View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by INRIA a CCSD electronic archive server

Verifiable certificates for predicate subtyping

Frédéric Gilbert

► To cite this version:

Frédéric Gilbert. Verifiable certificates for predicate subtyping. 2019. hal-01977585

HAL Id: hal-01977585
https://hal.inria.fr/hal-01977585

Preprint submitted on 10 Jan 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.

Verifiable certificates for predicate subtyping

Frederic Gilbert

Inria,
frederic.a.gilbert@inria.fr

Abstract. Adding *predicate subtyping* to higher-order logic yields a very expressive language in which type-checking is undecidable, making the definition of a system of verifiable certificates challenging. This work presents a solution to this issue with a minimal formalization of predicate subtyping, named PVS-Core, together with a system of verifiable certificates for PVS-Core, named PVS-Cert. PVS-Cert is based on the introduction of proof terms and explicit coercions. Its design is similar to that of PTSs with dependent pairs, at the exception of the definition of conversion, which is based on a specific notion of reduction $\rightarrow_{\beta*}$, corresponding to β-reduction combined with the *erasure of coercions*. The use of this reduction instead of the more standard reduction $\rightarrow_{\beta\sigma}$ allows to establish a simple correspondence between PVS-Core and PVS-Cert. On the other hand, a type-checking algorithm is designed for PVS-Cert, built on proofs of type preservation of $\rightarrow_{\beta\sigma}$ and strong normalization of both $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta*}$. Using these results, PVS-Cert judgements are used as verifiable certificates for predicate subtyping. In addition, the reduction $\rightarrow_{\beta\sigma}$ is used to define a cut elimination procedure adapted to predicate subtyping. Its use to study the properties of predicate subtyping is illustrated with a proof of consistency.

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

1 Introduction

Extending higher-order logic with *predicate subtyping* yields a very expressive type system, used notably at the core of the proof system PVS [17]. However, proof judgements and typing judgements become entangled in the presence of predicate subtyping, making type-checking undecidable. As a consequence, defining a language of verifiable proofs for predicate subtyping becomes challenging. In pure higher-order logic, complete judgement derivations are too heavy to be used in practice as certificates, but lighter certificates can be produced by removing typing rules, recording deduction rules only: as this approach requires the decidability of type-checking, it doesn't apply directly to predicate subtyping.

This paper presents a new formal language, PVS-Cert, designed to be used as a language of verifiable certificates for predicate subtyping. PVS-Cert is built starting from a minimal formalization of predicate subtyping named PVS-Core, by adding explicit proofs and coercions. PVS-Cert is also equipped with a notion of *cut elimination*, which can be used directly to study both PVS-Cert and PVS-Core meta-theoretical properties.

1.1 Extending higher-order logic with predicate subtyping

Higher-order logic is characterized by the coexistence of *types* and *predicates* as two radically different kinds of attributes to mathematical expressions. 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 assignment of types remains very simple: in particular, type-checking is decidable in higher-order logic. In return, most attributes of mathematical expressions formulated as predicates cannot be formulated as types: for instance, being a natural number different from 0 is expressible as a predicate, but not as a type.

Predicate subtyping allows to recover a symmetrical situation between the expressivity of types and predicates. It 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 $\{x : A \mid P(x)\}$ 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 1 admits the type $\{x : \mathit{Nat} \mid \mathit{Nonzero}(x)\}$ as long as $\mathit{Nonzero}(1)$ is provable.

This augmented expressivity of the language of types permits to exclude many unwanted expressions from reasoning. For instance, defining the denominators domain of Euclidean division as $\{x : \mathit{Nat} \mid \mathit{Nonzero}(x)\}$, all divisions in which the denominator is not provably different from zero become ill-typed.

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

As previously mentioned, a major counterpart of this extension of higher-order logic is the fact that typing judgements and proof judgements become entangled. For instance, proving the equality $(1/1) = 1$ requires that 1 can be assigned the type $\{x : \mathit{Nat}|\mathit{Nonzero}(x)\}$, which, in turn, requires that $\mathit{Nonzero}(1)$ is provable. As a direct consequence, type-checking is not decidable in the presence of predicate subtyping.

1.2 Contributions

PVS-Core Higher-order logic, as well as its extension with predicate subtyping, can be defined in various ways. The first contribution of this paper is the formalization, in Section 2, of a minimal system for predicate subtyping, denoted PVS-Core. Besides its minimality, the main design choice for this system is the use of β-equivalence as a conversion relation (or definitional equality).

PVS-Cert and its basic properties Starting from PVS-Core, the second contribution of this work is the formalization, in Section 3, 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.

