorem 10.3.2. *In PVS-Core, Leibniz's definition of equality matches conversion: whenever some judgement of the form $\vdash \forall x : (\Pi y : A.Prop).xt \Rightarrow xu$ is derivable, then $t \equiv_\beta u$.*

This theorem is more difficult to transpose to PVS-Core as the proposition $\forall x : (\Pi y : A.Prop).xt \Rightarrow xu$ is not necessarily a higher-order logic expression. Given a derivation D

proving this proposition, the strategy in its proof is to analyze the PVS-Cert proposition proved in CERTIFICATE(D) from its head towards its deeper subterms to conclude that it can be converted to some PVS-Cert proposition on which Theorem 6.7.2 is applicable. More precisely, the proof is the following.

Proof. If the judgement $\vdash \forall x : (\Pi y : A.Prop).xt \Rightarrow xu$ admits a PVS-Core derivation D, we define $\vdash p : P = \text{CERTIFICATE}(D)$: by definition, $[\![P]\!] = \forall x : (\Pi y : A.Prop).xt \Rightarrow xu$. Hence, by Proposition 9.2.2, P admits a normal form with respect to $\triangleright_*$ which has the form $\Pi x : B.P'$. By straightforward induction on the corresponding reduction, $[\![B]\!] = \Pi y : A.Prop$ and $[\![P']\!] = xt \Rightarrow xu$.

By Theorem 9.3.1, $\vdash p : P$ is derivable in PVS-Cert. Hence, by Theorem 5.2.3 followed by Proposition 9.3, $\vdash \Pi x : B.P' : Prop$ is derivable as well. As a consequence, by the subderivations theorem and the renaming theorem, the judgement $x : B \vdash P' : Prop$ is derivable. By Proposition 9.2.2, P' admits a normal form with respect to $\triangleright_*$ which has the form $\Pi h : P_1.Q_1$. By straightforward induction on the corresponding reduction, $[\![P_1]\!] = xt$ and $[\![Q_1]\!] = xu$. Applying Proposition 9.3 again, we conclude that the judgement $x : B \vdash \Pi h : P_1.Q_1 : Prop$ is derivable. Thus, by the subderivations theorem, the renaming theorem, and the stratification theorem, the judgement $x : B, h : P_1 \vdash Q_1 : Prop$ and the judgement $x : B \vdash P_1 : Prop$ are derivable.

By the strong normalization theorem 6.5.2, P_1 and Q_1 admits two respective normal forms P_2 and Q_2 with respect to $\twoheadrightarrow_\sigma$. By straightforward induction on the length of these respective reductions, $[\![P_2]\!] = xt$ and $[\![Q_2]\!] = xu$. By the type preservation theorem 5.5.2, the judgement $x : B \vdash P_2 : Prop$ and the judgement $x : B, h : P_1 \vdash Q_2 : Prop$ are derivable.

We consider the unique way to write $P_2 = e[t_0]$ where e contains only the projections $\pi_1(\cdot)$ and where the head of t_0 doesn't have the form $\pi_1(\cdot)$. We first show that t_0 in an application by discarding all other possibilities, as follows. As $[\![t_0]\!] = [\![P_2]\!] = xt$, the only other possibility is $t_0 = \langle t_1', p_1 \rangle_{A_1}$. In this case, as P_2 is in normal form with respect to $\rightarrow_\sigma$, e does not contain any projection $\pi_1(\cdot)$: thus, $P_2 = t_0$. Hence, by the subderivations theorem, A_1 has the form $\{y : Q \mid A_2\}$, and $\{y : Q \mid A_2\} \equiv_{\beta *} Prop$. This is not possible by Theorem 5.3.2.

As a consequence, t_0 is an application. We write $t_0 = t_1 t_2$: in this setting, $[\![t_1]\!] = x$ and $[\![t_2]\!] = t$. Iterating the subderivations theorem from the derivation of $x : B \vdash P_2 : Prop$ and applying the stratification theorem, we first notice that there exists some derivable judgement of the form $x : B \vdash t_1 : \Pi x_{t_1} : A_{t_1}.B_{t_1}$. Using this fact, we first show that $t_1 = x$ in the following way. We consider the unique way to write $t_1 = e'[t_1']$ where e' contains only the projections $\pi_1(\cdot)$ and where the head of t_1' doesn't have the form $\pi_1(\cdot)$. We first show that $t_1' = x$ by discarding all other possibilities, as follows. As $[\![t_1']\!] = [\![t_1]\!] = x$, the only other possible form for t_1' is $\langle t_1'', p_1 \rangle_{A_1}$. In this case, as P_2 is in normal form with respect to $\rightarrow_\sigma$, e' does not contain any projection $\pi_1(\cdot)$: thus,

$t_1 = t'_1$. Hence, by the subderivations theorem, A_1 has the form $\{y : Q \mid A_2\}$, and $\{y : Q \mid A_2\} \equiv_{\beta*} \Pi x_{t_1} : A_{t_1}.B_{t_1}$. This is not possible by Theorem 5.3.2.

As a consequence, $t'_1 = x$. We show that $t_1 = t'_1 = x$ in the following way. If this is not the case, then e' contains at least one projection $\pi_1(\cdot)$: iterating the subderivations theorem from the derivation of $x : B \vdash t_1 : \Pi x_{t_1} : A_{t_1}.B_{t_1}$, we conclude that there exists some derivable judgement of the form $x : B \vdash x : \{y : Q \mid B'\}$. As, by the thinning theorem followed by the application of the rule VAR, the judgement $x : B \vdash x : B$ is derivable as well, by Theorem 5.6.1, $B \equiv_{\beta*} \{y : Q \mid B'\}$. On the other hand, as $[\![B]\!] = \Pi y : A.Prop$, by Proposition 9.2.2, B has the form $\Pi x : A'.Prop$, where $[\![A']\!] = A$. Hence, the equation $B \equiv_{\beta*} \{y : Q \mid B'\}$ cannot hold by Theorem 5.3.2. As a consequence, $t_1 = t'_1 = x$.

Using this fact, we show that $P_2 = t_0 = xt_2$ in the following way. If this is not the case, then e contains at least one projection $\pi_1(\cdot)$: iterating the subderivations theorem from the derivation of $x : B \vdash P_2 : Prop$, we first conclude that there exists some derivable judgement of the form $x : B \vdash xt_2 : \{y : Q \mid B'\}$. Applying the subderivations theorem again, there exists some derivable judgement of the form $x : B \vdash x : \Pi z : B''.B'''$ is derivable, where $B'''[t_2/z] \equiv_{\beta*} \{y : Q \mid B'\}$. By Theorem 5.6.1, $\Pi x : A'.Prop \equiv_{\beta*} \Pi z : B''.B'''$. Hence, by Theorem 5.3.2, hence $Prop \equiv_{\beta*} B'''$. By Theorem 5.3.2 again, this implies that $B''' = Prop$, from which we conclude $Prop \equiv_{\beta*} \{y : Q \mid B'\}$. Applying Theorem 5.3.2 again, this situation cannot occur.

As a consequence, $P_2 = xt_2$. In the same way as the data of a derivation of judgement $x : B \vdash P_2 : Prop$ was used to prove that P_2 has the form xt_2 with $[\![t_2]\!] = t$, the derivation of the judgement $x : B, h : P_1 \vdash Q_2 : Prop$ is used to prove that Q_2 has the form xu_2 with $[\![u_2]\!] = u$.

Using the rule DECL, the judgement $x : B, h : xt_2 \vdash WF$ is derivable. On the other hand, by Theorem 5.5.1, $P_1 \equiv_{\beta*} P_2 = xt_2$. Hence, using the notion of context conversion defined in Definition 5.7.1 and Proposition 5.7.1, we conclude that the judgement $x : B, h : xt_2 \vdash xu_2 : Prop$ is derivable. As the judgement $x : B \vdash xt_2 : Prop$ is also derivable, we conclude applying the rule PROD that the judgement $x : B \vdash \Pi h : xt_2.xu_2 : Prop$ is derivable. On the other hand, by the subderivations theorem and the stratification theorem, the judgement $\vdash B : Type$ is derivable. Hence, applying the rule PROD, the judgement $\vdash \Pi x : B.\Pi h : xt_2.xu_2 : Prop$ is derivable.

As $[\![\Pi x : B.\Pi h : xt_2.xu_2 : Prop]\!] = \forall x : (\Pi y : A.Prop).xt \Rightarrow xu = [\![P]\!]$, by Theorem 9.3.4, $\Pi x : B.\Pi h : xt_2.xu_2 : Prop \equiv_{\beta*} P$. As a consequence, the conversion rule can be applied to conclude that the judgement $\vdash p : \Pi x : B.\Pi h : xt_2.xu_2$ is derivable. As $B = \Pi x : A'.Prop$, Theorem 6.7.2 can be applied to conclude that $t_2 \equiv_{\beta*} u_2$. Hence, applying Proposition 9.1.2, $t \equiv_\beta u$. □

Chapter 11

Conclusion

11.1 Summary of the main contributions

The first contribution of this work is the definition of PVS-Core (Chapter 3), a minimal system expressing the extension of higher-order logic with predicate subtyping, obtained from the practice of predicate subtyping in PVS (Chapter 2). Besides its minimality, the main design choice for this system is the introduction of a definitional equality, referred to as *conversion*, corresponding to syntactical equality modulo β-equivalence.

The second contribution of this work is the definition of PVS-Cert (Chapter 4), a language of verifiable certificates for PVS-Core, designed as the addition of explicit proof terms to PVS-Core as well as the addition, at the level of expressions, of explicit coercions based on these proof terms. The addition of explicit proof terms follows the Curry-Howard correspondence in the sense that PVS-Cert proofs terms are typed by their corresponding formulas. On the other hand, the addition of explicit coercions ensures the decidability of type-checking. A terminating, sound and complete type-checking algorithm for PVS-Cert is presented in Chapter 7.

In order to maintain a simple correspondence between PVS-Core and PVS-Cert, *conversion* in PVS-Cert is not defined as in PVS-Core but through a more distinctive notion, $\equiv_{\beta *}$ (Definition 4.1.3), corresponding to syntactical equality modulo β-equivalence *and coercion erasure*. We present a translation from PVS-Cert to PVS-Core and, at the level of derivation trees, a translation from PVS-Core to PVS-Cert (Chapter 9). These translations are used in Chapter 10 together with the type-checking algorithm of PVS-Cert to define how to use PVS-Cert as a language of a verifiable proofs for PVS-Core (Definition 10.1.1).

PVS-Cert is very similar to the formalism of PTSs extended with dependent pairs. Nevertheless, instead of the case of the reduction $\rightarrow_{\beta\sigma}$ (Definition 4.2.2) in PTSs with dependent pairs, $\rightarrow_{\beta *}$ is not a type preserving reduction in PVS-Cert. We prove that $\rightarrow_{\beta\sigma}$ is a type preserving reduction in PVS-Cert (Theorem 5.5.2). As a consequence, it

defines, when applied to proof terms, a notion of *cut elimination*.

The strong normalization of the reductions $\rightarrow_{\beta*}$ and $\rightarrow_{\beta\sigma}$ is proved in Chapter 6 (Theorem 6.5.2). While the termination of the reduction $\rightarrow_{\beta*}$ is at the core of the termination of the type-checking algorithm defined for PVS-Cert, the termination of the reduction $\rightarrow_{\beta\sigma}$ provides a cut elimination theorem (Theorem 6.5.3), which is a useful tool to analyze specific properties of PVS-Cert and PVS-Core – and thus predicate subtyping itself –, from consistency (Theorems 6.7.1 and 10.3.1) to more complex theorems such as the analysis of Leibniz's equality (Theorems 6.7.2 and 10.3.2).

Last, we have presented in Chapter 8 a translation (Definition 8.2.5) from PVS-Cert to its restriction to the PTS λ-HOL. Using this translation and the fact that any PVS-Cert formula in which predicate subtyping is not explicitly used is translated as itself through this translation, we have proved that PVS-Cert is a conservative extension of λ-HOL (Theorem 8.5.2), and as a consequence that PVS-Core is a conservative extension of higher-order logic (Theorem 10.2.4). These theorems allow to reduce the question of the provability of any proposition using predicate subtyping to the question of the provability of a proposition formulated in pure higher-order logic. Hence, they provide useful tools to study the properties of predicate subtyping, in complement of the cut elimination theorem.

11.2 Perspectives for PVS

One of the most interesting perspectives from this work is the definition of a *kernel* for PVS, in the sense of a back-end language in which the whole PVS language – or at least a large part of it – could be expressed. As detailed in Section 11.2.1, this kernel could be defined as a well-suited extension of PVS-Core. Moreover, PVS-Cert could be extended accordingly to define a system of *verifiable certificates* for this kernel, and thus for PVS itself.

11.2.1 Extending PVS-Cert and PVS-Cert

A *kernel* language wouldn't require to include all features of PVS: most of them could be either axiomatized or encoded into this language. For instance, following two examples mentioned in Chapter 2, propositional extensionality wouldn't be required as it can be added as an axiom (as suggested in Section 2.2.2), and the connective `FALSE` wouldn't be required as it could be encoded as `FORALL (P : bool) : P` (as suggested in Section 2.1.2).

In theory, most features of PVS could be removed from such a kernel language through axiomatizations and encodings. We can conjecture that, whenever the use of arbitrarily complex encodings was allowed, PVS-Core itself would be used as a kernel for PVS. Proceeding further, PVS-Core wouldn't be a minimal kernel for PVS, as PVS-Core

can be expressed in turn into higher-order logic using the encoding described in Chapter 8 (Definition 8.2.5).

However, in practice, using heavy encodings can be prohibitive to scale up to large theories, or simply to recognize a PVS theory from its expression in the kernel. For these reasons, we consider on the contrary that PVS-Core, which is a minimal system for the expression of predicate subtyping, should be extended in several directions to obtain a system which could be used as a kernel for PVS. In this setting, PVS-Cert should be extended accordingly to obtain a system of verifiable certificates for this kernel, and thus for PVS.

We consider that the first extensions to be added to PVS-Core and PVS-Cert to define respectively a kernel and a system of verifiable certificates for PVS are the following.

- Some form of polymorphism at the level of types should be added to both systems to express the abstract and modular formalizations which can be performed in PVS. Such a mechanism is indeed indispensable when using PVS for the formalization of mathematics or for the definition and the verification of algorithms. For instance, it is used in PVS to define a theory of groups or a theory of lists using an abstract parameter for the underlying type of elements, allowing to use these theories anywhere else through some instantiation with a concrete type. These abstract parameters are referred to as *formal parameters* (mentioned in Section 2.1.1), while their concrete instantiations are referred to as *actual parameters*. When using a theory with abstract types, all abstract parameters must be fully instantiated, either by the user or through the *name resolution* mechanism (mentioned as well in Section 2.1.1). This constraint, which can be thought as a restriction to *prenex polymorphism*, ensures in particular the impossibility to express Girard's paradox [40] in PVS. We conjecture that PVS-Cert could be extended with the same form of polymorphism by following the formalization referred to as *predicative polymorphism* in [18], and that PVS-Core could be extended accordingly.

- Some mechanisms dedicated to induction and recursion should be added as well to both systems. They could be used to express several useful advanced features of PVS such as *datatypes*, *inductive definitions*, and *recursive definitions* (mentioned in Section 2.1.1), but also more basic features such as *tuple types*, *records* (mentioned in Section 2.1.2), including their dependent versions. The formalization of *numeric expressions* (mentioned in Section 2.1.2 as well) could be also eased using such mechanisms. Among the many formalizations of induction and recursion in type theory, it seems possible, for instance, to adapt the definitions given in [20] or in [78], which are based on the calculus of constructions, to PVS-Cert and then to PVS-Core.

We conjecture that the result of strong normalization proved in Chapter 6 can be extended with these additional features. We also conjecture that the conservativity theorem presented in Chapter 8 (and adapted to PVS-Core in Theorem 10.2.4) can be

extended as well in the sense that these extensions of PVS-Cert (resp. PVS-Core) are conservative over corresponding extensions of λ-HOL (resp. higher-order logic).

A last extension suggested for PVS-Core and PVS-Cert to define respectively a kernel and a system of verifiable certificates for PVS is the addition of a mechanism of constant definitions. More precisely, this system would alter PVS-Core and PVS-Cert by adding the equivalent of PVS's *interpreted declarations* (mentioned in Section 2.1.1), and by extending the conversion relation to allow *reasoning modulo the unfolding of constant definitions*. This mechanism wouldn't be useful only because PVS theories usually use several layers of definitions, but also because many features of PVS could be expressed in a kernel language through constant definitions. The previous example of the expression of the constant connective `FALSE` as `FORALL (P : bool) : P` is such a case. Without a mechanism of constant definitions, one would either have to keep all of these definitions unfolded, which would lead to a significant increase of the size of expressions, or to add corresponding equality axioms – for instance, `FALSE = FORALL (P : bool) : P` –, which would lead to a significant increase of the size of proofs, as these equality axioms would be used intensively. On the contrary, using a mechanism of constant definitions, the constant connective `FALSE` as well as several other features of PVS could be removed from the kernel without affecting neither the expressions nor the proofs. One possible way to formalize a mechanism of constant definitions for PVS-Core and PVS-Cert is to adapt the notion of *PTS with definitions* presented in [67].

As the obtained extensions of PVS-Core and PVS-Cert would remain much smaller than PVS, several layers of axiomatization and encodings should be defined to express PVS in it. One first step in this direction would be to explore further the suggestions presented in Chapter 2: for instance, expressing extra connectives and quantifiers through simple definitions (as investigated in Section 2.1.2), expressing the more flexible typing rules of predicate subtyping in PVS through η-expansions (as investigated in Section 2.2.1), or adding extra reasoning capabilities through the addition of axioms (as investigated in Section 2.2.2).

11.2.2 The problem of extracting certificates from PVS

Before using such an extension of PVS-Cert as a system of verifiable certificates for PVS, one preliminary work is to have, in practice, the possibility to extract externally exploitable data from PVS. As mentioned in Chapter 1, PVS is based on two separate tools, the *type-checker* and the *prover*: in order to build such certificates, data from both tools have to be gathered. However, extracting information from these tools is difficult: the *type-checker* outputs no information during typechecking other than the TCCs, and the only pieces of information emitted by the *prover* are proof scripts, which are well-suited to rerun proofs internally, but not to be usable outside PVS in practice. For this reason, it appears that both the *type-checker* and the *prover* should be instrumented to extract a sufficient amount of information to build externally verifiable certificates written in some well-suited extension of PVS-Cert.