PVS-Cert is an extension of the Pure Type System (PTS) λ-HOL (see for instance [4], where λ-HOL as well as the general notion of PTS are defined). More precisely, PVS-Cert is designed to extend λ-HOL in the same way as PVS-Core extends higher-order logic (denoted HOL in the following). This situation is illustrated in this diagram, where vertical arrows represent extensions and horizontal arrows represent the introduction of explicit proofs (and, in the case of PVS-Core and PVS-Cert, of explicit coercions).

$$\begin{tikzcd}
\textbf{PVS-Core} \arrow[r] & \textbf{PVS-Cert} \\
\textbf{HOL} \arrow[u] \arrow[r] & \lambda\textbf{-HOL} \arrow[u]
\end{tikzcd}$$

This choice of a PTS-like system is well-suited to describe reasoning modulo β: all steps of β-reduction or β-expansion are kept implicit in proof terms, which allows to keep them compact. As detailed in Section 3.3, PVS-Cert is comparable to the formalism of PTSs with dependent pairs. However, conversion in PVS-Cert is neither defined as $\equiv_\beta$ nor as its extension $\equiv_{\beta\sigma}$ (see for instance [16]) used in PTSs with dependent pairs: instead, it uses a new conversion relation $\equiv_{\beta*}$ corresponding to syntactical equality modulo β-reduction *and coercion erasure* (defined in Section 3.1). This distinctive definition allows to define a simple correspondence between PVS-Core and PVS-Cert – presented later in Section 9.

Basic properties of PVS-Cert are presented in Section 4, containing notably the Church-Rosser property for the reduction $\rightarrow_{\beta*}$ underlying the conversion $\equiv_{\beta*}$, as well as the uniqueness of types: contrary to the case of PVS-Core, a well-typed term admits a unique type up to $\equiv_{\beta*}$.

As in λ-HOL, well-typed terms are organized according to a stratification, presented in Section 5, which includes a class of *types*, a class of *expressions* (containing notably propositions), and a class of *proof terms*. This stratification is at the core of the correspondence between PVS-Cert and PVS-Core.

Type preservation and strong normalization 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 in Section 6 that $\rightarrow_{\beta\sigma}$ is a type preserving reduction in PVS-Cert (Theorem 6).

In Section 7, we present the main ideas leading to a proof of strong normalization for both $\rightarrow_{\beta*}$ and $\rightarrow_{\beta\sigma}$ (Theorem 7) – the details of the proof can be found in the author's PhD dissertation [1]. Moreover, the strong normalization of the type preserving reduction $\rightarrow_{\beta\sigma}$ defines a *cut elimination theorem* (Theorem

8). This theorem is used in the following of this section to prove the consistency of PVS-Cert. This result is used in turn at the very end of this work to conclude the consistency of PVS-Core, illustrating how cut elimination in PVS-Cert can be used to study the meta-theoretical properties of predicate subtyping.

Type-checking in PVS-Cert We present in Section 8 the design of a type-checking algorithm for PVS-Cert, showing that, contrary to the case of PVS-Core, type-checking is decidable in PVS-Cert. This algorithm is based on the type preservation of $\rightarrow_{\beta\sigma}$ as well as the strong normalization of $\rightarrow_{\beta*}$ and $\rightarrow_{\beta\sigma}$.

Using PVS-Cert as a system of verifiable certificates for PVS-Core The connection between PVS-Core and PVS-Cert is formalized in Section 9. On the one hand, a translation from PVS-Cert to PVS-Core is defined through the erasure of coercions. On the other hand, the choice of conversion $\equiv_{\beta*}$ in PVS-Cert allows to define a very simple translation from PVS-Core derivations to PVS-Cert derivable judgements (Definition 7 and Theorem 11).

These translations are used in Section 10 together with the PVS-Cert type-checking algorithm to define how to use PVS-Cert judgements as verifiable certificates for PVS-Core (Definition 8), reaching the first purpose of this paper. Such certificates are much lighter than the PVS-Core *derivations* represented through them, as they only require to record one single judgement.

Last, the translations between PVS-Core and PVS-Cert are exploited to transpose the consistency property, established in PVS-Cert using cut elimination, to PVS-Core. This illustrates how the PVS-Cert cut elimination theorem can be used to study both PVS-Cert and PVS-Core meta-theoretical properties.

1.3 Related works

The most important related work is the author's PhD dissertation [1], which contains detailed versions of all proofs presented in this paper.

The introduction of predicate subtyping can be traced back to the first-order language OBJ2 [9] 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 for instance in [17] and [20].

In the present work, the issue of the undecidability of predicate subtyping is handled with the introduction of an alternative system, PVS-Cert. An alternative approach to this issue is to weaken the definition of predicate subtyping sufficiently to obtain systems in which type-checking remains decidable. This approach has been followed in [19, 13]. A intermediate situation is followed in [15], in which predicate subtyping is weakened sufficiently to allow for run-time type-checking verifications. However, contrary to the case of PVS, predicate subtyping is not fully represented in these different systems.

As mentioned in the previous section, 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, 24–26, 11, 3]. More specifically, PVS-Cert is derived from the notion of PTSs *with dependent pairs*, which has its roots in the system ECC [16]. A subsystem of PVS-Cert, named PVS-Cert$^-$ and presented in Section 3, corresponds directly to a fragment of ECC (PVS-Cert$^-$ is the system obtained from PVS-Cert by replacing $\equiv_{\beta *}$ by the standard conversion $\equiv_{\beta\sigma}$ of PTSs with dependent pairs). PVS-Cert$^-$ is also comparable to the notion of *subset types* in Coq [5]. However, contrary to PVS-Cert, PVS-Cert$^-$ and subset types are not well-suited to reflect predicate subtyping, as conversion in these systems do not reflect conversion in PVS-Core – more precisely, Proposition 5 doesn't hold with $\equiv_{\beta\sigma}$.