In Part II, a first prototype is presented for the extraction from PVS of proof certificates that can be verified externally. As a prototype, this work admits some restrictions and consequently some important differences with PVS-Cert:

- This prototype is restricted to the extraction of information from the *prover* only. For this reason, the emitted certificates contain no typing information, and hence the language in which they are written does not correspond to some extension of PVS-Cert nor to any other typed language. As a consequence, at this stage, these certificates can only be used to monitor the reasoning steps performed in the *prover*, without fully ensuring the soundness of the results.

- In this prototype, proofs are kept in the formalism of sequent calculus, which is the formalism used internally in PVS. As discussed in Section 2.2.2, the addition of typing information to the formalism of sequent calculus is not as simple as in the case of natural deduction. As a consequence, it could be preferable in practice to translate these certificates into the formalism of natural deduction before adding typing information. Following this strategy, the obtained language of certificates draw near some extension of PVS-Cert.

- This prototype does not include the implementation of a kernel in which the whole PVS language would be encoded. Instead, it is focused on some selection of the most widely used features of PVS, without expressing them in a minimal system. In particular, all logical connectives are kept primitive in this prototype.

Yet, as in PVS-Cert and PVS-Core, this prototype uses a form of conversion. This notion of conversion includes β-equivalence, but also the mechanism of constant definitions suggested in Section 11.2.1 as an extension of PVS-Cert and PVS-Core. These mechanisms turn out to be indispensable to keep certificates sufficiently compact in practice. A shared perspective from Part I and Part II is to turn this prototype into a complete system of certificates expressed in some extension of PVS-Cert. This goal requires several important additional works, including in particular the instrumentation of the PVS *type-checker* to add typing information to certificates.

11.3 Other perspectives

As mentioned in the presentation of related works (Section 1.5), the presented results have been developed in relation with several other works (such as [61], [70], [79], and [53]), and exploring further these relations could also lead to new perspectives.

One of the most interesting extension of this work to complete this analysis of predicate subtyping would be the definition of formal semantics for PVS-Cert and PVS-Core. On the one hand, the standards set theoretical semantics defined for PVS in [61] could be adapted to define standard set theoretical semantics for PVS-Core and possibly PVS-Cert. On the other hand, another interesting approach would be to define *complete*

families of models for PVS-Core and PVS-Cert, in the sense that one proposition in PVS-Core (respectively PVS-Cert) is provable if and only if it is true in all models of these respective families. We conjecture that complete families of set theoretical models can be defined by relaxing the definition of standard set theoretical models to accept alternative models where the interpretation of a function type does not necessarily contain all functions from the interpretation of the domain to the interpretation of the co-domain and the interpretation of the type of propositions does not necessarily contain only two elements.

Another perspective is the extension of the presented conservativity theorem of predicate subtyping over higher-order logic (Theorem 8.5.2) to more complex systems. More precisely, the conservativity of PVS-Cert over λ-HOL implies the conservativity of its subsystem PVS-Cert$^-$ (Definition 4.2.3) over λ-HOL. PVS-Cert$^-$ is a fragment of the system ECC presented in [53], for which the conservativity over higher-order logic remains an open problem. Hence, an interesting extension of the presented work would be to investigate to which extent the proof of conservativity presented for PVS-Cert can be extended to some larger fragment of ECC than simply PVS-Cert$^-$.

Part II

Proof certificates in PVS

Chapter 12

Proof certificates in PVS

Given the complexity of proof assistants such as PVS, external verifications become necessary to reach the highest levels of trust in their results. The simplest way to perform such verifications is to require from these proof systems to export certificates that can be checked using third-party tools. This approach is followed for instance in the OpenTheory project [47], in which the higher order logic theorem provers HOL Light, HOL4, and ProofPower are instrumented to export verifiable certificates in a shared format.

The type system PVS-Cert presented in the first part of this work would be well-suited as a language of verifiable certificates for the system PVS-Core. The precise way to use PVS-Cert judgements as PVS-Core certificates is detailed in Definition 10.1.1. However, as detailed in Chapter 2, PVS is a much more complex system than PVS-Core, as this latter system was designed as a minimal fragment of PVS containing predicate subtyping.

As a consequence, the extension of PVS-Cert to a complete language of certificates for PVS is left as future work. Instead, we present in this second part a first prototype to build externally verifiable certificates from PVS, in which certificates do not contain any typing information. For this reason, it is a much simpler system than PVS-Cert from the point of view of type theory. Although the certificates generated with this prototype cannot be considered as comprehensive proofs because of this lacking information, this mechanism is a first step towards the generation of externally verifiable proofs from PVS. Following the dichotomy between the *type-checker* and the *prover* in PVS presented in Section 1.3, this prototype is suited for the verification of the reasoning steps performed in the *prover* only.

This proof generation mechanism is built by instrumenting the PVS proof system itself. More precisely, the PVS prover is modified to record detailed proofs step by step during the proof search process. In the systems HOL Light, HOL4, and ProofPower, all reasoning steps are built from a small number of simple logical rules, and these logical rules are used in turn as starting points in the generation of OpenTheory certificates.

For this reason, the instrumentation of these systems is economical in the sense that it is sufficient to instrument these logical rules to generate OpenTheory certificates. However, as detailed in the following, PVS is not based on a layer of simple logical rules: in PVS, the primitive reasoning steps may be much more complex. For this reason, the instrumentation of PVS to generate certificates is much less economical, as almost the whole prover would need to be instrumented to generate complete certificates.

At the current stage of this work, the instrumentation of the PVS prover is not complete. Whenever some reasoning step is performed through some part of the prover that has not been instrumented, an unverified assumption is generated to complete the proof, making this certificate generation mechanism usable for any PVS theory. For a restricted fragment of PVS, the proof certificates are exported to the universal proof checker Dedukti [66], and the unverified assumptions are exported and proved externally using the automated theorem prover MetiTarski [1].

The following of this chapter is directly adapted from the work published in [39] presented by the author, with the addition of some comparisons and references to the first part of this work – in particular, to Chapter 2 and Chapter 3.

12.1 Certificates as refinements of the PVS proof traces

This section is an overview of the proving process in PVS. A more general overview of PVS can be found in the PVS documentation [60] as well as in Chapter 2. In particular, a more abstract view of proving in PVS can be found in Section 2.2.2, which is complementary to the more technical description provided in the following of this section.

In PVS [60], the proof process is decomposed into a succession of proof steps. These proof steps are recorded into a proof trace format, the `.prf` files. Proof traces can be used internally to rerun and verify proofs, but cannot be used to check proofs externally without reimplementing PVS proof search mechanisms almost entirely.

The purpose of the certificate language presented in this work is to check PVS results externally using small systems. To this end, we present a decomposition of PVS proof steps into a small number of atomic rules, which are easier to encode into a third-party system than the PVS proof steps themselves. The proof certificates are built on these atomic rules, and can be checked without having to reimplement PVS proof steps.

These atomic rules are defined as a refinements of an intermediate decomposition of proof steps which is already present in PVS. This intermediate decomposition is based on a specific subset of proof steps, the primitive rules. In PVS, every proof step, including defined rules and strategies, can be decomposed as a sequence of primitive rules. As any primitive step is a proof step, this intermediate level of decomposition can be formalized in the original format of `.prf` proof traces. In fact, such a decom-

position can be performed using the PVS package Manip [24], in which the instruction `expand-strategy-steps` allows one to decompose every proof step into a succession of primitive rules.

However, this intermediate decomposition is not sufficient to make proof traces verifiable externally using small systems. Indeed, the complexity of PVS proof mechanisms lies for the largest part in the primitive rules themselves. In particular, the implementation of primitive rules is one order of magnitude larger than the implementation of strategies. For instance, the primitive rule `simplify` hides advanced reasoning techniques including simplifications, rewritings, and Shostak's decision procedures.

In order to export a refinement of the primitive rule decomposition, PVS is modified directly to record reasoning at a higher level of precision. The main part of this modification is done in the source code of the primitive rules themselves. This instrumentation doesn't affect the reasoning in any way besides some slowdown due to the recording of proofs. In particular, it doesn't affect the generation of `.prf` proof traces, which continue to be used internally to rerun proofs as in the original system.

In the next section, we present the formalization of the certificate language chosen for PVS. Then, we present a first attempt to export these proofs to the universal proof checker Dedukti [66], and to export their unverified assumptions to the theorem prover MetiTarski [1].

12.2 Proofs certificates in PVS

The language of proof certificates is presented in three parts. The layer of expressions in the certificate language is presented first, equipped with a notion of conversion. Then, an abstract presentation of the layer of deduction rules is defined above the layer of expressions. Last, a concrete formalization of these abstract rules is presented, defining the language of proof certificates.

12.2.1 Expressions and conversion

Proof are added as a new layer of abstract syntax, on top of the existing layers of PVS expressions and PVS sequents. Besides the precise definition of expressions in the PVS documentation itself [62, 68], an overview of the language of PVS expressions can be found Section 2.1.2 and a brief presentation of PVS sequents is provided in Section 2.2.2.

However, the precise expressions and sequents used in this language are the syntax trees corresponding to their internal representations in PVS. For readability, these trees will be denoted as they are printed in PVS. We stress the fact that this denotation is not faithful: internal representations contain additional information such as types and name resolutions, generated through complex PVS mechanisms, but erased through the

PVS printing.

In the proof format presented in this chapter, PVS expressions are equipped with a new notion of conversion inspired from the work on PTSs with definitions presented in [67]: conversion modulo β-reduction and δ-reduction, this latter kind of reduction corresponding to the unfolding of constant definitions. This conversion relation will be denoted $\equiv_{\beta\delta}$. Using such a notion of conversion, it is not necessary to record the unfolding of a definition or the reduction of a β-redex in proof certificates, making these certificates particularly compact.

Although PVS is not based on reasoning modulo δ-reduction, i.e. modulo the unfolding of constant definitions, its specification admits a system of constant definitions, used here to give a precise meaning to δ-reduction. These definitions, briefly mentioned in the overview of PVS provided in Chapter 2, are referred to as *interpreted constant declarations*. The interpreted constant declaration of a new constant `x` of type `A` defined as the expression `t` is written `x : A = t` in PVS. More details about this constant definition mechanism can be found in the PVS documentation [62]. At this stage, more complex kinds of definitions such as recursive definitions are kept out of the conversion relation: in the current certificate format, the unfolding of recursive definitions are kept as explicit reasoning steps.

Besides the main difference between the current certificate language and the system PVS-Cert defined in Chapter 3, which is the fact that certificates do not include any type information, the presence of δ-reduction in the definition of certificates is a second important difference. In the present context of a practical certificate format for PVS, this feature is key to keep certificates as compact as possible and to scale up the certificate generation mechanism to large and complex PVS proofs.

On the theoretical point of view, the addition of δ-reduction in systems with β-reduction was studied in [67] in the case of PTSs, with the main conclusion that it doesn't affect the most important properties of these systems, such as strong normalization – whenever this property was established for the original system. As discussed in Section 11.2, we leave as a future work the extension of PVS-Cert with this mechanism, which would narrow the gap between the work done in Part I and the present chapter.

12.2.2 Reasoning

In the same way as the internal representation of expressions is used to define the layer of expressions in this certificate language, the internal representation of sequents will be used to define a new layer on top of the layer of expressions. As specified in Section 2.2.2, in PVS, a sequent is recorded internally in a single list containing the succedents and all negations of antecedents. For instance, a sequent appearing as

```
[-1] NOT P1
[-2] P2
 |-------
[1] Q
```

is recorded internally as some permutation of the list `NOT NOT P1, NOT P2, Q`. Denoting Γ the union of this list together with the list of hidden formulas, the corresponding sequent will be denoted $\vdash \Gamma$.

As also specified in Section 2.2.2, the PVS sequents are recorded as lists but manipulated as multisets: the ordering between formulas has no influence on the deduction rules that can be applied to them. Following this idea, we equip the sequents composing the certificate language with the relation of identification modulo permutation. In the same way as the conversion $\equiv_{\beta\delta}$ allows to keep some reasoning steps implicit in proof certificates, the identification of sequents modulo permutation avoids the recording of exchange rules in proof certificates, making them as compact as possible.

The following atomic rules are defined on top of this layer of sequents. All of them are presented modulo the conversion $\equiv_{\beta\delta}$ on expressions and the identification of sequents modulo permutation.

Structural rules

$$\frac{}{\vdash \Gamma, \texttt{P, NOT P}} \qquad \frac{\vdash \Gamma, \texttt{P} \qquad \vdash \Gamma, \texttt{NOT P}}{\vdash \Gamma} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \texttt{P}} \qquad \frac{\vdash \Gamma, \texttt{P, P}}{\vdash \Gamma, \texttt{P}}$$

Propositional rules

$$\frac{}{\vdash \Gamma, \texttt{TRUE}} \qquad \frac{\vdash \Gamma, \texttt{NOT TRUE}}{\vdash \Gamma} \qquad \frac{\vdash \Gamma, \texttt{FALSE}}{\vdash \Gamma} \qquad \frac{}{\vdash \Gamma, \texttt{NOT FALSE}}$$

$$\frac{\vdash \Gamma, \texttt{P} \qquad \vdash \Gamma, \texttt{Q}}{\vdash \Gamma, \texttt{P AND Q}} \qquad \frac{\vdash \Gamma, \texttt{NOT P, NOT Q}}{\vdash \Gamma, \texttt{NOT (P AND Q)}}$$

$$\frac{\vdash \Gamma, \texttt{P, Q}}{\vdash \Gamma, \texttt{P OR Q}} \qquad \frac{\vdash \Gamma, \texttt{NOT P} \qquad \vdash \Gamma, \texttt{NOT Q}}{\vdash \Gamma, \texttt{NOT (P OR Q)}}$$

$$\frac{\vdash \Gamma, \texttt{NOT P, Q}}{\vdash \Gamma, \texttt{P IMPLIES Q}} \qquad \frac{\vdash \Gamma, \texttt{NOT Q} \qquad \vdash \Gamma, \texttt{P}}{\vdash \Gamma, \texttt{NOT (P IMPLIES Q)}} \qquad \frac{\vdash \Gamma, \texttt{P}}{\vdash \Gamma, \texttt{NOT NOT P}}$$

$$\frac{\vdash \Gamma, \texttt{P IMPLIES Q} \qquad \vdash \Gamma, \texttt{Q IMPLIES P}}{\vdash \Gamma, \texttt{P IFF Q}}$$

$$\frac{\vdash \Gamma, \texttt{NOT (P IMPLIES Q), NOT (Q IMPLIES P)}}{\vdash \Gamma, \texttt{NOT (P IFF Q)}}$$

$$\frac{\vdash \Gamma, \texttt{P IMPLIES Q1} \qquad \vdash \Gamma, \texttt{NOT P IMPLIES Q2}}{\vdash \Gamma, \texttt{IF(P, Q1, Q2)}}$$

$$\frac{\vdash \Gamma, \texttt{NOT (P AND Q1)} \qquad \vdash \Gamma, \texttt{NOT (NOT P AND Q2)}}{\vdash \Gamma, \texttt{NOT IF(P, Q1, Q2)}}$$

Quantification rules

$$\frac{\vdash \Gamma, \texttt{P}}{\vdash \Gamma, \texttt{FORALL (x : T) : P}} \qquad \frac{\vdash \Gamma, \texttt{NOT P[t/x]}}{\vdash \Gamma, \texttt{NOT FORALL (x : T) : P}}$$

$$\frac{\vdash \Gamma, \texttt{P[t/x]}}{\vdash \Gamma, \texttt{EXISTS (x : T) : P}} \qquad \frac{\vdash \Gamma, \texttt{NOT P}}{\vdash \Gamma, \texttt{NOT EXISTS (x : T) : P}}$$

Equality rules

$$\frac{}{\vdash \Gamma, \texttt{t = t}} \qquad \frac{\vdash \Gamma, \texttt{t1 = t2} \qquad \vdash \Gamma, \texttt{t2 = t3}}{\vdash \Gamma, \texttt{t1 = t3}}$$

$$\frac{\vdash \Gamma, \texttt{P[t/x]} \qquad \vdash \Gamma, \texttt{t = u}}{\vdash \Gamma, \texttt{P[u/x]}} \qquad \frac{\vdash \Gamma, \texttt{u1 = u2}}{\vdash \Gamma, \texttt{t[u1/x] = t[u2/x]}}$$

$$\frac{\vdash \Gamma, \texttt{NOT P, t1 = t2}}{\vdash \Gamma, \texttt{IF(P, t1, u) = IF(P, t2, u)}} \qquad \frac{\vdash \Gamma, \texttt{P, u1 = u2}}{\vdash \Gamma, \texttt{IF(P, t, u1) = IF(P, t, u2)}}$$

Extensionality rules

$$\frac{\vdash \Gamma, \texttt{P IFF Q}}{\vdash \Gamma, \texttt{P = Q}} \qquad \frac{\vdash \Gamma, \texttt{t = u}}{\vdash \Gamma, \texttt{LAMBDA (x : T) : t = LAMBDA (x : T) : u}}$$

$$\frac{\vdash \Gamma, \texttt{t = u}}{\vdash \Gamma, \texttt{FORALL (x : T) : t = FORALL (x : T) : u}}$$

$$\frac{\vdash \Gamma, \texttt{t = u}}{\vdash \Gamma, \texttt{EXISTS (x : T) : t = EXISTS (x : T) : u}}$$

Extra rules

$$\frac{\vdash \Gamma, \Delta \qquad \vdash \Gamma, \Delta_1 \qquad \cdots \qquad \vdash \Gamma, \Delta_n}{\vdash \Gamma, \Delta}\ \text{TCC}$$

$$\frac{\vdash \Gamma, \Delta_1 \qquad \cdots \qquad \vdash \Gamma, \Delta_n}{\vdash \Gamma, \Delta}\ \textsc{Assumption}$$

Only the two last rules, TCC and ASSUMPTION, are specific to this system.