Another important related work is [8], in which two systems are presented: ICC$_\Sigma$, a type system with *implicit* type constructions, and AICC$_\Sigma$, a system obtained from ICC$_\Sigma$ by adding *explicit* coercions. ICC$_\Sigma$ contains several advanced features, including a generalization of predicate subtypes. The construction of PVS-Cert from PVS-Core follows the same idea as the construction of AICC$_\Sigma$ from ICC$_\Sigma$: 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$_\Sigma$ is a very rich and complex language, making its analysis difficult – in particular, strong normalization in ICC$_\Sigma$ 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 simpler.

A variant of predicate subtyping was also formalized as an extension of the calculus of constructions in [22]. 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 the functioning of PVS.

Contrary to the case of PVS-Core and Russell, PVS-Cert and the counterpart of Russell with explicit coercions have similar characteristics. Although its theoretical properties are not formalized, this latter system is presented as a simple extension of the proof-irrelevant type theory presented in [27]. There exists indeed a tight connection between proof irrelevance and PVS-Cert: if one considers for instance the usual predicate *Even* on natural numbers expressing divisibility by two, the predicate subtype $\mathit{even} = \{x : \mathit{Nat} \mid \mathit{Even}(x)\}$, and two expression with explicit coercions $\langle 2, p\rangle_{\mathit{even}}$ and $\langle 2, q\rangle_{\mathit{even}}$ of this type with p and q two proofs of $\mathit{Even}(2)$, then the hypothesis of proof irrelevance ensures that the expressions $\langle 2, p\rangle_{even}$ and $\langle 2, q\rangle_{even}$ are convertible, as does the choice of conversion relation $\equiv_{\beta *}$ in PVS-Cert.

This relation between proof irrelevance and predicate subtyping is explored further in [27]. 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 possible strategy for building a proof of strong normalization for this system from the proof of strong normalization presented in Section 7. Also following the relation between proof irrelevance and predicate subtyping, the system IITT presented in [2], which is equipped with explicit occurrences of irrelevant terms, also admits some similarities with PVS-Cert. However, it is restricted to predicative type theory, in which higher-order reasoning cannot be expressed.

Another important work carried out on predicate subtyping is the presentation of *formal semantics* for PVS in [18]. 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. Such an approach is complementary to the presented paper, which is only focused on the distinction between *provable* propositions and *unprovable* ones. As a possible future work, it would be interesting to adapt the work presented in [18] to obtain a notion of *standard model* for PVS-Core.

2 PVS-Core: a minimal extension of HOL with predicate subtyping

This section is dedicated to the first contribution of this work: the formalization of a minimal system for predicate subtyping. This system is named PVS-Core, in reference to PVS [17]. The main distinctive design choice for PVS-Core is the introduction of a conversion relation (or definitional equality), corresponding to β-equivalence.

2.1 Definitions

Variables and terms We first define a 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: $A, B := X \mid \mathit{Prop} \mid \varPi x : A.B \mid \{x : A \mid P\}$
- The last set is the set of **expressions**. It is given with the following grammar: $t, u, P, Q := x \mid \forall x : A.P \mid P \Rightarrow Q \mid \lambda x : A.t \mid tu$

Remark 1. 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.*

Declarations, contexts, judgements We define:

- Three kinds of **declarations**:
  $X : \mathit{Type} \mid x : A \mid P$
- **Contexts**, denoted $\varGamma$, as lists of declarations:
  $\varGamma := \varnothing \mid \varGamma, X : \mathit{Type} \mid \varGamma, x : A \mid \varGamma, P$
- Four kinds of **judgements**:
  $\varGamma \vdash WF \mid \varGamma \vdash A : \mathit{Type} \mid \varGamma \vdash t : A \mid \varGamma \vdash P$

We use the notation $DV(\varGamma)$ to refer to the set of variables declared in a context $\varGamma$: for instance, $DV(P, x : A, X : \mathit{Type}) = \{x, X\}$.

Reduction We equip PVS-Core terms with the usual β-reduction. In the following, we use the notation $\rhd_\beta$ for the reduction of a β-redex, $\rightarrow_\beta$ for the context closure of $\rhd_\beta$, $\twoheadrightarrow_\beta$ for the reflexive transitive closure of $\rhd_\beta$, and $\equiv_\beta$ for the symmetric closure of $\twoheadrightarrow_\beta$, i.e. β-conversion.