- The rule TCC is due to the appearance of type-checking conditions during proof

runs, for instance after giving an instantiation for an existential proposition. In the rule TCC, these type-checking conditions correspond to the additional premises $\vdash \Gamma, \Delta_i$, while the continuation of the main proof corresponds to the premise $\vdash \Gamma, \Delta$. As the current format of certificates does not include typing information, this rule could be skipped, replacing any proof tree ending with this rule by its first subtree. However, this rule allows to record in a single proof tree all reasoning steps performed in the PVS prover during proof runs, including the reasoning steps ensuring typing constraints.

- The second rule, ASSUMPTION, is theoretically equivalent to the addition of a trusted assumption, but more practical to handle in the instrumentation of PVS. Such a rule is generated by default whenever some uninstrumented reasoning step is invoked during proof runs. In practice, the use of the rule ASSUMPTION doesn't imply that the corresponding reasoning gap could not be described using the other rules. For instance, the primitive rule *bddsimp*, which calls a function outside the PVS kernel, is not instrumented, and its use triggers the generation of an ASSUMPTION rule. Yet, the corresponding reasoning steps could be justified using structural and propositional rules only. On the other hand, the strategy *prop*, which has the same role, doesn't generate any ASSUMPTION rule, as the underlying primitive rules $flatten$ and *split* are both instrumented.

Contrary to the system PVS-Cert, this system is not designed to be minimal but to be faithful to the way reasoning is performed in PVS, in order to make the instrumentation of PVS as simple and transparent as possible for the generation of certificates during proof runs. As a consequence, contrary to the work presented in Section 2.1.2 defining a minimal set of connectives and quantifiers for PVS-Cert, the different connectives and quantifiers are not factored in any way in this language of certificates. For the same reason, the set of deduction rules is not minimal, some deduction rules being redundant. For instance, the six equality rules could be reduced to two: the reflexivity introduction rule and the elimination rule corresponding Leibniz's characterization of equality. However, keeping all these rules available eases the instrumentation of PVS to generate certificates. In these two cases, some post-processing factorization performed outside the instrumentation of the prover could be defined to yield a more compact certificate language, but, at this stage, the implementation of such a mechanism is left for future work.

12.2.3 Proof objects

In order to record lightweight proofs, we record only the rules used in the proofs, provided with a sufficient amount of rule parameters.

For instance, the proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \texttt{NOT P, NOT Q, P}}}{\vdash \texttt{NOT P, NOT Q, NOT NOT P}}}{\vdash \texttt{NOT (P AND Q), NOT NOT P}}}{\vdash \texttt{(P AND Q) IMPLIES NOT NOT P}}$$

is recorded as follows:

```
RImplies(P AND Q, NOT NOT P,
  RNotAnd(P, Q,
    RNotNot(P,
      RAxiom(P))))
```

where `RImplies`, `RNotAnd`, `RNotNot`, and `RAxiom` denote the rules used in the proof, and accept as argument a list of parameters followed by a (possibly empty) list of sub-proofs.

12.3 Checking PVS proofs using Dedukti and Metitarski

This part of the work is only at the stage of a first prototype. The universal proof checker Dedukti is used to verify the proof certificates. As these certificates contain unverified assumptions, the automated theorem prover MetiTarski is used to prove them externally.

12.3.1 Translating proofs to Dedukti

The proof checker Dedukti was chosen to verify PVS certificates because of its very flexible definition of conversion. In Dedukti, conversion is based on the extension of β-reduction with any set of rewrite rules defined by the user. In particular, δ-reductions can be defined as particular cases of rewriting rules in Dedukti, allowing this proof checker to accept natively proofs based on reasoning modulo $\equiv_{\beta\sigma}$ without having to construct the steps of β-reduction or δ-reduction kept implicit in such proofs. As a consequence, the translation of PVS certificates to Dedukti is kept much more compact than what could be done using a proof checker with no conversion mechanism.

Dedukti is a dependently typed language. However, as certificates contain reasoning steps without any typing information, PVS types are kept out of the translation of PVS certificates to Dedukti. In this purpose, we declare one single Dedukti type `type` for all PVS expressions. In order to translate applications, we introduce a new Dedukti constant `apply` of type `type -> type -> type`. Conversely, in order to translate lambda expressions, we introduce a new Dedukti constant `lambda` of type `(type -> type) -> type`. A similar technique is used to translate the connectives `TRUE`, `FALSE`, `NOT`, `AND`, `OR`, `IMPLIES`, `IFF`, `IF` as well as the quantifiers `FORALL` and `EXISTS` and equality, yielding a translation from the subset of PVS based on these constructions to Dedukti. In the

following, we denote this translation of a PVS expression `t` as `[t]`.

The translation from PVS certificates to Dedukti is a translation from sequent calculus to natural deduction. The use of Dedukti being based on the Curry-Howard isomorphism, a proof of a proposition `P` is expected as a term of type `[P]`. The main translation function takes as inputs a proof of a sequent $\vdash \texttt{P}_1, ..., \texttt{P}_n$ and a list of proof variables $h_1, ..., h_n$, and outputs a Dedukti term p which has the type `[FALSE]` in the context $h_1 : \texttt{[NOT P}_1\texttt{]}, ..., h_n : \texttt{[NOT P}_n\texttt{]}$. It is defined straightforwardly with the introduction of one new Dedukti constant per deduction rule, each constant being declared with an appropriate Dedukti type. For instance, the constant declared for the introduction rule of the constant connective `TRUE` is declared with the type of functions from `[NOT TRUE]` to `[FALSE]`.

Using this main translation function, for any proposition `P` proved in PVS, and for any proof variable h, we build a Dedukti proof p of type `[FALSE]` in the context h : `[NOT P]`. Then, using a rule of negation introduction together with a rule of double negation elimination, we get a Dedukti proof term p' of type `[P]` in the empty context, as expected.

12.3.2 Checking assumptions with MetiTarski

Every rule except ASSUMPTION is valid in classical higher-order logic. In order to check the assumption rules as well, we use an automated theorem prover. As the certificate generation mechanism has been mostly tested with the arithmetic theories (`ints`) of the NASA Library `nasalib`, a large part of the generated assumptions correspond to formulas mixing uninterpreted symbols with arithmetic. For this reason, the first-order theorem prover MetiTarski was chosen to prove these assumptions externally, as it is well adapted for these kinds of problems.

Using conjunctions, disjunctions and implications, every ASSUMPTION rule is translated into a single proposition, which in turn is translated to the TPTP [72] format, which is the input format of MetiTarski. The main issue in this translation is the presence of higher-order expressions, such as λ-expressions or if-then-else expressions for instance. These terms are translated as constant symbols: the obtained expressions are syntactically correct TPTP formulas, and their validity in first-order logic ensures the validity of the corresponding original expression in higher-order logic.

12.4 Results

The instrumentation of PVS to build proof certificates is not restricted to any fragment of PVS, contrary to the translation of these certificates to Dedukti. The generation of certificates has been tested using the arithmetic theories (`ints`) of the NASA Library `nasalib`. The generation of all certificates for the whole (`ints`) library (32 files, 268

proofs) was performed in one hour.

On the other hand, the exportation of certificates and trusted assumptions to Dedukti and MetiTarski respectively has been tested only tested on simpler examples written for this purpose, including the following one:

```
induction :  THEORY
  BEGIN
  f :  [nat -> nat]
  nat_sum :  LEMMA
      (f(0) = 0 AND (FORALL (n:nat):  f(n+1) = f(n) + n + 1))
           IMPLIES FORALL (n:nat):  2 * f(n) = n * (n + 1)
END induction
```

This theorem was proved in two steps: `flatten`, and `induct-and-simplify`. The Dedukti file generated has been successfully checked by Dedukti. It contained 19 unverified assumptions. All of them have been successfully proved using MetiTarski.

12.5 Conclusion

In this part, we presented a first prototype to build externally verifiable certificates from PVS. This prototype is based on the instrumentation of the PVS *prover* to extract reasoning steps at a sufficient level of detail. As it does not include any instrumentation of the PVS *type-checker* yet, the emitted certificates contain no typing information. As a consequence, at this stage, these certificates can only be used to monitor the reasoning steps performed in the *prover*, without fully ensuring the soundness of the claimed theorems.

Although the PVS *prover* itself is not fully instrumented, this prototype is programmed to generate an assumption automatically whenever some uninstrumented part of the prover is used, making the certificate generation mechanism usable for any PVS theory. For a restricted fragment of PVS, the proof certificates are exported to the universal proof checker Dedukti [66], and the unverified assumptions are exported and proved externally using the automated theorem prover MetiTarski [1].

Several further perspectives can be studied from this work. First, one interesting direction would be to enrich the instrumentation of the PVS prover as well as the translations to Dedukti and Metitarski in order to extend to more and more theories the result presented in Section 12.4, in which a Dedukti certificate is generated and all generated assumptions are proved by MetiTarski. The arithmetic theories (`ints`) of the NASA Library `nasalib` would be interesting candidates to be used in this purpose.

Another interesting perspective from this work, which is shared with the work pre-

sented in Part I, is to turn this prototype into a system of certificates expressed in some well-suited extension of the system PVS-Cert, as discussed in Section 11.2. As developed in Section 11.2, this goal requires several important additional works, including the instrumentation of the PVS *type-checker* to add typing information to certificates.

Part III

Expressing classical first-order logic in constructive systems

The final part of this work is dedicated to the investigation of the expression of classical proofs in constructive systems. This question is interesting in the perspective of expressing the results of classical theorem provers in a constructive system without having to extend this latter system with the law of excluded middle or any other classical axiom. This situation can be instantiated with many concrete systems as most first-order theorem provers are classical systems and several proof assistants as well as several systems based on type theory – with the notable exception of PVS – are constructive systems. In this work, all experiments have been performed using the first-order classical theorem prover Zenon [12] and the constructive proof checker Dedukti [11].

The oldest and most intensively studied translations from classical systems to constructive ones are the double-negation translations, such as Kolmogorov's [49], Gödel-Gentzen's [33, 43] and Kuroda's [51]. However, the emergence of classical theorem provers and the even more recent emergence of theorem provers which are able to generate proof trees sheds a new light on this topic.

On the one hand, the size of automatically generated proofs increases with the improvement of proof search capabilities, and, as a consequence, the question of the impact of such translations on the size of proofs becomes significant. The first chapter of this part (Chapter 13) is dedicated to this question. In this chapter, a lightweight double-negation translation is defined from classical logic to constructive logic, limiting the growth of formulas through translations. This translation is proved minimal among a large class of double-negation translations.

On the other hand, the large libraries of classical proofs generated from these new proof systems become an opportunity for designing new approaches to tackle the problem of constructivization, i.e. the problem of finding whether some classical theorem also holds constructively. The last chapter of this part (Chapter 14) is dedicated to this question. In comparison with Chapter 13, the translation is not required to be total anymore, but the translation of a classical proof is required to be a constructive proof *of the same theorem*. A new constructivization algorithm is presented in this chapter. Besides the presentation of the statistics of success corresponding to this approach, this new algorithm is also studied on a theoretical standpoint, yielding three large fragments of first-order logic statements for which this algorithm is provably complete – and for which, as a consequence, classical provability matches constructive provability.

Chapter 13 and Chapter 14 are independent from the rest of this work presented in Part I and Part II and, besides this shared motivation of investigating the different expressions of classical proofs in constructive systems, the respective contents of these two chapter are independent from each other. As a consequence, these two chapters are rendered as in the publications [37] and [38] in which they have been respectively presented by the author, at the exception of Section 14.8 in the last chapter.

Chapter 13

A lightweight double-negation translation

Deciding whether a classical theorem can be proved constructively is a well-known undecidable problem. As a consequence, any computable double-negation translation inserts some unnecessary double negations. It is shown in this chapter that most of these unnecessary insertions can be avoided without any use of constructive proof search techniques. For this purpose, we restrict the analysis to syntax-directed double-negation translations, which translate a proposition through a single traversal – and include most of the usual translations such as Kolmogorov's [49], Gödel-Gentzen's [33, 43], and Kuroda's [51]. A partial order among translations are presented to select translations avoiding as many double negations as possible. This order admits a unique minimal syntax-directed translation with noticeable properties.

13.1 Introduction

The status of the law of excluded middle in a proof system has significant implications: classical systems admit faster proof search algorithms, while constructive systems have more straightforward proofs-as-programs interpretations. As more and more systems implementing classical and constructive logic are developed, the question of translating theorems or proofs from one to the other becomes relevant. Translating constructive logic into classical logic is easy, as constructive theorems and proofs are a fortiori classical ones, but all distinctive properties of constructive proofs, such as the witness property, are lost in such translations. Thus, it seems more interesting to go in the other direction. However, as not all classical theorems are provable constructively – this question is even undecidable – classical logic can be embedded only through a correct translation, i.e. a function mapping classical theorems to constructive ones. In first-order logic, a usual way to translate a classical theorem is to insert double negations in it. Based on this idea, many correct double-negation translations have been developed; Kolmogorov's [49], Gödel-Gentzen's [33, 43], and Kuroda's [51] can be mentioned among them – the associated proofs translations are thoroughly studied in [57].

In order to push interoperability between classical and constructive systems as far as possible, we want to transpose from classical to constructive logic not only absolute provability but also provability relative to a given context. For this purpose, a translation f will be given by a pair of functions (f^+, f^-), with f^+ mapping propositions to propositions and f^- mapping contexts to contexts, defining $f(\Gamma \vdash C)$ as $f^-(\Gamma) \vdash f^+(C)$. This definition based on sequents instead of propositions is more expressive at the cost of some extra restrictions. Using a definition restricted to propositions, i.e. to absolute provability, all unprovable propositions are considered the same: for instance, any atomic proposition can be mapped to $\bot$. Using this definition based on sequents, such translations are not allowed anymore. Many double-negation translations presented as translations of propositions can fit the sequents-based definition, using the same function to translate propositions and contexts – Kolmogorov's, Gödel-Gentzen's, and Kuroda's can be mentioned among them. Other translations as in [27] are designed to transpose absolute provability only and cannot be transposed to correct sequent-based translations in such a way.

The intention in this work is to present sequent-based translations that are not only correct but also as faithful as possible: indeed, the most convenient way to reuse a classical proof in a constructive context is to translate it to a constructive judgment chosen as compliant as possible to the original one. For instance, a translation mapping all provable propositions to $\top$ wouldn't be very useful when trying to reuse classical theorems or proofs in a constructive context. As double-negation translations alter sequents in a more limited way, they are better suited to fit these restrictions. For this reason, we choose to focus in this work on double-negations translations only.

Given a sequent S, two characteristics of S will be used as a bases for compliance comparisons: its syntax, and the set of judgments that derive from it constructively. Given a double-negation translation S' of S, the first standpoint suggests that S' should contain as few inserted double negations as possible, while the second suggests that as many sequents as possible should be either derivable from both S and S' or from none of them. In the following, the effect of a double negation translation from the first standpoint will be denoted as the syntax alteration, while its effect from the second standpoint will be denoted as the strength alteration. For instance, among all translations of $\vdash \neg\neg P \Rightarrow P$, the syntax alteration is better in $\vdash \neg\neg P \Rightarrow \neg\neg P$ than in $\vdash \neg\neg(\neg\neg P \Rightarrow \neg\neg P)$, but, as both translations are constructively equivalent, the strength alteration is the same.

These compliance requirements are not sufficient to ensure a genuine interoperability between classical and constructive logic. In particular, we do not consider as a tool of interoperability an algorithm from classical proofs to constructive ones that discard its argument and builds its image from scratch using constructive proof search algorithms – even if this kind of translation would be potentially good with respect to compliance requirements. For this reason, we will select translations avoiding any use of constructive

proof search techniques.

The purpose of this work is to determine to what extent the images of double-negation translations can be faithful to the original ones without using constructive theorem proving techniques. First, a set of double-negation translations fitting this restriction is presented. It is denoted as the set of syntax-directed translations, and contains several usual translations such as Kolmogorov's, Gödel-Gentzen's, and Kuroda's. Then, we will show that translations avoiding as many double-negations as possible can be considered to alter less the strength of sequents as well as their syntax. This observation allows to define two partial orders of double negation translations, the second being an extension of the first. The extended partial order leads to a unique minimal translation among correct syntax-directed translations, which avoids many unnecessary double negation insertions compared to the usual ones.

13.2 Syntax-directed double-negation translations

This work is about classical and constructive first-order predicate logic, presented in the sequent calculi LK and LJ. The connectives and quantifier are the usual: $\wedge$, $\vee$, $\Rightarrow$, $\neg$, $\forall$, $\exists$, $\bot$, $\top$. In the current context, it is important to refrain from applying classical equivalences to eliminate one of them, for instance defining $A \Rightarrow B$ as $B \vee \neg A$: keeping the distinction between both is crucial to be able to translate them in different ways in constructive logic.

The purpose of this section is to discard double-negation translations that might be based on constructive theorem proving techniques: as mentioned earlier, the benefit of translating classical proofs to constructive ones would be lost if constructive proofs were built from constructive theorem proving techniques, without using the data of the original classical proofs. We want to exclude translations such as, for instance, uncomputable ones mapping any constructively provable sequent to itself.

In sequent calculus without cuts, the subformula property holds: the proof of a sequent is built from proofs of sequents involving subformulas of the original sequent exclusively. On the one hand, the leafs of the proof are built from the identification of disjoint subformulas. On the other hand, the inner nodes are built recursively, from deep to shallow subformulas. Therefore, an effective way to avoid any constructive theorem proving computation is to decide the number of double negation inserted in front of a given subformula regardless of the content of any disjoint subformulas on the one hand, and regardless of any strictly included subformulas on the other hand. Furthermore, following the same idea, the translation of a proposition in a context Γ must be done regardless of the presence of other propositions in Γ: $f^-(C_1, ..., C_n) = f^-(C_1), ..., f^-(C_n)$.

In a given sequent, the remaining information available to decide the number of double negations inserted at a given occurrence is contained in the path going from the root

of the sequent to this occurrence. Therefore, such translations can be computed in a single traversal of the sequent beginning at its root – a necessary and sufficient condition is that any node is discovered before its children. For this reason, these translations will be denoted as syntax-directed translations.

The path going from the root of a sequent to one of its occurrences contains the following information: the place of the proposition containing the occurrence in the sequent (antecedent or succedent), the labels of the nodes going from the root of this proposition to the occurrence (connectives, quantifiers, or predicate symbols), and, for every binary connective crossed above the occurrence, which direct child contains the occurrence (left one or right one). We formalize this notion of path in the following way, using the usual notations of regular expressions:

Definition 13.2.1.
- *The place of a proposition in a sequent is noted* $(-)$ *for antecedents and* $(+)$ *for succedents.*
- *The set of predicate symbols is noted* S*. We define the set of labels* L *as the following:* $(\wedge \mid \vee \mid \Rightarrow \mid \neg \mid \forall \mid \exists \mid \bot \mid \top \mid S)$*.*
- *The indication of a direct child of a proposition beginning with a binary connective is noted* L *for the left one and* R *for the right one. We define the language* E *of directed edges:* $(\wedge\mathrm{L} \mid \wedge\mathrm{R} \mid \vee\mathrm{L} \mid \vee\mathrm{R} \mid \Rightarrow \mathrm{L} \mid \Rightarrow \mathrm{R} \mid \neg \mid \forall \mid \exists)$*.*
- *The language of* P *paths is the following:* $(- \mid +)E^*L$

For instance, the path corresponding to an atomic proposition P in the sequent $A \wedge (\neg P \vee C) \vdash D$ is $(- \wedge \mathrm{R} \vee \mathrm{L} \neg P)$.

A syntax-directed translation inserts double-negations at a given occurrence according to the path leading to it. Therefore, it can be given by a multiset of paths: for each path, the number of double negations inserted at the occurrence corresponding to this path is its multiplicity in the multiset. Formally, the link between a multiset of paths and its corresponding syntax-directed translation is the following:

Definition 13.2.2. *Let* X *be a multiset of paths. Let* $w \in (- \mid +)E^*$*. Let* A *be a proposition. We define the proposition* $F_X(w, A)$ *recursively as follows.*

Let l_A *be the label at the root of* A*. Let* $n(w, A, X)$ *be the multiplicity of* wl_A *in* X*.*

- *If* $l_A \in (\bot \mid \top \mid S)$*,* $F_X(w, A) = (\neg\neg)^{n(w,A,X)}A$*.*
- *If* $l_A \in (\neg \mid \forall \mid \exists)$*,* $F_X(w, A) = (\neg\neg)^{n(w,A,X)}l_A(F_X(wl_A, A_{\mathrm{C}}))$*, where* $A = l_A(A_{\mathrm{C}})$*.*
- *If* $l_A \in (\wedge \mid \vee \mid \Rightarrow)$*,* $F_X(w, A) = (\neg\neg)^{n(w,A,X)}l_A(F_X(wl_A\mathrm{L}, A_{\mathrm{L}}), F_X(wl_A\mathrm{R}, A_{\mathrm{R}}))$*, where* $A = l_A(A_{\mathrm{L}}, A_{\mathrm{R}})$*.*

For any multiset of paths X, we define f^-_X, f^+_X, and f_X as follows:

- $f^+_X(A) = F_X(+, A)$ *and* $f^-_X(A) = F_X(-, A)$
- $f^-_X(A_1, ..., A_n) = f^-_X(A_1), ..., f^-_X(A_n)$
- $f(\Gamma \vdash C) = f^-(\Gamma) \vdash f^+(C)$

f_X is referred as the syntax-directed translation associated with X.

Example 13.2.1. *Kolmogorov's, Gödel-Gentzen's, and Kuroda's double negation translations are all syntax-directed translations.*

- *Kolmogorov double-negation translation consists in the insertion of double negations at all occurrences: the corresponding multiset is the set of all paths $X_{Ko} = P$.*
- *Gödel-Gentzen double-negation translation consists in the insertion of double negations in front of all occurrences of labels among $(\vee \mid \exists \mid S)$: the corresponding multiset is the set of paths $X_{GG} = (- \mid +)E^*(\vee \mid \exists \mid S)$.*
- *Kuroda double-negation translation consists the insertion of in double negation at the root and after all occurrences of labels $\forall$: the corresponding multiset is the set of paths $X_{Ku} = (- \mid +)(E^*\forall)^?L$.*

13.3 Partial orders among double-negation translations

The main issue in the comparison of double-negations translations is the fact that the insertion of double negation does not always alter the strength of judgments in the same way. Some of them weaken judgments, while other strengthen them, and the insertion of a given double negation might partially compensate the insertion of an other. A simple example of this phenomenon can be found using the judgment $\vdash (\forall x P(x)) \Rightarrow Q$. This judgment can be translated both to $\vdash (\forall x P(x)) \Rightarrow \neg\neg Q$ – this can be done using the translation presented in the following – and to $\vdash (\forall x \neg\neg P(x)) \Rightarrow \neg\neg Q$, which is stronger. The double negation on the right of the implication weakens the statement, while that on the left strengthens it. Indeed:

- $(\forall x P(x)) \Rightarrow \neg\neg Q \nvdash (\forall x P(x)) \Rightarrow Q$
- $(\forall x P(x)) \Rightarrow \neg\neg Q \vdash (\forall x P(x)) \Rightarrow \neg\neg Q$
- $(\forall x P(x)) \Rightarrow \neg\neg Q \nvdash (\forall x \neg\neg P(x)) \Rightarrow \neg\neg Q$

This shows that the second translation, which adds more double negations, allows nevertheless to compensate part of the weakening effect of the first translation.

This example shows that one crucial question when inserting a double negation at a given occurrence is to predict if this double negation weakens or strengthens the original

judgment. Fortunately, it is possible to split the set of occurrences and the set of paths into the set of positive ones, at which the insertion of a double negation weakens the sequent, and the set of negative ones, at which the insertion of a double negation strengthens the sequent. This notion of polarity can be formalized in the language of paths as follows:

Definition 13.3.1. *The set of directed edges E can be split into the set of negative ones E_- and the set of positive ones E_+:*
$E_- = (\Rightarrow \text{L} \mid \neg)$, *and* $E_+ = (\wedge\text{L} \mid \wedge\text{R} \mid \vee\text{L} \mid \vee\text{R} \mid \Rightarrow \text{R} \mid \forall \mid \exists)$
Considering furthermore the antecedent position $(-)$ as negative and the succedent position $(+)$ as positive, the polarity of an occurrence is given by its path:

- *The set of positive paths is* $P_+ = (-E_+^*E_- \mid +)(E_+^*E_-)^{2*}E_+^*L$.
- *The set of negative paths is* $P_- = (- \mid +E_+^*E_-)(E_+^*E_-)^{2*}E_+^*L$

All double negations at negative occurrences, which strengthen sequents, have no use concerning the correctness of translations: if the image of a sequent by a given translation has a constructive proof, then, a fortiori, the sequent obtained from the same translation limited to positive double negations has also a constructive proof.

Therefore, it is possible to limit translations to positive occurrences, leaving the choice of negative ones to the user: from a classical proof of $\Gamma \vdash (\forall x P(x)) \Rightarrow Q$, the user might choose between computing the first translation, or, if a stronger conclusion is needed, to compute first a classical proof of $\Gamma \vdash (\forall x \neg\neg P(x)) \Rightarrow Q$ from the original one before translating it, which leads to the second translation.

As a consequence, only the double negations at positive occurrences should be taken into consideration when comparing the strength alteration of different translations, and furthermore, translations inserting less double negations at negative occurrences can be considered better. This leads to consider that both the syntax and the strength of a sequent S are less altered by a translation $f(S)$ than by a translation $g(S)$ if the set of occurrences of S where f inserts double negations is included in the set of occurrences in sequents where g does, which leads to the following partial order:

Definition 13.3.2. *The relation $\leq$ is defined as follow: for two double-negation translations f and g, $f \leq g$ if for any sequent S, f only inserts double negations in S where g does. Let f_X and $f_{X'}$ be two syntax-directed translations. $f \leq g$ if and only if $X \subseteq X'$.*

This partial order being based on an inclusion relation, many translations are incomparable. However, at the syntax level, there is no universal way to compare double negations inserted at different positions. For instance, if a proposition $A \wedge B$ can be translated to $\neg\neg A \wedge \neg\neg B$ or to $\neg\neg(A \wedge B)$ one user could prefer the first one, which translated a conjunction to a conjunction, while another could prefer the second one, which inserts less double negations.

We suggest conversely to consider such choices as equivalent when the strength of the translation stays unchanged. Double negation commutes with connectives and quantifiers in the following cases:

Proposition 13.3.1.
- $\neg\neg(A \wedge B)$ *is constructively equivalent to* $\neg\neg A \wedge \neg\neg B$
- $\neg\neg(A \Rightarrow B)$ *is constructively equivalent to* $A \Rightarrow \neg\neg B$
- $\neg\neg(\neg A)$ *is constructively equivalent – and identical – to* $\neg(\neg\neg A)$

The proof is based on short and straightforward constructive proofs. Proposition 13.3.1 leads to the definition of the following elementary transformations, which map correct translations to correct translations:

- If at least one double negation is inserted at the head of a subformula $A \wedge B$, one of them is replaced by an extra double negations at the head of A and an extra double negation at the head of B.
- If at least one double negation is inserted at the head of a subformula $A \Rightarrow B$, one of them is replaced by an extra double negation at the head of B.
- If at least one double negation is inserted at the head of a subformula $\neg A$, one of them is replaced by an extra double negation at the head of A.

We use the notation $f^{\downarrow}$ to refer to the normal form of a double-negation translation f under these transformations. From this definition, an extended partial order $\leq^{\downarrow}$ between double-negation translations can be defined as follow:

Definition 13.3.3. *The relation* $\leq^{\downarrow}$ *is defined as follow: for two double-negation translations* f *and* g, $f \leq^{\downarrow} g$ *if* $f^{\downarrow} < g^{\downarrow}$ *or if* $f = g$.

It is straightforward to see that this definition corresponds also to a partial order – it is reflexive, antisymmetric and transitive. But one can also notice that if $f \leq g$ implies $f^{\downarrow} \leq g^{\downarrow}$: therefore, $\leq^{\downarrow}$ is an extension of $\leq$.

This extended partial order can be compared with the translations simplifications presented in [29] – which is defined in a broader context, including translations that are not strictly based on the insertion of double-negations, such as [50] – allowing to show minimal properties of some translations such as Kuroda's for instance. The extended partial order $\leq^{\downarrow}$ leads to even lighter translations. In particular, it leads to a unique minimal syntax-directed translation, which is presented in the following section.

13.4 The minimal translation

The idea leading to the minimal syntax-directed translation is to mix the respective advantages of Kuroda's and Gödel-Gentzen's translations, limiting them to positive occurrences only.

As in Kuroda's translation, we begin restricting double negations to the following set:

$$(- \mid +)(E^*\forall)^? L$$

Restricting it to double negations at positive occurrences, this leads to:

$$+L \mid (-E_+^* E_- \mid +)(E_+^* E_-)^{2*} E_+^* \forall L$$

Finally, inspired from Gödel-Gentzen's translation, we suggest to push double negations into subformulas until they reach a label among the set $G = (\vee \mid \exists \mid S)$. To be more precise, double negations can be pushed to both children for the $\wedge$ connective, and only to the right child for $\Rightarrow$; they can be pushed through $\forall$ quantifiers; they can be eliminated when they reach $\bot$, $\neg$, or $\top$ connectives. Using the set $H = (\wedge\text{L} \mid \wedge\text{R} \mid \Rightarrow \text{R})$ of directed edges, this leads to the following set:

$$M = +H^*G \mid (-E_+^* E_- \mid +)(E_+^* E_-)^{2*} E_+^* \forall H^* G$$

It will be shown in the following sections that this set defines a correct syntax-directed translation, which can be proved minimal.

One can notice that the only information used from the path to decide if a double negation must be inserted is the polarity of the occurrence and, in the positive case, the presence of a root or a quantifier $\forall$ separated from the end only by directed edges of H. If we split the set of words $w \in (+ \mid -)E^*$ according to the parity of negative elements in it, and, in the case of an even number, is in one of these three cases, according to the presence of a root or a quantifier $\forall$ separated from the end only by directed edges of H, the three subsets of $(+ \mid -)E^*$ can be easily defined by mutual recursion. Following the same idea, the function f_M can be defined by mutually recursion, using the following functions φ, χ and ψ:

$$\begin{aligned}
\varphi(A \wedge B) &= \varphi(A) \wedge \varphi(B) \\
\varphi(A \vee B) &= \neg\neg(\psi(A) \vee \psi(B)) \\
\varphi(A \Rightarrow B) &= \chi(A) \Rightarrow \varphi(B) \\
\varphi(\neg A) &= \neg\chi(A) \\
\varphi(\forall x A) &= \forall x \varphi(A) \\
\varphi(\exists x A) &= \neg\neg\exists x \psi(A) \\
\varphi(\bot) &= \bot \\
\varphi(\top) &= \top \\
\varphi(P) &= \neg\neg P, P atomic
\end{aligned}
\qquad
\begin{aligned}
\chi(A \wedge B) &= \chi(A) \wedge \chi(B) \\
\chi(A \vee B) &= \chi(A) \vee \chi(B) \\
\chi(A \Rightarrow B) &= \psi(A) \Rightarrow \chi(B) \\
\chi(\neg A) &= \neg\psi(A) \\
\chi(\forall x A) &= \forall x \chi(A) \\
\chi(\exists x A) &= \exists x \chi(A) \\
\chi(\bot) &= \bot \\
\chi(\top) &= \top \\
\chi(P) &= P, P atomic
\end{aligned}
\qquad
\begin{aligned}
\psi(A \wedge B) &= \psi(A) \wedge \psi(B) \\
\psi(A \vee B) &= \psi(A) \vee \psi(B) \\
\psi(A \Rightarrow B) &= \chi(A) \Rightarrow \psi(B) \\
\psi(\neg A) &= \neg\chi(A) \\
\psi(\forall x A) &= \forall x \varphi(A) \\
\psi(\exists x A) &= \exists x \psi(A) \\
\psi(\bot) &= \bot \\
\psi(\top) &= \top \\
\psi(P) &= P, P atomic
\end{aligned}$$

The relation between f_M and these three functions is simply the following:

Proposition 13.4.1. *f_M is given by $(f_M^-, f_M^+) = (\chi, \varphi)$.*

Proof. This proposition follows from this stronger proposition, which can be proved directly by induction: let $w \in (- \mid +)E^*$, let A a proposition,

- $F_X(w, A) = \varphi(A)$ if $w \in +H^* \mid (-E_+^* E_- \mid +)(E_+^* E_-)^{2*} E_+^* \forall H^*$,
  i.e. w if is the longest strict prefix of some path $p \in M$

- $F_X(w, A) = \chi(A)$ if $w \in (-E_+^* E_- \mid +)(E_+^* E_-)^{2*} E_+^*$,
  i.e. w if is the longest strict prefix of some path $p \in P_-$

- $F_X(w, A) = \psi(A)$ else,
  i.e. w if is the longest strict prefix of some path $p \in P_+ \backslash M$.

Then, for any proposition A, $f_M^-(A) = F_M(-, A) = \chi(A)$ and $f_M^+(A) = F_M(+, A) = \varphi(A)$. □

This translation is light compared to Gödel-Gentzen's and Kuroda's. We will give two examples to illustrate the differences between these translation.

First, using the classical theorem $\vdash \forall z((\exists x P(x, z)) \vee \forall x \neg P(x, z))$:

- Gödel-Gentzen: $\vdash \forall z \neg\neg((\neg\neg\exists x \neg\neg P(x, z)) \vee \forall x \neg\neg\neg P(x, z))$

- Kuroda: $\vdash \neg\neg\forall z \neg\neg((\exists x P(x, z)) \vee \forall x \neg\neg\neg P(x, z))$

- f_M: $\vdash \forall z \neg\neg((\exists x P(x, z)) \vee \forall x \neg P(x, z))$

In this case, no unnecessary double negation was inserted by f_M.

Second, using the constructive theorem $\vdash \forall z \neg\neg((\exists x P(x, z)) \vee \forall x \neg P(x, z))$:

- Gödel-Gentzen: $\vdash \forall z \neg\neg\neg\neg((\neg\neg\exists x \neg\neg P(x, z)) \vee \forall x \neg\neg\neg P(x, z))$

- Kuroda: $\vdash \neg\neg\forall z \neg\neg\neg\neg((\exists x P(x, z)) \vee \forall x \neg\neg\neg P(x, z))$

- f_M: $\vdash \forall z \neg\neg((\exists x P(x, z)) \vee \forall x \neg P(x, z))$

In this case, no double negation at all was inserted by f_M.

The latter example is also an illustration of the fact that f_M is a projection: the set of sequents invariant by f_M is exactly its image. The images of contexts is even lighter than the images of succedents: several finite theories as Presburger arithmetic or Robinson arithmetic are translated by themselves using this translation.

In the following section, we show that f_M is correct.

13.5 Correctness of the minimal translation

In this section, for any function or connective f mapping propositions to propositions and for any multiset of propositions $\Gamma = A_1, ..., A_n$, we will use the notation $f(\Gamma) = f(A_1), ..., f(A_n)$.

Theorem 13.5.1. *$f_M = (\chi, \varphi)$ is a translation from classical to constructive logic: for any proposition C and context Γ,*

- *$\Gamma \vdash C$ admits a classical proof if and only if $\chi(\Gamma) \vdash \varphi(C)$ admits a constructive one*
- *$\Gamma \vdash$ admits a classical proof if and only if $\chi(\Gamma) \vdash$ admits a constructive one.*

Furthermore, if the original sequents admit classical proofs, the constructive proofs of the images can be computed from the original ones.

The proof is based on three lemmas.

Lemma 13.5.1. *From any double-negation translation inserting double negations only at positive occurrences, one can compute a proof of the image of a sequent S from any proof of S.*

The idea of the proof is the following: in the original proof, positive subformulas of the original sequent cannot occur at the root of an antecedent – they appear only inside propositions or at the root of succedents. Therefore, in order to build a proof of the image of the sequent, one can insert the rules $\neg_{\mathrm{R}}$ and $\neg_{\mathrm{L}}$ in the original proof to adapt the proof to the additional double negations as soon as they appear at the root of succedents.