Derivation rules The rules of PVS-Core are the following:

Well-formed contexts

$$\frac{}{\varnothing \vdash WF}\ \textsc{Empty} \qquad \frac{\varGamma \vdash WF}{\varGamma, X : \mathit{Type} \vdash WF}\ \textsc{TypeDecl}\ X \in \mathcal{V}_{types} \backslash DV(\varGamma)$$

$$\frac{\varGamma \vdash P : \mathit{Prop}}{\varGamma, P \vdash WF}\ \textsc{Assumption} \qquad \frac{\varGamma \vdash A : \mathit{Type}}{\varGamma, x : A \vdash WF}\ \textsc{EltDecl}\ x \in \mathcal{V}_{expressions} \backslash DV(\varGamma)$$

Well-formed types

$$\frac{\varGamma \vdash WF}{\varGamma \vdash X : \mathit{Type}}\ \textsc{TypeVar}\ (X : \mathit{Type}) \in \varGamma \qquad \frac{\varGamma \vdash WF}{\varGamma \vdash \mathit{Prop} : \mathit{Type}}\ \textsc{Prop}$$

$$\frac{\varGamma, x : A \vdash B : \mathit{Type}}{\varGamma \vdash \varPi x : A.B : \mathit{Type}}\ \textsc{Pi} \qquad \frac{\varGamma, x : A \vdash P : \mathit{Prop}}{\varGamma \vdash \{x : A \mid P\} : \mathit{Type}}\ \textsc{Subtype}$$

Well-typed expressions

$$\frac{\Gamma \vdash WF}{\Gamma \vdash x : A}\ \textsc{EltVar}\ (x : A) \in \Gamma \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \mathit{Type}}{\Gamma \vdash t : B}\ \textsc{TypeConversion}\ A \equiv_\beta B$$

$$\frac{\Gamma, x : A \vdash P : \mathit{Prop}}{\Gamma \vdash \forall x : A.P : \mathit{Prop}}\ \textsc{Forall} \qquad \frac{\Gamma, P \vdash Q : \mathit{Prop}}{\Gamma \vdash P \Rightarrow Q : \mathit{Prop}}\ \textsc{Imply}$$

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

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : \mathit{Type}}{\Gamma \vdash t : \{x : A \mid P\}}\ \textsc{SubtypeIntro}$$

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash t : A}\ \textsc{SubtypeElim1}$$

Deductions

$$\frac{\Gamma \vdash WF}{\Gamma \vdash P}\ \textsc{Axiom}\ P \in \Gamma \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q : \mathit{Prop}}{\Gamma \vdash Q}\ \textsc{PropConversion}\ P \equiv_\beta Q$$

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q}\ \textsc{ImplyIntro} \qquad \frac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}\ \textsc{ImplyElim}$$

$$\frac{\Gamma, x : A \vdash P}{\Gamma \vdash \forall x : A.P}\ \textsc{ForallIntro} \qquad \frac{\Gamma \vdash \forall x : A.P \qquad \Gamma \vdash t : A}{\Gamma \vdash P[t/x]}\ \textsc{ForallElim}$$

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash P[t/x]}\ \textsc{SubtypeElim2}$$

2.2 A minimal system expressing of predicate subtyping

Predicate subtyping is expressed in PVS-Core with the term construction $\{x : A \mid P\}$ and the following rules:

- Subtype, the rule of formation of predicate subtypes.
- SubtypeIntro, which is a rule of introduction.
- SubtypeElim1 and SubtypeElim2, which are rules of elimination.

The system obtained from PVS-Core by removing the construction $\{x : A \mid P\}$ and these four rules is a formulation of constructive higher-order logic. In particular, the types of this subsystem correspond to the expected simple types : for any type of the form $\Pi x : A.B$ in this subsystem, x cannot appear free in B, hence this type is a non-dependent function type. As a consequence, the rule TypeConversion can be safely removed from this subsystem to obtain a simpler but equivalent formulation of higher-order logic.

PVS-Core is a minimal constructive system, which can be extended with classical reasoning or extensionality principles through the addition of axioms.

The rule PropConversion, in which conversion is a side condition instead of a premise, allows to consider reasoning *modulo* β, which will be useful in the definition of PVS-Cert to keep proof terms compact. The rule TypeConversion is its counterpart at the level of typing, allowing to consider both types and expressions *modulo* β.

3 PVS-Cert: verifiable certificates for PVS-Core

This section is dedicated to the presentation of an alternative system, PVS-Cert, which will be used to achieve the purpose of the work: defining a language of verifiable certificates for predicate subtyping.

At first glance, there is no need to introduce any new system to design PVS-Core certificates: 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. As a comparison, in higher-order logic, as type-checking is decidable, only the deduction rules need to be recorded.

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, and to introduce explicit coercions based on these proof terms in order to ensure the decidability of type-checking. As a consequence, a complete certificate is simply the typing judgement of some proof term with its corresponding theorem. Such certificates are much lighter than PVS-core derivations, as only one single judgement is recorded.

Moreover, PVS-Cert will be equipped (in Section 7) with a definition of *cut elimination*, defined as a computation rule on proof terms.

3.1 Definitions

As detailed further in Section 3.2, the definition of PVS-Cert is strongly related to the formalism of PTSs, presented for instance in [4].

Terms We define:

- **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 s such that $v \in \mathcal{V}_s$.

- **Terms** $\mathcal{T}$ is given by the following grammar:
  $M, N, T, U := s \mid v \mid \lambda v : T.M \mid MN \mid \Pi v : T.U \mid \{v : T \mid U\} \mid \langle M, N\rangle_T \mid \pi_1(M) \mid \pi_2(M)$

Contexts, judgements We define:

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

As in PVS-Core, set of variables declared in a context Γ is denoted $DV(\Gamma)$.