Lemma 13.5.2. *For any context Γ and for any proposition C, if $\Gamma, \neg\varphi(C) \vdash$ admits a constructive proof, then $\Gamma \vdash \varphi(C)$ admits a constructive proof that can be computed from the original one.*

Proof. This is done by structural induction on C. Let Π be the proof of $\Gamma, \neg\varphi(C) \vdash$:

- The cases of $\bot$ and $\top$ are straightforward.
- If C is an atom, $\varphi(C)$ is $\neg\neg C$. Then the proof can be obtained eliminating cuts in

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg C \vdash \neg C}\,\text{axiom}}{\neg C, \neg\neg C \vdash}\,\neg_{\mathrm{R}}}{\neg C \vdash \neg\neg\neg C}\,\neg_{\mathrm{R}} \qquad \dfrac{\Pi}{\Gamma, \neg\neg\neg C \vdash}}{\Gamma, \neg C \vdash}\,\text{cut}}{\Gamma \vdash \neg\neg C}\,\neg_{\mathrm{R}}$$

- If C begins with the $\neg$ connective, the $\vee$ connective, or the $\exists$ quantifier, $\varphi(C)$ begins with a negation, which allows to use the same technique as the atomic case.

- If $C = A \Rightarrow B$, we consider the proof Π' obtained eliminating cuts in:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{axiom}\ \dfrac{}{\chi(A) \vdash \chi(A)} \quad \text{axiom}\ \dfrac{}{\chi(A)\varphi(B) \vdash \varphi(B)}}{\Rightarrow_{\text{L}}\ \chi(A), \chi(A) \Rightarrow \varphi(B) \vdash \varphi(B)}}{\neg_{\text{L}}\ \chi(A), \neg\varphi(B), \chi(A) \Rightarrow \varphi(B) \vdash}}{\neg_{\text{R}}\ \chi(A), \neg\varphi(B) \vdash \neg\varphi(A \Rightarrow B)} \qquad \dfrac{\Pi}{\Gamma, \neg\varphi(A \Rightarrow B) \vdash}}{\text{cut}\ \Gamma, \chi(A), \neg\varphi(B) \vdash}$$

  We can apply the induction hypothesis to Π' to get a constructive proof Π'' of $\Gamma, \chi(A) \vdash \varphi(B)$. Then, we can build the proof

$$\dfrac{\dfrac{\Pi''}{\Gamma, \chi(A) \vdash \varphi(B)}}{\Rightarrow_{\text{R}}\ \Gamma \vdash \varphi(A \Rightarrow B)}$$

- Else, C begins with the $\wedge$ connective or the $\forall$ quantifier, and techniques similar to the case of the $\Rightarrow$ connective can be applied, introducing and eliminating cuts to get proofs for which the induction hypothesis can be applied. In the case of the $\wedge$ connective, proofs corresponding to both children will be needed.

□

Lemma 13.5.3. *For all multisets of propositions Γ and Δ, if $\Gamma \vdash \Delta$ admits a classical proof, then $\chi(\Gamma), \neg\psi(\Delta) \vdash$ admits a constructive proof, which can be computed from the original one.*

Proof. This is done by structural induction on the classical proof of $\Gamma \vdash \Delta$. In the following, we will restrict to a representative subset of cases. For readability, we only show in sequents the propositions that are active in the last rule of the classical proof – the general case follows adding $\chi(\Gamma'), \neg\psi(\Delta')$ to all contexts, Γ' and Δ' being the multiset of inactive propositions one the left-hand side and on the right-hand side respectively.

- Weakening, contractions, $\bot_{\text{L}}$ and $\top_{\text{R}}$ are translated in a straightforward way.
- The axiom rule:

$$\text{axiom}\ \dfrac{}{A \vdash A}$$

  Using Lemma 13.5.1, the axiom proof of $A \vdash A$ can be weakened to a proof Π of $\chi(A) \vdash \psi(A)$, which allows to build

$$\dfrac{\dfrac{\Pi}{\chi(A) \vdash \psi(A)}}{\neg_{\text{L}}\ \chi(A), \neg\psi(A) \vdash}$$

- The $\wedge_{\mathrm{L}}$ rule:

$$\wedge_{\mathrm{L}} \frac{A, B \vdash}{A \wedge B \vdash}$$

  Getting a proof Π of $\chi(A), \chi(B) \vdash$ from induction hypothesis, and using that $\chi(A \wedge B) = \chi(A) \wedge \chi(B)$, we get the proof

$$\wedge_{\mathrm{L}} \frac{\dfrac{\Pi}{\chi(A), \chi(B) \vdash}}{\chi(A \wedge B) \vdash}$$

- The other left rules – $\vee_{\mathrm{L}}$, $\Rightarrow_{\mathrm{L}}$, $\exists_{\mathrm{L}}$, $\neg_{\mathrm{L}}$, and $\forall_{\mathrm{L}}$ –, are translated exactly in the same way as $\wedge_{\mathrm{L}}$.

- The $\wedge_{\mathrm{R}}$ rule:

$$\wedge_{\mathrm{R}} \frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$$

  Getting by induction hypothesis a proof Π_A of $\neg\psi(A) \vdash$ and a proof Π_B of $\neg\psi(B) \vdash$, we get the result by cut elimination of the following proof:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\text{axiom}\ \dfrac{}{\psi(A), \psi(B) \vdash \psi(A)} \quad \text{axiom}\ \dfrac{}{\psi(A), \psi(B) \vdash \psi(B)}}{\psi(A), \psi(B) \vdash \psi(A) \wedge \psi(B)}\,\wedge_{\mathrm{R}}}{\neg(\psi(A) \wedge \psi(B)), \psi(A), \psi(B) \vdash}\,\neg_{\mathrm{L}} \qquad \dfrac{\Pi_B}{\neg\psi(B) \vdash}}{\neg(\psi(A) \wedge \psi(B)), \psi(A) \vdash \neg\psi(B)}\,\neg_{\mathrm{R}} \quad \dfrac{\Pi_A}{\neg\psi(A) \vdash}}{\neg(\psi(A) \wedge \psi(B)) \vdash \neg\psi(A)}\,\text{cut}}{\neg(\psi(A \wedge B)) \vdash}\,\text{cut}$$

- All other right rules except $\forall_{\mathrm{R}}$ – i.e. $\vee_{\mathrm{R}}$, $\Rightarrow_{\mathrm{R}}$, $\exists_{\mathrm{R}}$, and $\neg_{\mathrm{R}}$, are translated in the same way as $\wedge_{\mathrm{R}}$: inserting the induction hypotheses with cuts, what remains is proved straightforwardly, and cuts can be eliminated afterwards.

- The $\forall_{\mathrm{R}}$ rule, introducing a fresh variable x:

$$\forall_{\mathrm{R}} \frac{\vdash A(x)}{\vdash \forall x A(x)}$$

  We look for a proof of $\neg\psi(\forall x A(x)) \vdash$, i.e. $\neg\forall x \varphi(A(x)) \vdash$. We get by induction a proof of $\neg\psi(A(x)) \vdash$ that can be weakened by Lemma 13.5.1 to a proof Π of $\neg\varphi(A(x)) \vdash$.

  From Π, we could try the technique used for the other right rules, but the proof will be stuck at the remaining sequent $\neg\forall x \varphi(A(x)) \vdash \neg\varphi(A(x))$, which is not provable

– even classically. The main reason of this failure is that the technique we apply for right rules, through cut eliminations, consists in postponing the application of the rule to a deeper place in the proof. However, $\exists_{\mathrm{L}}$ and $\forall_{\mathrm{R}}$ cannot be postponed in general: they introduce fresh variables that might be necessary for the application of deeper rules – $\exists_{\mathrm{R}}$ and $\forall_{\mathrm{L}}$ in particular.

Therefore, as done in the case of left rules, the $\forall_{\mathrm{R}}$ has to be applied as soon as possible. This case is slightly more difficult because this $\forall_{\mathrm{R}}$ is a right rule. It is done applying lemma 13.5.2 to Π: this produces a proof Π' of $\vdash \varphi(A(x))$ which, in turn, is used to build the following proof:

$$\dfrac{\dfrac{\dfrac{\Pi'}{\vdash \varphi(A(x))}}{\vdash \forall x \varphi(A(x))}\,\forall_{\mathrm{R}}}{\neg\forall x \varphi(A(x)) \vdash}\,\neg_{\mathrm{L}}$$

□

Using these three lemmas, we prove Theorem 13.5.1 as follows:

Theorem 13.5.1.

- If $\Gamma \vdash C$ admits a classical proof, then we can compute from Lemma 13.5.3 a constructive proof of $\chi(\Gamma), \neg\psi(C) \vdash$. From Lemma 13.5.1, we deduce from it a constructive proof of $\chi(\Gamma), \neg\varphi(C) \vdash$, and, from Lemma 13.5.2, we finally compute a constructive proof of $\chi(\Gamma) \vdash \varphi(C)$
- If $\Gamma \vdash$ admits a classical proof, then we can compute from Lemma 13.5.3 a constructive proof of $\chi(\Gamma) \vdash$ admits a constructive one.

Conversely, if such constructive proofs exist, they can be considered as classical proofs, and as any proposition is classically equivalent to its double negation, the original sequent also admits a classical proof. □

The only source of complexity in f_M comes from the use of cut eliminations. However, one can notice that the complexity is much smaller using focused proofs.

In the following section, we show the proof of minimality of f_M according to $\leq^{\downarrow}$. The set of translations that are minimal according to $\leq$ – containing f_M – is also identified.

13.6 Minimality of the translation

The identification of minimal translations according to the partial orders $\leq$ and $\leq^{\downarrow}$ derives from the following theorem:

Theorem 13.6.1. *Let X be a multiset of paths. f_X correct if and only if the following holds: Any path $p \in M$ can be mapped to a subpath p_X of p such that $p_X \in X$ and such that the directed edges completing p_X to p are all members of $H = (\wedge\mathrm{L} \mid \wedge\mathrm{R} \mid\Rightarrow \mathrm{R})$.*

Before showing the details of the proof, we will focus first on its consequences:

Theorem 13.6.2. *The following propositions hold:*

- *Let X be a multiset of paths. f_X is a minimal translation among correct syntax-directed translations according the partial order $\leq$ if and only if the following holds:*
  - *Any path $p \in M$ can be mapped to a subpath p_X of p such that $p_X \in X$ and such that the directed edges completing p_X to p are all members of H.*
  - *X is the set union of all paths p_X.*
- *According to the extended partial order $\leq^{\downarrow}$, f_M is the unique minimal translation among correct syntax-directed translations.*

Proof. The first proposition can be proved as follows. Let f_X be a correct syntax-directed translation. From Theorem 13.6.1, any path $p \in M$ can be mapped to a subpath p_X of p such that $p_X \in X$ and such that the directed edges completing p_X to p are all members of H. Let X' be the set union of all paths p_X. From Theorem 13.6.1, $f_{X'}$ is also correct, and for any strict subset X'' of X', $f_{X''}$ is cannot be correct. Therefore, as $X' \subseteq X$, f_X is minimal according to $\leq$ if and only if $X' = X$.

The second proposition can be proved as follows. As f_M doesn't put any double negation in front of a $\Rightarrow$, a $\wedge$, or a $\neg$ connective, $f_M^{\downarrow} = f_M$. Let f_X be a correct syntax-directed translation. From Theorem 13.6.1, any path $p \in M$ can be mapped to a subpath p_X of p such that $p_X \in X$ and such that the directed edges completing p_X to p are all members of H. Therefore, $f_X^{\downarrow}$ inserts double negations at any path $p \in M$, and, in case $p_X \neq p$, $f_X^{\downarrow}$ inserts strictly more double negations than f_M. Then, either $f_M < f_X^{\downarrow}$, either $f_M = f_X^{\downarrow} = f_x$. As a consequence, f_M is minimal among correct syntax-directed translations according to the extended partial order $\leq^{\downarrow}$. □

The proof of Theorem 13.6.1 is based on the existence of sequents that ensure the existence of some paths in a multiset X when f_X is correct. In this purpose, we introduce the following notion of set of critical paths:

Definition 13.6.1. *Let S be a sequent. A set C_S of paths in S is denoted as the set of critical paths of S if the following holds: any double negation translation of S is constructively provable if and only if it adds at least one double negation at one path of C_S – if such a set exists, it is unique.*

Using this definition, the following lemma holds.

Lemma 13.6.1. *For all path $p \in M$, there exists a sequent S admitting as critical paths all subpaths p' of p such that the directed edges completing p' to p are all members of H.*

Proof. The proof is done step by step from particular to general cases. The first step begins with the paths in $M_1 = +(\forall)^? G$: using for instance Kripke semantics, the lemma can be proved for each one of them:

- $+P$ is the only critical path of $\neg\neg P \vdash P$,
- $+\vee$ is the only critical path of $\vdash P \vee \neg P$,
- $+\exists$ is the only critical path of $\vdash \exists x(\exists y P(y) \Rightarrow P(x))$,
- $+\forall P$ is the only critical path of $\forall z \neg\neg P(z) \vdash \forall z P(z)$,
- $+\forall\vee$ is the only critical path of $\vdash \forall z(P(z) \vee \neg P(z))$,
- $+\forall\exists$ is the only critical path of $\vdash \forall z(\exists x(\exists y P(y,z) \Rightarrow P(x,z)))$.

The second step generalizes the first one to $M_2 = +(\neg^{2*}\forall)^?G$, i.e. to add the paths $+\neg^{2+}\forall G$. Each paths p of this set can be generated inserting double negations from a path $q \in +\forall G \subseteq M_1$. From the result of the previous step, there exists a sequent $\Gamma \vdash C$ admitting q as its single critical path. Inserting in it the number n of double negations that transform q to p, the produced sequent $\Gamma \vdash \neg^{2n}C$ can be proved to admit p as its single critical path.

The third step generalizes the second one to $M_3 = +G \mid (+ \mid -\neg)\neg^{2*}\forall G$, i.e.to add the paths $-\neg\neg^{2*}\forall G$. Each path p of this set can be generated from a path $q \in +\neg^{2*}\forall G \subseteq M_2$. From the result of the previous step, there exists a sequent $\Gamma \vdash C$ such that q is its single critical path. This allows to prove that $\Gamma, \neg C \vdash$ admits p as its single critical path.

The fourth step generalizes the third one to $M_4 = +G \mid (+ \mid -E_-)E_-^{2*}\forall G$. Each path p of M_4 can be generated from a path $q \in M_3$, changing some directed edges $\neg$ to $\Rightarrow$ L in q. Let S be a sequent admitting q as its single path. Mapping some subformulas $\neg A$ to $A \Rightarrow \bot$ according to the transformation of q to p, the produced sequent S' can be proved to admit p as its single critical path.

The last step generalizes the fourth to M. Each path $p \in M$ can be generated from a path $q \in M_4$, adding positive directed edges to q. Let S be a sequent admitting q as its single path. Mapping some subformulas A to $A \wedge \top$, $\top \wedge A$, $\top \Rightarrow A$, $A \vee \bot$, $\bot \vee A$, $\forall x A$, or $\exists x A$ according to the transformation of q to p, the produced sequent S' can be proved to admit as critical paths all subpaths p' of p such that the directed edges completing p' to p are all members of H. □

Theorem 13.6.1 follows from this lemma with the following proof:

Theorem 13.6.1. Let X be a multiset of paths such that f_X is correct. Let $p \in M$. From Lemma 13.6.1, there exists a sequent S admitting as a critical paths all subpaths p' of p such that the directed edges completing p' to p are all members of H. As f_X is correct, it necessarily inserts at least one double negation at one critical path: then, there exists a subpath p_X of p such that $p_X \in X$ and such that the directed edges completing p_X to

p are all members of H.

Conversely, let X be a multiset such that any path $p \in M$ can be mapped to a subpath p_X of p such that $p_X \in X$ and such that the directed edges completing p_X to p are all members of H. Let S be a sequent with a classical proof. From Proposition 13.3.1, the image of S f_X is weaker of equivalent to the image S by f_M: therefore, as f_M is correct, f_X is also correct. □

13.7 Conclusion

This work shows that simple syntax observations are enough to get rid of many of the unnecessary double negations inserted by double-negation translations. According to a comparison of translations based on the alteration of the syntax and the strength of sequents, the problem of finding the best correct syntax-directed translation is solved, leading to the unique minimal translation f_M.

Similar techniques can be used to search the minimal translations among correct syntax-directed translations using techniques different from double-negations, such as Friedman A-translation [31]. Another possible further path is to use extra tools to eliminate the double negations left by the minimal translation f_M. As mentioned earlier, constructive proof search techniques must stay avoided in the context of proof interoperability, but other ways can be followed. One of them is to analyze classical proofs in order to check, for instance, if only one subformula of a disjunction was used in its proof, or if an existential quantification was instantiated by only one term. Another possibility, when working in well-known theories, is to benefit from some of their properties. An interesting case is arithmetic: for instance, as equality is constructively decidable in this theory, decidability proofs can be used to eliminate double negations in front of all equalities. More ambitious translations from classical to constructive proofs in arithmetic could try to eliminate a larger amount of double negations, combining the presented translation with Π^0_2-conservative translations [31, 52].

Chapter 14

Automated constructivization of proofs

No computable function can output a constructive proof from a classical one whenever its associated theorem also holds constructively. We show in this chapter that it is however possible, in practice, to turn a large amount of classical proofs into constructive ones. We describe for this purpose a linear-time constructivization algorithm which is provably complete on large fragments of predicate logic.

14.1 Introduction

Classical and constructive provability match on several specific sets of propositions. In propositional logic, as a consequence of Glivenko's theorem [41], a formula $\neg A$ is a classical theorem if and only if it is a constructive one. In arithmetic, a Π^0_2 proposition is a theorem in Peano arithmetic if and only if it is a theorem in Heyting arithmetic [31].

We present in this work an efficient constructivization algorithm Construct for predicate logic in general, from cut-free classical sequent calculus **LK** to constructive sequent calculus **LJ**. Unlike the two previous examples, constructivization in predicate logic is as hard as constructive theorem proving. Therefore, as we expect Construct to terminate, Construct is incomplete in the sense that it may terminate with a failure output.

Construct consists of three **linear-time** steps:

1. An algorithm Normalize, designed to push occurrences of the right weakening rule towards the root in **LK** proofs. Its purpose is to limit the number of propositions appearing at the right-hand side of sequents in **LK** proofs.

2. A partial translation from cut-free **LK** to a new constructive system **LI**. This algorithm is referred to as Annotate as the **LI** system is designed as **LK** equipped

with specific annotations – making it a constructive system. ANNOTATE is the only step which may fail.

3. A complete translation INTERPRET from **LI** to **LJ**.

The NORMALIZE step taken alone leads to a simple yet efficient constructivization algorithm WEAK CONSTRUCT, which is defined to succeed whenever the result of NORMALIZE happens to be directly interpretable in **LJ**, i.e. to have at most one proposition on the right-hand side of sequents in its proof.

The main property of CONSTRUCT is to be provably **complete** on large fragments of predicate logic, in the sense that for any proposition A in one of these fragments, CONSTRUCT is ensured to terminate successfully on any cut-free **LK** proof of A. Such fragments for which classical and constructive provability match will be referred to as **constructive fragments**. For instance, as a consequence of Glivenko's theorem [41], the set of negated propositions is a **constructive fragment** of propositional logic. The completeness properties of CONSTRUCT lead to the following results:

- The identification of a new constructive fragment F, the fragment of assertions containing no negative occurrence of the connective $\vee$ and no positive occurrence of the connective $\Rightarrow$. Both WEAK CONSTRUCT and CONSTRUCT are provably complete on F.

- The completeness of CONSTRUCT on two already known constructive fragments. The first one, referred to as F_{Ku}, appears as the set of fix points of a polarized version of Kuroda's double-negation translation [51, 13]. The second one, referred to as F_{Ma}, appears as a set of assertions for which any cut-free **LK** proof can be directly interpreted as a proof in Maehara's multi-succedent calculus [55]. Hence, the completeness of CONSTRUCT on these two fragments yields a uniform proof of two results coming from very different works.

After the introduction of basic notations and definitions, the two already known constructive fragments F_{Ku} and F_{Ma} are presented. Then, the NORMALIZE step is presented along with the simple constructivization algorithm WEAK CONSTRUCT. In the following section, the new constructive fragment F is defined, and WEAK CONSTRUCT is proved complete on F. Then, the full constructivization algorithm CONSTRUCT is introduced together with the proof of its completeness on F, F_{Ku} and F_{Ma}. In the last part, experimental results of constructivization using WEAK CONSTRUCT and CONSTRUCT are presented. These experiments are based the classical theorem prover **Zenon** [12] and the constructive proof checker **Dedukti** [11].

14.2 Notations and definitions

In the following, we only consider as primitive the connectives and quantifiers $\forall$, $\exists$, $\wedge$, $\vee$, $\Rightarrow$ and $\bot$. $\neg A$ is defined as $A \Rightarrow \bot$. $\top$, which doesn't appear in this work, could be

defined as $\bot \Rightarrow \bot$.

We use a definition of sequents based on **multisets**. The size of a multiset Γ will be referred to as $|\Gamma|$. We will use the notation (A) to refer to a multiset containing either zero or one element. Given a multiset $\Gamma = A_1, \cdots, A_n$, we will use the notations $\neg\Gamma$ and $\Gamma \Rightarrow B$ as shorthands for $\neg A_1, \cdots, \neg A_n$, and $A_1 \Rightarrow B, \cdots, A_n \Rightarrow B$ respectively. Finally, we use the notation $\bigvee$ to refer to an arbitrary encoding of the n-ary disjunction from the binary one – using $\bot$ for the nullary case.

Definition 14.2.1. *We define the cut-free classical sequent calculus **LK** with the following rules:*

$$\frac{}{\bot \vdash} \bot_L \quad \frac{}{A \vdash A}\, axiom$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta}\, weak_L \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \Delta'}\, weak_R$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\, contr_L \quad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\, contr_R$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} \wedge_L \quad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta} \wedge_R$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} \vee_L \quad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta} \vee_R$$

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \Rightarrow B \vdash \Delta} \Rightarrow_L \quad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta} \Rightarrow_R$$

$$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta} \forall_L \quad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x A, \Delta} \forall_R$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x A \vdash \Delta} \exists_L \quad \frac{\Gamma \vdash A[t/x], \Delta}{\Gamma \vdash \exists x A, \Delta} \exists_R$$

with the standard freshness constraints for the variables introduced in the rules $\forall_R$ and $\exists_L$.

Definition 14.2.2. *We define the constructive sequent calculus **LJ** from **LK**, applying the following changes:*

- *All rules except $contr_R$, $\vee_R$, $\Rightarrow_L$ are restricted to sequents with at most one proposition on the right-hand side of sequents.*

For instance, $\wedge_R$ *becomes* $\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_R$

- *There is no* $contr_R$ *rule*
- *The* $\vee_R$ *rule is split into two rules* $\dfrac{\Gamma \vdash A_i}{\Gamma \vdash A_0 \vee A_1} \vee_R$
- *The* $\Rightarrow_L$ *rule becomes* $\dfrac{\Gamma \vdash A \qquad \Gamma, B \vdash (C)}{\Gamma, A \Rightarrow B \vdash (C)} \Rightarrow_L$
- *We add a cut rule* $\dfrac{\Gamma \vdash A \qquad \Gamma, A \vdash (B)}{\Gamma \vdash (B)}\, cut$

Remark 14.2.1. *In these presentations of* ***LK*** *and* ***LJ****,*

- *weakenings are applied to multisets instead of propositions*
- $\bot_L$ *and axiom are not relaxed to* $\dfrac{}{\Gamma, \bot \vdash \Delta} \bot_L$ *and* $\dfrac{}{\Gamma, A \vdash A, \Delta}\, axiom$

These specific conventions are chosen to ease the definition of the algorithm Normalize *in Section 14.5, which requires pushing weakenings towards the root of the proof.*

Definition 14.2.3. *We introduce the following notations in* ***LK****, along with their constructive analogs in* ***LJ****:*

- $\dfrac{}{\Gamma, A \vdash A, \Delta}\, axiom^*$ *for* $\dfrac{\dfrac{\dfrac{}{A \vdash A}\, axiom}{\Gamma, A \vdash A}\, weak_L}{\Gamma, A \vdash A, \Delta}\, weak_R$
- $\dfrac{}{\Gamma, \bot \vdash \Delta} \bot_L^*$ *for* $\dfrac{\dfrac{\dfrac{}{\bot \vdash}\, \bot_L}{\Gamma, \bot \vdash}\, weak_L}{\Gamma, \bot \vdash \Delta}\, weak_R$
- $\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta} \neg_L$ *for* $\dfrac{\Gamma \vdash A, \Delta \qquad \dfrac{}{\Gamma, \bot \vdash \Delta} \bot_L^*}{\Gamma, \neg A \vdash \Delta} \Rightarrow_L$
- $\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta} \neg_R$ *for* $\dfrac{\dfrac{\Gamma, A \vdash \Delta}{\Gamma, A \vdash \bot, \Delta}\, weak_R}{\Gamma \vdash \neg A, \Delta} \Rightarrow_R$

14.3 State of the art: two constructive fragments of predicate logic

Constructive sequent calculus – as well as constructive natural deduction – extends the notion of constructive provability from propositions to sequents of the shape $\Gamma \vdash (G)$, which will be referred to as **mono-succedent sequents**. As a consequence, we will define constructive fragments of predicate logic as sets of mono-succedent sequents instead of sets of simple propositions.

The definitions of these fragments will be based on the usual notion of polarity of occurrences of connectives, quantifiers and atoms in a sequent: given a sequent $\Gamma \vdash \Delta$,

- the root of a proposition in Γ is negative, the root of a proposition in Δ is positive
- polarity only changes between an occurrence of $A \Rightarrow B$ and the occurrence of its direct subformula A (in particular, as $\neg A$ is defined as $A \Rightarrow \bot$, it changes between $\neg A$ and its direct subformula A)

Definition 14.3.1. *We define the following fragments of predicate logic:*

- F_{Ku}*, the fragment of sequents of the shape* $\Gamma \vdash$ *containing no positive occurrence of* $\forall$*.*
- F_{Ma}*, the fragment of mono-succedent sequents containing no positive occurrence of* $\forall$ *and no positive occurrence of* $\Rightarrow$*.*

Theorem 14.3.1. F_{Ku} *is a constructive fragment of predicate logic: for any sequent* $\Gamma \vdash$ *in* F_{Ku}*,* $\Gamma \vdash$ *is classically provable if and only if it is constructively provable.*

The key arguments to prove this theorem as an adaptation of Kuroda's double negation translation [51] are the following:

1. Kuroda's double negation translation [51] is based on a double negation translation $|\cdot|_{Ku}$ inserting double-negations after any occurrence of $\forall$. The original theorem is that a proposition A is classically provable if and only if $\neg\neg|A|_{Ku}$ is constructively provable.
2. It can adapted in two ways. First, $|\cdot|_{Ku}$ can be lightened to insert double negations only after positive occurrences of $\forall$ as shown in [13], and extended from propositions to contexts. Second, the main statement can be turned to the following one: a classical sequent $\Gamma \vdash \Delta$ is classically provable if and only if $|\Gamma, \neg\Delta|_{Ku} \vdash$ is constructively provable
3. By definition of F_{Ku}, a sequent $\Gamma \vdash$ in F_{Ku} admits the property $\Gamma = |\Gamma|_{Ku}$, hence $\Gamma \vdash$ is classically provable if and only if it is constructively provable.

We do not provide more details on this proof as the completeness of Construct on F_{Ku} shown in Section 14.6 will yield a new proof of this result.

Remark 14.3.1. *One could expect similar constructive fragments to be found using other double negation translations, such as Gödel-Gentzen's [43, 33] or Kolmogorov's [49]. Unfortunately, these two translations always insert double-negations in front of atoms, hence they cannot be easily modified to leave a large fragment of propositions unchanged.*

Theorem 14.3.2. *F_{Ma} is a constructive fragment of predicate logic: for any sequent $\Gamma \vdash (G)$ in F_{Ma}, $\Gamma \vdash (G)$ is classically provable if and only if it is constructively provable.*

It lies on a key idea: polarity restrictions have a direct influence on the shape of cut-free proofs. It can be presented in the following way:

Lemma 14.3.1. *For any connective or quantifier X and any cut-free **LK** proof Π of a sequent $\Gamma \vdash \Delta$:*

- *If $\Gamma \vdash \Delta$ contains no positive occurrence of X, then Π doesn't contain the rule X_R.*
- *If $\Gamma \vdash \Delta$ contains no negative occurrence of X, then Π doesn't contain the rule X_L.*

This lemma can be proved directly by induction on cut-free **LK** proofs. Using this lemma, the key arguments to prove Theorem 14.3.2 are the following:

1. All **LK** rules except $\Rightarrow_R$ and $\forall_R$ rules belong in Maehara's multi-succedent calculus [55], a constructive multi-succedent sequent calculus.
2. By lemma 14.3.1, F_{Ma} sequents are proved by cut-free **LK** proofs without the $\Rightarrow_R$ and $\forall_R$ rules.
3. Hence, a sequent $\Gamma \vdash (G)$ in F_{Ma} is classically provable if and only if it is constructively provable.

Again, we do not provide more details on this proof as the completeness of Construct on F_{Ma} shown in Section 14.6 will yield a new proof of this result.

Remark 14.3.2. *The same fragment F_{Ma} can be found using similar multi-succedent constructive systems, such as Dragalin's calculus GHPC [28].*

14.4 The weakening normalization

A naive constructivization algorithm can be defined by selecting **LK** proofs which can be directly interpreted in **LJ**.

In this direct interpretation, premises of the classical rules $\vee_R$ and $\Rightarrow_L$ may be multi-succedent only when they are introduced by a $weak_R$ whose premise is a mono-succedent sequent. For instance, the classical derivation

$$\dfrac{\dfrac{\Gamma \vdash A}{\Gamma \vdash A, B}\, weak_R}{\Gamma \vdash A \vee B}\, \vee_R \text{ can be interpreted as } \dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\, \vee_R .$$

However, in practice, the $weak_R$ rule doesn't appear as low as possible – in presentations using multi-succedents *axiom* rules, they may not appear at all. Such situations are problematic for constructive interpretations: for instance, a classical proof such as

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\, axiom}{A \vdash A, B}\, weak_R}{\vdash A \Rightarrow A, B}\, \Rightarrow_R}{\vdash (A \Rightarrow A) \vee B}\, \vee_R$$

cannot be interpreted in **LJ** directly because the $weak_R$ rule doesn't occur immediately above the $\vee_R$ rule.

The NORMALIZE algorithm is designed to address this issue, pushing the application of $weak_R$ as low as possible in proofs. In its definition, we need to consider all possible configuration of $weak_R$ appearing above a **LK** rule. In order to factor this definition, we partition all such configurations into three classes **A**, **B**, and **C**.

These definitions will be based on the following notation of **LK** proofs:

Definition 14.4.1. *We write any cut-free **LK** rule X as*

$$\dfrac{\Gamma, L_1 \vdash R_1, \Delta \qquad \cdots \qquad \Gamma, L_n \vdash R_n, \Delta}{\Gamma, L \vdash R, \Delta}\, X$$

*where L_1, $\cdots$, L_n, R_1, $\cdots$, R_n, L and R are the (possibly empty) **multisets** of propositions containing the active propositions of the rule X.*

For instance, in the rule $\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}\, \Rightarrow_R$,

$L_1 = \{A\}$, $R_1 = \{B\}$, $L = \emptyset$, and $R = \{A \Rightarrow B\}$.

The classes **A**, **B**, and **C** are defined as follows:

Definition 14.4.2. *We consider all configurations where $weak_R$ appears above a **LK** rule X, in its i-th premise:*

$$\dfrac{\cdots \qquad \dfrac{\Gamma, L_i \vdash \Delta_i}{\Gamma, L_i \vdash R_i, \Delta}\, weak_R \qquad \cdots}{\Gamma, L \vdash R, \Delta}\, X$$

This weakening can be done on propositions in R_i, in Δ or both: in the general case, we only know $\Delta_i \subseteq (R_i, \Delta)$. We define the following partition of all cases:

- ***A***: $R_i \subseteq \Delta_i$

- ***B***: $R_i \not\subseteq \Delta_i$ *and* $\Delta_i \subseteq \Delta$

- ***C***: $R_i \not\subseteq \Delta_i$ *and* $\Delta_i \not\subseteq \Delta$. *This only happens when* $|R_i| = 2$, *when exactly one proposition of* R_i *is in* Δ_i.

Definition 14.4.3. NORMALIZE *is a linear-time algorithm associating any cut-free* ***LK*** *proof of a sequent* $\Gamma \vdash \Delta$ *to a proof of a sequent* $\Gamma \vdash \Delta'$, *where* $\Delta' \subseteq \Delta$. *It is defined recursively. Using the conventions of Definition 14.4.1, we describe the original proof* Π *as*

$$\dfrac{\dfrac{\Pi_1}{\Gamma, L_1 \vdash R_1, \Delta} \quad \cdots \quad \dfrac{\Pi_n}{\Gamma, L_n \vdash R_n, \Delta}}{\Gamma, L \vdash R, \Delta}\, X$$

The definition of NORMALIZE(Π) *is based on the analysis of the proof*

$$\dfrac{\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma, L_1 \vdash \Delta_1}}{\Gamma, L_1 \vdash R_1, \Delta}\, weak_R \quad \cdots \quad \dfrac{\dfrac{\text{NORMALIZE}(\Pi_n)}{\Gamma, L_n \vdash \Delta_n}}{\Gamma, L_n \vdash R_n, \Delta}\, weak_R}{\Gamma, L \vdash R, \Delta}\, X$$

The different cases are the following:

- *Case* ***1****: for all index* i, ***A*** *holds, i.e.* $R_i \subseteq \Delta_i$.
  If X *is* $weak_R$, *we define* NORMALIZE(Π) *as* NORMALIZE(Π_1).

  Else, writing $\Delta_i = R_i, \Delta_i'$, *we define* NORMALIZE($\Pi$) *as*

$$\dfrac{\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma, L_1 \vdash R_1, \Delta_1'}}{\Gamma, L_1 \vdash R_1, \Delta'}\, weak_R \quad \cdots \quad \dfrac{\dfrac{\text{NORMALIZE}(\Pi_n)}{\Gamma, L_n \vdash R_n, \Delta_n'}}{\Gamma, L_n \vdash R_n, \Delta'}\, weak_R}{\Gamma, L \vdash R, \Delta'}\, X$$

  where Δ' *is the smallest multiset containing all multisets* Δ_i'

- *Case* ***2****: there exists a smallest premise* i *for which* ***B*** *holds, i.e.* $R_i \not\subseteq \Delta_i$ *and* $\Delta_i \subseteq \Delta$. *As* $R_i \neq \emptyset$, *ether* X *is* $\Rightarrow_R$ *or* $L_i = \emptyset$.

If X is $\Rightarrow_R$, we define NORMALIZE(Π) *as* $\dfrac{\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma, A \vdash \Delta_1}}{\Gamma, A \vdash B, \Delta_1}\, weak_R}{\Gamma \vdash A \Rightarrow B, \Delta_1} \Rightarrow_R$

Else, $L_i = \emptyset$ and we define NORMALIZE(Π) *as* $\dfrac{\dfrac{\text{NORMALIZE}(\Pi_i)}{\Gamma \vdash \Delta_i}}{\Gamma, L \vdash \Delta_i}\, weak_L$

- *Case **3**: there exists a smallest premise i for which the case **C** applies, i.e. $R_i \not\subseteq \Delta_i$ and $\Delta_i \not\subseteq \Delta$. This only happens when $|R_i| = 2$, when exactly one proposition of R_i is in Δ_i. In this case, X is either $contr_R$ or $\vee_R$.*

  If X is $contr_R$, we can write $R_1 = A, A$, and $\Delta_1 = (A, \Delta_1')$ with $\Delta_1' \subseteq \Delta$. We define NORMALIZE($\Pi$) *as* NORMALIZE($\Pi_1$).

  If X is $\vee_R$, we can write $R_1 = A_0, A_1$, and $\Delta_1 = (A_k, \Delta_1')$ with $\Delta_1' \subseteq \Delta$.