Reduction The main specificity of PVS-Cert is the use of a distinctive notion of reduction and conversion. In addition to the usual β-redex reduction $(\lambda v : T.M)N \rhd_\beta M[N/v]$, we introduce a new reduction relation $\rhd_*$, defined with the following rules:

- $\langle M_1, M_2\rangle_T \rhd_* M_1$
- $\pi_1(M) \rhd_* M$

We denote the union of $\rhd_\beta$ and $\rhd_*$ as $\rhd_{\beta *}$. As in the definition of PVS-Core, we use the notation $\rightarrow_{\beta *}$ for the context closure of $\rhd_{\beta *}$, $\twoheadrightarrow_{\beta *}$ for the reflexive transitive closure of $\rhd_{\beta *}$, and $\equiv_{\beta *}$ for the symmetric closure of $\twoheadrightarrow_{\beta *}$.

The new relation $\rhd_*$, which can be interpreted as a the elimination of a coercion at the head of a term, allows the expression of predicate subtyping in PVS-Cert. More detailed motivations and justifications for this definition are given in Section 3.3.

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

$$\frac{}{\varnothing \vdash WF}\ \textsc{Empty} \qquad \frac{\Gamma \vdash T : s}{\Gamma, v : T \vdash WF}\ \textsc{Decl}\ v \in \mathcal{V}_s \backslash DV(\Gamma)$$

$$\frac{\Gamma \vdash WF}{\Gamma \vdash v : T}\ \textsc{Var}\ (v : T) \in \Gamma \qquad \frac{\Gamma \vdash M : T \qquad \Gamma \vdash U : s}{\Gamma \vdash M : U}\ \textsc{Conversion}\ T \equiv_{\beta *} U$$

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

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

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

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

$$\frac{\Gamma \vdash M : \{v : T \mid U\}}{\Gamma \vdash \pi_1(M) : T}\ \textsc{Proj1} \qquad \frac{\Gamma \vdash M : \{v : T \mid U\}}{\Gamma \vdash \pi_2(M) : U[\pi_1(M)/v]}\ \textsc{Proj2}$$

3.2 An extension of λ-HOL

PVS-Cert is an extension of the PTS λ-HOL (see for instance [4]). More precisely, λ-HOL can be obtained from PVS-Cert by removing the term constructions $\{v : T \mid U\}$, $\pi_i(M)$, and $\langle M, N\rangle_T$, removing the rules Subtype, Pair, Proj1, and Proj2, and replacing $\equiv_{\beta *}$ by $\equiv_\beta$ in the Conversion rule.

As PTS-like systems, the formalism of PVS-Cert allows to describe reasoning modulo β: all steps of β-reduction or β-expansion in reasoning are kept implicit, which allows to keep proofs terms compact, making PVS-Cert more scalable. Moreover, the choice of formalization of PVS-Cert as a PTS-like system allows to transpose some PTS properties to PVS-Cert, such as the thinning property and the substitution property mentioned in the next section. It also allows to describe this system using a small number of rules in comparison with PVS-Core, making the proof of certain expected properties of PVS-Cert lighter.

The well-typed terms of PVS-Cert are classified into the same classes as in the case λ-HOL, involving a class of *types*, a class of *expressions*, and a class of *proof terms*. This property is presented in Section 5, and referred to as *stratification*.

3.3 Expressing predicate subtyping

The expression of predicate subtyping in PVS-Cert is enlightened through the *stratification*: indeed, in any derivable judgement,

- terms of the form $\{v : T \mid U\}$ are *types*, expressing predicate subtypes
- terms of the form $\langle M, N\rangle_T$ of $\pi_1(M)$ are *expressions*, and correspond respectively to explicit coercions going from a type to some of its predicate subtypes and back
- terms of the form $\pi_2(M)$ are *proofs*, expressing the PVS-Core deduction rule SubtypeElim2.

As mentioned in the introduction, this formalism used to express predicate subtyping is very similar to the formalism of dependent pairs, used for instance in the type system ECC [16]. 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 $\equiv_{\beta\sigma}$. This conversion is defined from the usual reduction $\pi_i\langle M_1, M_2\rangle_T \triangleright_\sigma M_i$. In the following, we define the relations $\triangleright_{\beta\sigma}$, $\rightarrow_{\beta\sigma}$, $\twoheadrightarrow_{\beta\sigma}$, and $\equiv_{\beta\sigma}$ in the same way as done in the definition of $\equiv_{\beta *}$ in the definition of PVS-Cert.

Applied to well-typed terms, the conversion $\equiv_{\beta *}$ includes the more standard conversion $\equiv_{\beta\sigma}$ (this property is a direct consequence of Theorem 5 together with the Church-Rosser property of $\rightarrow_{\beta\sigma}$). However, this inclusion is strict: for instance, it is not difficult to find two well-typed terms $\langle M, N_1\rangle_T$ and $\langle M, N_2\rangle_T$ which are not convertible using $\equiv_{\beta\sigma}$, although they are convertible using $\equiv_{\beta *}$.