  We define NORMALIZE(Π) *as* $\dfrac{\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma \vdash A_k, \Delta_1'}}{\Gamma \vdash A_0, A_1, \Delta_1'}\, weak_R}{\Gamma \vdash A_0 \vee A_1, \Delta_1'} \vee_R$

Remark 14.4.1. *The nullary rules axiom and $\bot_L$ having no premise, they match the first case.*

Definition 14.4.4. *We define a first constructivization algorithm* WEAK CONSTRUCT, *which*

- *takes as input a cut-free **LK** proof* $\dfrac{\Pi}{\Gamma \vdash (G)}$,
- *computes the proof* $\dfrac{\text{NORMALIZE}(\Pi)}{\Gamma \vdash (G)}\, weak_R$,
- *outputs its **LJ** interpretation if it exists and fails otherwise*

14.5 A new constructive fragment

Definition 14.5.1. *We define F as the fragment of mono-succedent sequents containing no negative occurrence of $\vee$ and no positive occurrence of $\Rightarrow$.*

Theorem 14.5.1. WEAK CONSTRUCT *is complete on F: if Π is a cut-free **LK** proof of a sequent $\Gamma \vdash (G) \in F$, then* WEAK CONSTRUCT(Π) *succeeds.*

Proof. By Lemma 14.3.1, F sequents are proved by cut-free **LK** proofs containing no $\vee_L$ or $\Rightarrow_R$ rule. We prove that for any such proof Π, NORMALIZE(Π) proves a mono-succedent sequent interpretable in **LJ**. This proof is done by induction on cut-free **LK** proofs containing no $\vee_L$ or $\Rightarrow_R$ rule, following the partition of cases and the notations introduced in the definition of NORMALIZE:

- Case **1**: we split this case according to the rule X.

  - nullary rules: *axiom* and $\bot_L$ are interpretable in **LJ**.

  - $weak_R$: The result follows directly by induction hypothesis.

  - other unary rules: In these cases $\Delta' = \Delta'_1$, hence NORMALIZE(Π) is

    $$\dfrac{\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma, L_1 \vdash R_1, \Delta'_1}}{\Gamma, L_1 \vdash R_1, \Delta'_1}\, weak_R}{\Gamma, L \vdash R, \Delta'_1}\, X$$

    By induction hypothesis, NORMALIZE(Π_1) is interpretable in **LJ**. Hence, $|R_1| \leq 1$, which ensures that X is neither $contr_R$ nor $\vee_R$. All other unary rules lead to a proof interpretable in **LJ**, therefore the result is interpretable in **LJ**.

  - $\vee_L$: This case doesn't occur by hypothesis

  - $\Rightarrow_L$: By induction hypothesis, NORMALIZE(Π_1) and NORMALIZE(Π_2) are interpretable in **LJ**, hence $|R_1, \Delta'_1| \leq 1$. As $|R_1| = 1$, $\Delta'_1 = \emptyset$, and $\Delta' = \Delta'_2$.

    As $\dfrac{\dfrac{\Gamma \vdash A}{\Gamma \vdash A, \Delta'_2}\, weak_R \quad \dfrac{\Gamma, B \vdash \Delta'_2}{\Gamma, B \vdash \Delta'_2}\, weak_R}{\Gamma, A \Rightarrow B \vdash \Delta'_2}\, \Rightarrow_L$

    is interpretable as $\dfrac{\Gamma \vdash A \qquad \Gamma, B \vdash \Delta'_2}{\Gamma, A \Rightarrow B \vdash \Delta'_2}\, \Rightarrow_L$ in **LJ**, the result follows.

  - $\wedge_R$: By induction hypothesis, NORMALIZE(Π_1) and NORMALIZE(Π_2) are interpretable in **LJ**, hence $|R_1, \Delta'_1| \leq 1$ and $|R_2, \Delta'_2| \leq 1$. As $|R_1| = |R_2| = 1$, $\Delta'_1 = \Delta'_2 = \emptyset$. Therefore $\Delta' = \emptyset$, from which the result follows.

- Case **2**: By hypothesis, X is not $\Rightarrow_R$, hence NORMALIZE(Π) is defined as

$$\dfrac{\dfrac{\textsc{Normalize}(\Pi_i)}{\Gamma \vdash \Delta_i}}{\Gamma, L \vdash \Delta_i}\, weak_L$$

The result follows by induction hypothesis.

- Case **3**: If X is $contr_R$, the result follows directly by induction hypothesis. Else, X is $\vee_R$. By induction hypothesis, $\textsc{Normalize}(\Pi_1)$ is interpretable in **LJ**, thus $|A_k, \Delta_1'| \leq 1$, and $\Delta_1' = \emptyset$.

  As $\dfrac{\dfrac{\Gamma \vdash A_k}{\Gamma \vdash A_0, A_1}\, weak_R}{\Gamma \vdash A_0 \vee A_1}\, \vee_R$ is interpretable as $\dfrac{\Gamma \vdash A_k}{\Gamma \vdash A_0 \vee A_1}\, \vee_R$ in **LJ**,

  the result follows.

□

Corollary 14.5.1. *The fragment F is a constructive fragment of predicate logic: a sequent $\Gamma \vdash (G)$ is classically provable if and only if it is constructively provable.*

14.6 The full constructivization algorithm

The previous algorithm Weak construct was based on the reject of multi-succedent sequents. The idea leading to our main algorithm Construct is to try to interpret multi-succedent sequents constructively as well. This interpretation is based on a new multi-succedent constructive system, which will be referred to as **LI** in the following. As mentioned in the introduction, the constructivization algorithm Construct comprises three steps: first the algorithm Normalize, then a partial translation Annotate from **LK** proofs to **LI** proofs, and finally a complete translation Interpret from **LI** proof to **LJ** proofs.

There are several ways to interpret multi-succedent sequents constructively. For instance, $\Gamma \vdash \bigvee \Delta$ and $\Gamma, \neg\Delta \vdash$ are two possible interpretations of a multi-succedent sequent $\Gamma \vdash \Delta$. These interpretation are equivalent classically but not constructively: for instance, the classical sequent $\vdash A, \neg A$ is not provable constructively under the first interpretation, but it is provable constructively under the second one. As a consequence, some classical rules may be constructively valid or not according to the chosen interpretation of classical sequents.

The new system **LI** is built to benefit from the freedom left in the constructive interpretation of classical sequents. **LI** is designed as a sequent calculus based on **annotated sequents**, where the annotation will refer to the choice of constructive interpretation of the underlying classical sequent. We formalize first the notion of **annotated sequents**.

Definition 14.6.1. *We define the set of* ***annotated sequents*** *as sequents of the shape* $\Gamma \vdash \Delta_1; \Delta_2$.

We define the following interpretation INTERPRET *on annotated sequents:*
INTERPRET$(\Gamma \vdash \Delta_1; \Delta_2) \quad = \quad \Gamma, \neg\Delta_2 \vdash \bigvee \Delta_1$.
In the following, this function will be extended from ***LI*** *proofs to* ***LJ*** *proofs.*

We define the following erasure function ERASE *on annotated sequents:*
ERASE$(\Gamma \vdash \Delta_1; \Delta_2) \quad = \quad \Gamma \vdash \Delta_1, \Delta_2$.
In the following, this function will be extended from ***LI*** *proofs to* ***LK*** *proofs.*

Then, we define the system **LI** in the following way:

Definition 14.6.2. ***LI*** *is based on the following rules:*

$$\frac{}{\bot \vdash;}\,\bot_L \qquad \frac{}{A \vdash A;}\,axiom^1 \qquad \frac{}{A \vdash; A}\,axiom^2$$

$$\frac{\Gamma \vdash \Delta_1; \Delta_2}{\Gamma, \Gamma' \vdash \Delta_1; \Delta_2}\,weak_L \qquad \frac{\Gamma \vdash \Delta_1; \Delta_2}{\Gamma \vdash \Delta_1, \Delta_1'; \Delta_2, \Delta_2'}\,weak_R$$

$$\frac{\Gamma, A, A \vdash \Delta_1; \Delta_2}{\Gamma, A \vdash \Delta_1; \Delta_2}\,contr_L \qquad \frac{\Gamma \vdash A, A, \Delta_1; \Delta_2}{\Gamma \vdash A, \Delta_1; \Delta_2}\,contr_R^1 \qquad \frac{\Gamma \vdash \Delta_1; A, A, \Delta_2}{\Gamma \vdash \Delta_1; A, \Delta_2}\,contr_R^2$$

$$\frac{\Gamma, A, B \vdash \Delta_1; \Delta_2}{\Gamma, A \wedge B \vdash \Delta_1; \Delta_2}\,\wedge_L \qquad \frac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma \vdash B, \Delta_1; \Delta_2}{\Gamma \vdash A \wedge B, \Delta_1; \Delta_2}\,\wedge_R^1$$

$$\frac{\Gamma \vdash; A, \Delta_2 \qquad \Gamma \vdash; B, \Delta_2}{\Gamma \vdash; A \wedge B, \Delta_2}\,\wedge_R^2 \qquad \frac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma \vdash B, \Delta_1; \Delta_2}{\Gamma \vdash \Delta_1; A \wedge B, \Delta_2}\,\wedge_R^3,\ |\Delta_1| \geq 1$$

$$\frac{\Gamma, A \vdash \Delta_1; \Delta_2 \qquad \Gamma, B \vdash \Delta_1; \Delta_2}{\Gamma, A \vee B \vdash \Delta_1; \Delta_2}\,\vee_L$$

$$\frac{\Gamma \vdash A, B, \Delta_1; \Delta_2}{\Gamma \vdash A \vee B, \Delta_1; \Delta_2}\,\vee_R^1 \qquad \frac{\Gamma \vdash \Delta_1; A, B, \Delta_2}{\Gamma \vdash \Delta_1; A \vee B, \Delta_2}\,\vee_R^2$$

$$\frac{\Gamma \vdash; A, \Delta_2 \qquad \Gamma, B \vdash; \Delta_2}{\Gamma, A \Rightarrow B \vdash; \Delta_2}\,\Rightarrow_L^1 \qquad \frac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma, B \vdash \Delta_1; \Delta_2}{\Gamma, A \Rightarrow B \vdash \Delta_1; \Delta_2}\,\Rightarrow_L^2,\ |\Delta_1| \geq 1$$

$$\frac{\Gamma, A \vdash B; \Delta_2}{\Gamma \vdash A \Rightarrow B; \Delta_2}\,\Rightarrow_R^1 \qquad \frac{\Gamma, A \vdash; B, \Delta_2}{\Gamma \vdash; A \Rightarrow B, \Delta_2}\,\Rightarrow_R^2$$

$$\frac{\Gamma, A[t/x] \vdash \Delta_1; \Delta_2}{\Gamma, \forall x A \vdash \Delta_1; \Delta_2}\,\forall_L \qquad \frac{\Gamma \vdash A; \Delta_2}{\Gamma \vdash \forall x A; \Delta_2}\,\forall_R^1 \qquad \frac{\Gamma \vdash A; \Delta_2}{\Gamma \vdash; \forall x A, \Delta_2}\,\forall_R^2$$

$$\frac{\Gamma, A \vdash \Delta_1; \Delta_2}{\Gamma, \exists x A \vdash \Delta_1; \Delta_2}\,\exists_L \qquad \frac{\Gamma \vdash A[t/x], \Delta_1; \Delta_2}{\Gamma \vdash \exists x A, \Delta_1; \Delta_2}\,\exists_R^1 \qquad \frac{\Gamma \vdash \Delta_1; A[t/x], \Delta_2}{\Gamma \vdash \Delta_1; \exists x A, \Delta_2}\,\exists_R^2$$

with the standard freshness constraints for the variables introduced in the rules $\forall_R^i$ and $\exists_L$.

All **LI** rules correspond to a **LK** rule through the erasure of the premises and the conclusions. Hence, we can extend the ERASE function from **LI** rules to **LK** rules, and consequently from **LI** proofs to **LK** proofs.

In the same way, we would like to extend the INTERPRET function from **LI** proofs to **LJ** proofs. This can done associating each **LI** rule to a partial **LJ** proof deriving the interpretation of its conclusion from the interpretation of its premises. However, such an approach would be heavy: as the disjunction in **LJ** is binary, $\bigvee$ is a based on a nesting of binary disjunctions, and a proposition in $\Gamma \vdash \Delta_1; \Delta_2$ can occur deep in $\Gamma, \neg\Delta_2 \vdash \bigvee \Delta_1$. As INTERPRET will be part of the constructivization algorithm CONSTRUCT, we need to find another method to define it as a linear-time algorithm.

For this reason, we will define the interpretation of rules using the property that $\Gamma \vdash \bigvee \Delta$ is constructively provable if and only if $\Gamma, \Delta \Rightarrow G \vdash G$ is provable for any proposition G.

Definition 14.6.3. *We define the function* $\text{INTERPRET}'(\cdot \mid G)$ *on annotated sequents as* $\text{INTERPRET}'(\Gamma \vdash \Delta_1; \Delta_2 \mid G) = (\Gamma, \Delta_1 \Rightarrow G, \neg\Delta_2 \vdash G)$.

We extend INTERPRET′ *from* ***LI*** *rules to partial* ***LJ*** *derivations in the following way:*

From a ***LI*** *rule* $\dfrac{\Gamma^1 \vdash \Delta_1^1; \Delta_2^1 \quad \cdots \quad \Gamma^n \vdash \Delta_1^n; \Delta_2^n}{\Gamma \vdash \Delta_1; \Delta_2} R$

and a proposition G*, we define a partial* ***LJ*** *derivation* $\text{INTERPRET}'(R \mid G)$ *as a partial derivation of the form*

$$\dfrac{\text{INTERPRET}'(\Gamma^1 \vdash \Delta_1^1; \Delta_2^1 \mid G^1) \quad \cdots \quad \text{INTERPRET}'(\Gamma^n \vdash \Delta_1^n; \Delta_2^n \mid G^n)}{\dfrac{\vdots}{\text{INTERPRET}'(\Gamma \vdash \Delta_1; \Delta_2 \mid G)}}$$

The ***LI*** *system is designed to ensure that such definitions rely on simple constructive tautologies.As an illustration, we present here the case of the rule*

$$\dfrac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma, B \vdash \Delta_1; \Delta_2}{\Gamma, A \Rightarrow B \vdash \Delta_1; \Delta_2} \Rightarrow_L^3$$

From a proposition G*, defining* $\Sigma = \Gamma, \Delta_1 \Rightarrow G, \neg\Delta_2$*, we derive*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma, A \vdash A}\, axiom^* \quad \dfrac{\Sigma, B \vdash G}{\Sigma, B, A \vdash G}\, weak_L}{\Sigma, A \Rightarrow B, A \vdash G} \Rightarrow_L}{\Sigma, A \Rightarrow B \vdash A \Rightarrow G} \Rightarrow_R \qquad \dfrac{\Sigma, A \Rightarrow G \vdash G}{\Sigma, A \Rightarrow B, A \Rightarrow G \vdash G}\, weak_L}{\Sigma, A \Rightarrow B \vdash G}\, cut$$

where the two open premises correspond to $\textsc{Interpret}'(\Gamma, B \vdash \Delta_1; \Delta_2 \mid G)$ *and* $\textsc{Interpret}'(\Gamma \vdash A, \Delta_1; \Delta_2 \mid G)$ *respectively.*

Remark 14.6.1. *In this case, we chose* $G_1 = G_2 = G$*. Other choices for* G_i *appear in the cases* $\wedge_R^2$, $\Rightarrow_L^1$, $\Rightarrow_R^2$, *and* $\forall_R^2$.

In a second step, we extend $\textsc{Interpret}'(\cdot \mid G)$ *from* ***LI*** *proofs to* ***LJ*** *proofs recursively. Finally, we extend* $\textsc{Interpret}(\cdot)$ *from* ***LI*** *proofs of sequents of the shape* $\Gamma \vdash (G)$; *to* ***LJ*** *proofs:*

- *for* Π *a* ***LI*** *proof of a sequent* $\Gamma \vdash$;, *we define* $\textsc{Interpret}(\Pi)$ *as*

$$\dfrac{\dfrac{}{\Gamma, \bot \vdash}\,\bot_L^* \qquad \dfrac{\textsc{Interpret}'(\Pi \mid \bot)}{\Gamma \vdash \bot}}{\Gamma \vdash}\,cut$$

- *for* Π *a* ***LI*** *proof of a sequent* $\Gamma \vdash G$;, *we define* $\textsc{Interpret}(\Pi)$ *as*

$$\dfrac{\dfrac{\textsc{Interpret}'(\Pi \mid G)}{\Gamma, G \Rightarrow G \vdash G} \qquad \dfrac{\dfrac{}{\Gamma, G \vdash G}\,axiom^*}{\Gamma \vdash G \Rightarrow G}\Rightarrow_R}{\Gamma \vdash G}\,cut$$

Definition 14.6.4. *We define the linear-time partial algorithm* $\textsc{Annotate}(\cdot \mid \cdot)$ *with, as inputs, a* ***LI*** *sequent* S *and a cut-free* ***LK*** *proof* Π *of* $\textsc{Erase}(S)$ *and, as output, either a* ***LI*** *proof of* S *or a failure. This annotation is done from the root to the leaves: at each step, the first argument* S *prescribe how the current conclusion must be annotated. The definition is recursive on the second argument.*

Describing S *as* $\Gamma \vdash \Delta_1; \Delta_2$ *and* Π *as* $\dfrac{\dfrac{\Pi^1}{\Gamma^1 \vdash \Delta^1} \quad \cdots \quad \dfrac{\Pi^n}{\Gamma^n \vdash \Delta^n}}{\Gamma \vdash \Delta_1, \Delta_2}\,R$,

- *If there exists a* ***LI*** *rule*

$$\dfrac{\Gamma^1 \vdash \Delta_1^1; \Delta_2^1 \qquad \cdots \qquad \Gamma^n \vdash \Delta_1^n; \Delta_2^n}{\Gamma \vdash \Delta_1; \Delta_2}\,R'$$

  such that for all i, $\Delta_1^i, \Delta_2^i = \Delta^i$, *then the output is*

$$\dfrac{\dfrac{\textsc{Annotate}(\Gamma^1 \vdash \Delta_1^1; \Delta_2^1 \mid \Pi^1)}{\Gamma^1 \vdash \Delta_1^1; \Delta_2^1} \quad \cdots \quad \dfrac{\textsc{Annotate}(\Gamma^n \vdash \Delta_1^n; \Delta_2^n \mid \Pi^n)}{\Gamma^n \vdash \Delta_1^n; \Delta_2^n}}{\Gamma \vdash \Delta_1; \Delta_2}\,R'$$

- *Else,* $\textsc{Annotate}(\cdot, \cdot)$ *fails.*

Remark 14.6.2. *The only failing cases appear when the rule R is either $\Rightarrow_R$ or $\forall_R$, and exclusively for sequents $\Gamma \vdash \Delta_1; \Delta_2$ such that $|\Delta_1, \Delta_2| > 1$.*

Definition 14.6.5. *We define the linear-time constructivization algorithm* Construct*, which*

- *takes as input a cut-free **LK** proof Π of a sequent $\Gamma \vdash (G)$,*
- *computes the proof $\Pi' = \dfrac{\textsc{Normalize}(\Pi)}{\Gamma \vdash (G)}\, weak_R$,*
- *outputs* Interpret(Annotate($\Gamma \vdash (G); \mid \Pi'$)) *if it exists and fails otherwise.*

Example 14.6.1. *We consider the law of excluded middle $A \vee \neg A$ given with the following **LK** proof:* $\dfrac{\dfrac{\dfrac{}{A \vdash A}\, axiom}{\vdash A, \neg A} \Rightarrow_R}{\vdash A \vee \neg A} \vee_R$ *. This proof is unchanged by* Normalize*.*

The Annotate *step fails as follows:* $\dfrac{\dfrac{\textsc{Failure}}{\vdash A, \neg A;}}{\vdash A \vee \neg A;} \vee^1_R$

Example 14.6.2. *We consider a variant of the non contradiction of law of excluded middle, $(\neg(A \vee \neg A)) \Rightarrow B$, given with the proof:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A, B}\, axiom^*}{\vdash A, \neg A, B} \Rightarrow_R}{\vdash A \vee \neg A, B} \vee_R \qquad \dfrac{}{\bot \vdash B}\, \bot^*_L}{\dfrac{\neg(A \vee \neg A) \vdash B}{\vdash (\neg(A \vee \neg A)) \Rightarrow B} \Rightarrow_R} \Rightarrow_L$$

The result of Normalize *is*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\, axiom}{\vdash A, \neg A} \Rightarrow_R}{\vdash A \vee \neg A} \vee_R \qquad \dfrac{}{\bot \vdash}\, \bot_L}{\neg(A \vee \neg A) \vdash} \Rightarrow_L}{\dfrac{\neg(A \vee \neg A) \vdash B}{\vdash (\neg(A \vee \neg A)) \Rightarrow B} \Rightarrow_R}\, weak_R$$

Then, the result of Annotate *is*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash; A}\, axiom^2}{\vdash; A, \neg A} \Rightarrow^2_R}{\vdash; A \vee \neg A} \vee^2_R \qquad \dfrac{}{\bot \vdash;}\, \bot_L}{\neg(A \vee \neg A) \vdash;} \Rightarrow^1_L}{\dfrac{\neg(A \vee \neg A) \vdash B;}{\vdash (\neg(A \vee \neg A)) \Rightarrow B;} \Rightarrow^1_R}\, weak_R$$

As Annotate *is the only step which may fail,* Construct *succeeds on this example. We see on the example that the application of* Normalize *was crucial for* Annotate *to succeed.*

Theorem 14.6.1. CONSTRUCT *is complete on* F, F_{Ku}, *and* F_{Ma}*: for any proof* Π *of a sequent* S *in one of these fragments,* CONSTRUCT(Π) *succeeds.*

Proof. We consider F, F_{Ku}, and F_{Ma} separately:

- For F: we consider a cut-free **LK** proof Π of a sequent $\Gamma \vdash (G) \in F$.

  By Theorem 14.5.1, $\Pi' = \dfrac{\text{NORMALIZE}(\Pi)}{\Gamma \vdash (G)}\, weak_R$ is interpretable in **LJ**.

  As a consequence, the only multi-succedent sequents in Π' are conclusions of weakenings. As all failing cases (cf. Remark 14.6.2) involve sequents $\Gamma \vdash \Delta_1; \Delta_2$ such that $|\Delta_1, \Delta_2| > 1$ which are conclusions of $\Rightarrow_R$ or $\forall_R$ rules, ANNOTATE succeeds. Hence, CONSTRUCT succeeds.

- For F_{Ku}: the result follows directly from a stronger assertion: for any cut-free **LK** proof Π of a sequent $\Gamma \vdash \Delta$ containing no $\forall_R$ rule, ANNOTATE($\Gamma \vdash; \Delta \mid \Pi$) succeeds. This assertion is proved by induction on such sequents and proofs, noticing that all induction hypotheses refer to sequents of the shape $\Gamma' \vdash; \Delta'$.

- For F_{Ma}: we consider a cut-free **LK** proof Π of a sequent in F_{Ma}. As mentioned in Remark 14.6.2 the only failing cases involve the $\Rightarrow_R$ or $\forall_R$ rules, which do not occur in a proof of a sequent in F_{Ma}. Hence, CONSTRUCT succeeds.

□

14.7 Experimental results

In order to measure the success of CONSTRUCT in practice, experiments were made on the basis of **TPTP** [71] first-order problems. The classical theorem prover **Zenon** [12] was used to prove such problems. **Zenon** builds cut-free **LK** proofs internally. It was instrumented to use these internal proofs as inputs for an implementation of WEAK CONSTRUCT and CONSTRUCT. The **LJ** proofs obtained as outputs were expressed and checked in the constructive logical framework **Dedukti** [11].

A set of 724 **TPTP** problems was selected for the experimentations, corresponding to all **TPTP** problems in the category **FOF** which could be proved in less than 1 second using the uninstrumented version of **Zenon**. The results are the following:

- WEAK CONSTRUCT led to constructive proofs in 51% of tested cases.
- CONSTRUCT led to constructive proofs in 85% of tested cases (including all WEAK CONSTRUCT successes).

All constructive proofs generated were successfully checked using **Dedukti**. Among all cases where CONSTRUCT failed, 35% are proved to be invalid constructively using the constructive theorem prover **ileanCoP** [59].

14.8 Conclusion

In this work, we presented a linear-time constructivization algorithm CONSTRUCT from cut-free classical sequent calculus **LK** to constructive sequent calculus **LJ**, which is provably complete on three large fragments of predicate logic:

- F_{Ku}, the fragment of sequents of the shape $\Gamma \vdash$ containing no positive occurrence of $\forall$.
- F_{Ma}, the fragment of mono-succedent sequents containing no positive occurrence of $\forall$ and no positive occurrence of $\Rightarrow$.
- F, the fragment of mono-succedent sequents containing no negative occurrence of $\vee$ and no positive occurrence of $\Rightarrow$.

The two first fragments of predicate logic were already known as fragments on which classical provability matches constructive provability, as a direct consequence of Kuroda's double-negation translation and Maehara's definition of multi-succedent sequent calculus respectively. Yet, the completeness of the constructivization algorithm on the third fragment yields a first proof of the fact that classical provability matches constructive provability on this fragment.

Several further perspectives can be studied from this work. For instance, the linear-time constraint could be relaxed to investigate more advances algorithms, implying for instance the permutation between some rules in order to improve the performances of constructivization. For instance, applying firstly all **LK** rules which are both constructively valid and invertible could be one interesting approach.

Another perspective, which remains compatible with a linear-time constraint, would be to specialize this algorithm to arithmetic, or more generally to theories in which atomic predicates are logically decidable: for instance, this additional hypothesis could lead to the definition of some additional **LI** rule of the following form, to be applied as much as possible to improve the performances of the constructivization algorithm:

$$\frac{\Gamma \vdash \Delta_1; P, \Delta_2}{\Gamma \vdash P, \Delta_1; \Delta_2}\ P \text{ atomic}$$

Bibliography

[1] Behzad Akbarpour and Lawrence Charles Paulson. Metitarski: An automatic theorem prover for real-valued special functions. Journal of Automated Reasoning, 44(3):175–205, 2010.

[2] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a logical framework based on the lambda-picalculus modulo theory, 2016.

[3] Henk Barendregt. Introduction to generalized type systems. Journal of functional programming, 1(2):125–154, 1991.

[4] Henk Barendregt. Lambda calculi with types, handbook of logic in computer science vol. ii, 1992.

[5] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. The Coq proof assistant reference manual: Version 6.1. PhD thesis, Inria, 1997.

[6] Gilles Barthe. Type-checking injective pure type systems. Journal of Functional Programming, 9(06):675–698, 1999.

[7] Stefano Berardi. Towards a mathematical analysis of the coquand-huet calculus of constructions and the other systems in barendregt's cube. Technica1 report, Carnegie-Me11on University (USA) and Universita di Torino (Ita1y), 1988.

[8] Stefano Berardi. Type dependence and constructive mathematics. PhD thesis, PhD thesis, Dipartimento di Informatica, Torino, Italy, 1990.

[9] Stephano Berardi. Non-conservativity of coquand's calculus with respect to higher-order intuitionistic logic. In Talk given in the 3rd Jumelage meeting on Typed Lambda Calculi, Edinburgh, 1989.

[10] Bruno Bernardo. An implicit Calculus of Constructions with dependent sums and decidable type inference. Phd thesis, École polytechnique, October 2015.

[11] Mathieu Boespflug, Quentin Carbonneaux, and Olivier Hermant. The λΠ-calculus modulo as a universal proof language. In David Pichardie and Tjark Weber, editors, PxTP, 2012.

[12] Richard Bonichon, David Delahaye, and Damien Doligez. Zenon: An extensible automated theorem prover producing checkable proofs. In Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings, pages 151–165, 2007.

[13] Mélanie Boudard and Olivier Hermant. Polarizing double-negation translations. In Ken McMillan, Aart Middeldorp, and Andreï Voronkov, editors, LPAR, volume 8312 of LNCS ARCoSS, pages 182–197. Springer, 2013.

[14] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of agda–a functional language with dependent types. In International Conference on Theorem Proving in Higher Order Logics, pages 73–78. Springer, 2009.

[15] Alonzo Church. A note on the entscheidungsproblem. The journal of symbolic logic, 1(1):40–41, 1936.

[16] Alonzo Church. An unsolvable problem of elementary number theory. American journal of mathematics, 58(2):345–363, 1936.

[17] Alonzo Church. A formulation of the simple theory of types. The journal of symbolic logic, 5(2):56–68, 1940.

[18] Thierry Coquand. An analysis of Girard's paradox. PhD thesis, INRIA, 1986.

[19] Thierry Coquand and Gérard Huet. The calculus of constructions. Information and computation, 76(2-3):95–120, 1988.

[20] Thierry Coquand and Christine Paulin. Inductively defined types. In COLOG-88, pages 50–66. Springer, 1990.

[21] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In ACM sigplan notices, volume 35, pages 233–243. ACM, 2000.

[22] David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand, and Olivier Hermant. Zenon modulo: When achilles outruns the tortoise using deduction modulo. In International Conference on Logic for Programming Artificial Intelligence and Reasoning, pages 274–290. Springer, 2013.

[23] Nachum Dershowitz. Jumping and escaping: Modular termination and the abstract path ordering. Theoretical Computer Science, 464:35–47, 2012.

[24] Ben L Di Vito. Manip user's guide, version 1.3. 2011.

[25] Daniel J Dougherty. Adding algebraic rewriting to the untyped lambda calculus. Information and Computation, 101(2):251–267, 1992.

[26] Gilles Dowek. Collections, sets and types. Mathematical Structures in Computer Science, 9(1):109–123, 1999.

[27] Gilles Dowek. On the definition of the classical connectives and quantifiers. 2013.

[28] Albert Grigorevich Dragalin and Elliott Mendelson. Mathematical intuitionism. introduction to proof theory. 1990.

[29] Gilda Ferreira and Paulo Oliva. On various negative translations. arXiv preprint arXiv:1101.5442, 2011.

[30] Gottlob Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. L. Nebert, 1879.

[31] Harvey Friedman. Classically and intuitionistically provably recursive functions. In Higher set theory, pages 21–27. Springer, 1978.

[32] Kokichi Futatsugi, Joseph A Goguen, Jean-Pierre Jouannaud, and José Meseguer. Principles of obj2. In Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pages 52–66. ACM, 1985.

[33] Gerhard Gentzen. Über das verhältnis zwischen intuitionistischer und klassischer arithmetik. Archive for Mathematical Logic, 16(3):119–132, 1974.

[34] Herman Geuvers. Talk given at the jumelage meeting on typed lambda calculus'. Edinburgh, September, 1989.

[35] Herman Geuvers. A short and flexible proof of strong normalization for the calculus of constructions. In International Workshop on Types for Proofs and Programs, pages 14–38. Springer, 1994.

[36] Herman Geuvers and Mark-Jan Nederhof. Modular proof of strong normalization for the calculus of constructions. Journal of Functional Programming, 1(02):155–189, 1991.

[37] Frédéric Gilbert. A lightweight double-negation translation. In LPAR-20. 20th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, 2015.

[38] Frédéric Gilbert. Automated constructivization of proofs. In International Conference on Foundations of Software Science and Computation Structures, pages 480–495. Springer, 2017.

[39] Frédéric Gilbert. Proof certificates in pvs. In International Conference on Interactive Theorem Proving, pages 262–268. Springer, 2017.

[40] Jean-Yves Girard. Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur. PhD thesis, PhD thesis, Université Paris VII, 1972.

[41] Valery Glivenko. Sur quelques points de la logique de m. brouwer. Bulletins de la classe des sciences, 15(5):183–188, 1929.

[42] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. Monatshefte für mathematik und physik, 38(1):173–198, 1931.

[43] Kurt Gödel. Zur intuitionistischen arithmetik und zahlentheorie. Ergebnisse eines mathematischen Kolloquiums, 4(1933):34–38, 1933.

[44] Kurt Gödel, Stephen Cole Kleene, and John Barkley Rosser. On undecidable propositions of formal mathematical systems. 1934.

[45] John Harrison. Hol light: An overview. In TPHOLs, volume 5674, pages 60–66. Springer, 2009.

[46] David Hilbert and Wilhelm Ackermann. Grundzüge der theoretischen logik. 1938.

[47] Joe Hurd. The opentheory standard theory library. NASA Formal Methods, pages 177–191, 2011.

[48] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: introduction and survey. Theoretical Computer Science, 121(1):279 – 308, 1993.

[49] Andrei Nikolaevich Kolmogorov. On the principle of excluded middle. Mat. Sb, 32(646-667):24, 1925.

[50] Jean-Louis Krivine. Opérateurs de mise en mémoire et traduction de gödel. Archive for Mathematical Logic, 30(4):241–267, 1990.

[51] Sigekatu Kuroda et al. Intuitionistische untersuchungen der formalistischen logik. Nagoya Mathematical Journal, 2:35–47, 1951.

[52] Daniel Leivant. Syntactic translations and provably recursive functions. The Journal of Symbolic Logic, 50(03):682–688, 1985.

[53] Zhaohui Luo. Ecc, an extended calculus of constructions. In Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on, pages 386–395. IEEE, 1989.

[54] Zhaohui Luo. A problem of adequacy: conservativity of calculus of constructions over higher-order logic. Laboratory for Foundations of Computer Science, Department of Computer Science, Univ., 1990.

[55] Shôji Maehara et al. Eine darstellung der intuitionistischen logik in der klassischen. Nagoya mathematical journal, 7:45–64, 1954.

[56] Per Martin-Löf and Giovanni Sambin. Intuitionistic type theory, volume 9. Bibliopolis Napoli, 1984.

[57] Chetan R Murthy. Extracting constructive content from classical proofs. Technical report, Cornell University, 1990.

[58] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. Isabelle/hol: A proof assistant for higher-order logic (lecture notes in computer science), 2002.

[59] Jens Otten. leancop 2.0and ileancop 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings, pages 283–291, 2008.

[60] Sam Owre, John M Rushby, and Natarajan Shankar. Pvs: A prototype verification system. In International Conference on Automated Deduction, pages 748–752. Springer, 1992.

[61] Sam Owre and Natarajan Shankar. The formal semantics of pvs. 1999.

[62] Sam Owre, Natarajan Shankar, John M Rushby, and David WJ Stringer-Calvert. Pvs language reference. Computer Science Laboratory, SRI International, Menlo Park, CA, 1(2):21, 1999.

[63] Christine Paulin-Mohring. Extracting ω's programs from proofs in the calculus of constructions. In Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 89–104. ACM, 1989.

[64] John Rushby, Sam Owre, and Natarajan Shankar. Subtypes for specifications: Predicate subtyping in pvs. IEEE Transactions on Software Engineering, 24(9):709–720, 1998.

[65] Bertrand Russell. Letter to frege. From Frege to Gödel, pages 124–125, 1902.

[66] Ronan Saillard. Dedukti: a universal proof checker. In Foundation of Mathematics for Computer-Aided Formalization Workshop, 2013.

[67] Paula Severi and Erik Poll. Pure type systems with definitions. Logical Foundations of Computer Science, pages 316–328, 1994.

[68] Natarajan Shankar, Sam Owre, John M Rushby, and Dave WJ Stringer-Calvert. Pvs prover guide. Computer Science Laboratory, SRI International, Menlo Park, CA, 1:11–12, 2001.

[69] Konrad Slind and Michael Norrish. A brief overview of hol4. Theorem Proving in Higher Order Logics, pages 28–32, 2008.

[70] Matthieu Sozeau. Subset coercions in coq. In International Workshop on Types for Proofs and Programs, pages 237–252. Springer, 2006.

[71] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning, 43(4):337–362, 2009.

[72] Geoff Sutcliffe. The tptp problem library and associated infrastructure. Journal of Automated Reasoning, 43(4):337, 2009.

[73] William W Tait. A realizability interpretation of the theory of species. In Logic Colloquium, pages 240–251. Springer, 1975.

[74] Jan Terlouw. Een nadere bewijstheoretische analyse van gstt's. Manuscript (in Dutch), 1989.

[75] Jan Terlouw. Sterke normaliszatie in c a la tait. In Notes of atalk held at the Intercity Seminar on Typed Lambda Calculus, Nijmegen, Netherlands, 1989.

[76] Jan Terlouw. Strong normalization in type systems: A model theoretical approach. Annals of Pure and Applied Logic, 73(1):53–78, 1995.

[77] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. Proceedings of the London mathematical society, 2(1):230–265, 1937.

[78] Benjamin Werner. Une théorie des constructions inductives. PhD thesis, Université Paris-Diderot-Paris VII, 1994.

[79] Benjamin Werner. On the strength of proof-irrelevant type theories. In IJCAR, volume 4130, pages 604–618. Springer, 2006.

[80] Alfred North Whitehead and Bertrand Russell. Principia mathematica, volume 2. University Press, 1912.