As a direct consequence of this property, PVS-Cert is an extension of the system obtained from it by replacing $\equiv_{\beta*}$ by $\equiv_{\beta\sigma}$, and this extension is strict. In this paper, this subsystem of will be referred to as PVS-Cert$^-$. It is a PTS with dependent pairs, and corresponds more precisely to the system obtained from the PTS λ-HOL by adding the single dependent pair rule $(Type, Prop, Type)$. It is strictly included in the type system ECC presented in [16].

An mentioned in the introduction, this choice of a strictly more flexible conversion allows to define a very simple translation from PVS-Core derivations to PVS-Cert derivable judgements. Indeed, using $\equiv_{\beta*}$ ensures that 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 7 and Theorem 11).

The reduction $\rightarrow_{\beta*}$ underlying conversion does not preserve typing: for instance, the judgement $x : Prop, h : x \vdash \langle x, h\rangle_T : T$ with $T = \{y : Prop \mid y\}$ is derivable, and $\langle x, h\rangle_T \rightarrow_{\beta*} x$, but $x : Prop, h : x \vdash x : T$ is not derivable. However, as presented in Section 6, the reduction $\rightarrow_{\beta\sigma}$ is type preserving, and will be used both as a definition of cut elimination for PVS-Cert proofs (Section 7) and in the definition of a type checking-algorithm (Section 8).

4 Properties of PVS-Cert

One of the most important properties satisfied by PVS-Cert if the Church-Rosser property.

Theorem 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 [14]), as rules are left-linear and non-overlapping. As proved in [14], such a system admits the Church-Rosser property.

In the case of PTSs, the Church-Rosser property of $\rightarrow_\beta$ is at the core of the type preservation of $\rightarrow_\beta$. In the case of PVS-Cert, the situation is different, as $\rightarrow_{\beta*}$ is not a type preserving reduction. However, in a first step, the Church-Rosser property of $\rightarrow_{\beta*}$ will be used to establish the expected stratification theorem, presented in Section 5. 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, $\rightarrow_{\beta\sigma}$, used both as a definition of cut elimination (Section 7) and at the core of the definition of a type-checking algorithm (Section 8).

Another important property of PVS-Cert used to design a type-checking algorithm is the uniqueness of types modulo conversion. As presented in Section 8, this property allows – together with the decidability of $\equiv_{\beta*}$ on well-typed terms – to reduce the problem of *type-checking* to a problem of *type inference.* This property also underlines the fact that that, even though PVS-Cert is designed to reflect predicate subtyping, it doesn't admit any subtyping itself. The proof of type uniqueness is standard, and does not involve any specific difficulty.

Theorem 2 (Uniqueness of types). *If two judgements $\Gamma \vdash M : T_0$ and $\Gamma \vdash M : T_1$ are derivable, then $T_0 \equiv_{\beta *} T_1$.*

PVS-Cert also satisfies several other standard properties expected from PTSs and PTSs extended with dependent pairs, among which thinning and substitution, described for instance in [4]), as well as context conversion, described for instance in [21], which is based on the extension of conversion to contexts. In these three cases, the corresponding proofs are straightforwardly adapted from the case of PTS.

We end this section with the following important theorem, which also holds in λ-HOL. The proof is adapted from the case of λ-HOL and does not involve any specific difficulty.

Theorem 3. *If $\Gamma \vdash M : T$ is derivable and $T \neq Kind$, there exists a sort s such that $\Gamma \vdash T : s$.*

5 Stratification in PVS-Cert

The stratification of terms in PVS-Cert reveals a strong link between PVS-Cert and PVS-Core (defined in Section 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 PTSs presented in [11] – in this paper, PTSs are referred to as GTSs, and 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 in some derivation, the two terms involved in the conversion 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 PTSs, these classes are specific classes of well-typed terms, and the stability under reduction follows from the type preservation of $\rightarrow_\beta$.

However, as mentioned in Section 3.3, type preservation does not hold for $\rightarrow_{\beta *}$ 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, even in the absence of type preservation for $\rightarrow_{\beta *}$, that most classes of stratified terms are stable by reduction with $\rightarrow_{\beta *}$.

We first present three classes of terms: **types**, **expressions**, and **proofs**. The expected property of stability by reduction will only be proved for types and expressions (Proposition 1), which is not problematic as the conversion rules are never directly applied to proofs in valid derivations.

Definition 1 (Variables stratification). *We introduce the notations:*

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

Definition 2 (Stratified terms). *We define stratified 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*** $p, q := h \mid \lambda h : P.p \mid \lambda x : A.p \mid p\ q \mid p\ t \mid \pi_2(t)$

Remark 2. As in the case of PVS-Core (Remark 1), there is no formal distinction between the notations t, u, P, and Q although, in the following, the notations of expressions P, Q will be preferred for expressions of type $Prop$.

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 term. 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 $\rightarrow_{\beta *}$, which is formalized in the following proposition – the proof does not involve any specific difficulty, as the definitions of types and expressions are designed to satisfy this property.

Proposition 1. *Whenever $M \rightarrow_{\beta *} N$ and M is a type (resp. an expression), so is N.*

Beyond its use in the proof of the stratification theorem (Theorem 4), this stability property is also directly useful in the proof of the strong normalization theorem for $\rightarrow_{\beta *}$ and $\rightarrow_{\beta\sigma}$, as briefly mentioned in Section 7.

Finally, we present the expected stratification theorem, based on the following definitions.

Definition 3 (Stratified contexts, stratified judgements). *We define*

- ***stratified contexts*** *as contexts in which all declarations have the form $X : Type$, $x : A$ (for some type A), or $h : P$ (for some expression P).*
- ***stratified judgements*** *as judgements of one of the following form, in which Γ is a stratified context:*

$$\begin{array}{ll} \Gamma \vdash WF & \Gamma \vdash Type : Kind \\ \Gamma \vdash A : Type & \Gamma \vdash t : A \\ \Gamma \vdash p : P & \end{array}$$

Theorem 4 (Stratification). *Any derivable judgement is stratified.*

Proof. The proof is straightforward by induction the derivation. In the case of CONVERSION, Proposition 1 and the Church-Rosser property of $\rightarrow_{\beta*}$ are used together to conclude that the two convertible terms are either both expressions, both types, both *Type*, or both *Kind*. Basic stability properties of types and expressions under substitution are also involved in the cases PROJ2 and APP. They are proved directly by induction.

6 A type preserving reduction

Contrary to the case of PTSs (resp. PTSs with dependent pairs), in which $\rightarrow_{\beta}$ (resp. $\rightarrow_{\beta\sigma}$) is a type preserving reduction, $\rightarrow_{\beta*}$ is not a type preserving reduction in PVS-Cert. Instead, we present in this section the type preservation of the reduction $\rightarrow_{\beta\sigma}$ in PVS-Cert. This reduction will be used both as a definition of cut elimination for PVS-Cert proofs (Section 7) and in the type-checking algorithm (Section 8).

The specificity of this proof of type preservation compared to similar results for PTSs lies in the fact that $M \rightarrow_{\beta\sigma} N$ does not imply $M \equiv_{\beta*} N$ in general. However, this implication always holds if M is either a type or an expression – the corresponding proof involves no particular difficulty.

Theorem 5. *Whenever $M \rightarrow_{\beta\sigma} N$ and M is a type (resp. an expression), so is N, and $M \equiv_{\beta*} N$.*

Finally, the type preservation theorem for $\rightarrow_{\beta\sigma}$ is the following.

Theorem 6. *Given a derivable judgement $\Gamma \vdash M : T$, and N such that $M \rightarrow_{\beta\sigma} N$, the judgement $\Gamma \vdash N : T$ is derivable.*

Proof. The proof is done by induction on the derivation. The situations where $M \not\triangleright_{\beta\sigma} N$ and the cases where $M \triangleright_{\beta\sigma} N$ are separated. We present here one case for each situation – the full proof can be found in the author's PhD dissertation [1].

- We illustrate the situation where $M \not\triangleright_{\beta\sigma} N$ with the case of the rule PROD, which involves Theorem 5. Discarding the notations of the original statement, we describe the last inference step with the following new notations:

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

  If the reduction occurs in U, we conclude directly by induction hypothesis. If the reduction hypothesis occurs in T, we write $T \rightarrow_{\beta\sigma} T'$. By induction hypothesis, $\Gamma \vdash T' : s_1$ is derivable. By the stratification theorem, $v \in \mathcal{V}_{s_1}$, hence $\Gamma, v : T' \vdash WF$ is derivable using the DECL rule. By the stratification theorem and Theorem 5, $T \equiv_{\beta*} T'$. Hence, using the second premise and context conversion (mentioned in Section 4), $\Gamma, v : T' \vdash U : s_2$ is derivable. Finally, using PROD, $\Gamma \vdash \Pi v : T'.U : s_3$ is derivable.

- We illustrate the situation where $M \triangleright_{\beta\sigma} N$ with the case of the rule Proj1. As M is a first projection and $M \triangleright_{\beta\sigma} N$, M is a σ-redex. We replace the notation M and T of the original statement by $\pi_1\langle M, N\rangle_T \triangleright_{\beta\sigma} M$ and T'. In this setting, the last inference step has the following form:

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

  Analyzing the derivation of the premise (and more precisely the last rule different from Conversion used in it, which is necessarily Pair), we conclude that 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 of the form

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

  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 the Church-Rosser property (Theorem 1). On the other hand, by the stratification theorem, $T' \neq \mathit{Kind}$, hence we can use Theorem 3 on the original conclusion to establish that $\Gamma \vdash T' : s$ is derivable from some sort s, as expected.

7 Strong normalization and cut elimination

This section is dedicated to the strong normalization of both $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta*}$ on well-typed PVS-Cert terms. These two reductions will be used separately in Section 8 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 type preserving reduction $\rightarrow_{\beta\sigma}$ will be used in the type-checking of applications. Moreover, the strong normalization of $\rightarrow_{\beta\sigma}$ combined with its type preservation property provides a cut elimination theorem, which is a powerful tool to study properties of both PVS-Cert and PVS-Core. Its use is illustrated in a proof of consistency of PVS-Cert (Theorem 9), used in turn to establish the consistency of PVS-Core (Theorem 12) at the end of this paper.

7.1 Strong normalization

A direct approach to prove the strong normalization of $\rightarrow_{\beta\sigma}$ 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 2. *There exists a well-typed term admitting an infinite reduction using $\rightarrow_{\beta\sigma*}$.*

Proof. We first define two well-typed terms M and N such that MN admits an infinite reduction. It is simple to find two such terms, using the fact that PVS-Cert is an extension of System F [12]. For instance:

- We take $\top = \Pi P : Prop.\Pi h : P.P$ together with $M = \lambda h : \top.h\ \top\ h$ and $N = \lambda h' : \top.\lambda h : \top.h\ \top\ h$
- M admits the type $\Pi h : \top.\top$ and N admits the type $\Pi h' : \top.\Pi h : \top.\top$.
- MN admits an infinite reduction $MN \rightarrow_{\beta\sigma*} N\ \top\ N \rightarrow_{\beta\sigma*} MN \rightarrow_{\beta\sigma*} ...$

Using these terms, we build the expected counter-example of normalization of $\rightarrow_{\beta\sigma*}$ as follows:

- We define $N' = \lambda P : Prop.\lambda h : P.h$, $T = \{x : Prop \mid \Pi h' : \top.\Pi h : \top.\top\}$, and $U = \{y : T \mid \top\}$.
- It is straightforward to show that $M\ \pi_2\langle\langle\top, N\rangle_T, N'\rangle_U$ admits the type $\top$.
- $M\ \pi_2\langle\langle\top, N\rangle_T, N'\rangle_U \twoheadrightarrow_{\beta\sigma*} MN$, hence it admits an infinite reduction.

Because of Proposition 2, we keep the expected strong normalization theorem in PVS-Cert formulated as follows.

Theorem 7 (Strong normalization). *For any derivable judgement $\Gamma \vdash M : T$, M is strongly normalizing under both $\rightarrow_{\beta\sigma}$ and $\rightarrow_{\beta*}$:*

- *any reduction sequence starting from M and using $\rightarrow_{\beta*}$ terminates*
- *any reduction sequence starting from M and using $\rightarrow_{\beta\sigma}$ terminates*

The proof of this theorem is left out of the scope of this paper. It is detailed in the author's PhD dissertation [1]. We simply highlight here some of its specificities, which illustrate the consequences of the choice, in PVS-Cert, of a conversion relation which is not based on a type-preserving reduction.

- The proof uses Tait's approach based on *saturated sets* (see for instance [23]). However, only one single notion of saturated set is used: saturated sets are defined here as specific subsets of the set of terms which are both strongly normalizing under $\rightarrow_{\beta\sigma}$ and strongly normalizing under $\rightarrow_{\beta*}$. As a consequence, compatibility properties for such saturated must be proved with respected to both reductions.
- Following Tait's approach, an interpretation function is defined in order to prove that, whenever term M admits a type T, it belongs to the interpretation of T, which is the main theorem established to conclude strong normalization. The definition of this function is inspired from the definitions of Girard in [12] for the strong normalization of F^ω – which corresponds to λ-HOL without type declarations –, but several ideas are also taken from [10], which presents, among other things, a proof of strong normalization of an extension of the calculus of constructions with dependent pairs.
- As the interpretation function is expected to be stable under $\rightarrow_{\beta*}$, its domain cannot be restricted to well-typed terms only, as well-typed terms are not stable under $\rightarrow_{\beta*}$. For this reason, it is chosen to define this interpretation function on the classes of types and expressions, as presented in the definition of stratified terms (Definition 3): indeed, this specific definition, which uses arbitrary terms instead of proof terms in the construction $\langle t, M\rangle_A$, is designed to ensure the stability of types and expressions under $\rightarrow_{\beta*}$.

7.2 Cut elimination in PVS-Cert

The following cut elimination theorem is a direct corollary of the strong normalization theorem and the type preservation of $\rightarrow_{\beta\sigma}$.

Theorem 8 (Cut elimination). *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 $\rightarrow_{\beta\sigma}$ to a normal form q such that the judgement $\Gamma \vdash q : P$ is derivable.*

Proof. By the strong normalization theorem, p can be reduced to a normal form q using the reduction $\rightarrow_{\beta\sigma}$. By the type preservation theorem (Theorem 6), the judgement $\Gamma \vdash q : P$ is derivable.

We conclude this section showing how the cut elimination theorem can be used together with the properties of terms in normal form with respect to $\rightarrow_{\beta\sigma}$ as a tool to analyze some meta-theoretical properties of PVS-Cert. As presented at the end of this work, this approach will also allow to use cut elimination in PVS-Cert to analyze some meta-theoretical properties of PVS-Core. This use of cut elimination is illustrated with the following proof of consistency.

Theorem 9. *PVS-Cert is consistent: there exists no proof term p such that $\vdash p : \Pi x : Prop.x$ is derivable.*

We use the following notion of *elimination context* in the proof :

Definition 4 (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

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

Proof (Theorem 9). 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 property (mentioned in Section 4), $x : Prop \vdash p : \Pi x : Prop.x$ 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 8, $\lambda x : Prop.(px)$ admits a normal form $\lambda x : Prop.q$ with respect to $\twoheadrightarrow_{\beta\sigma}$, which is such that the judgement $\vdash \lambda x : Prop.q : \Pi x : Prop.x$ is derivable.

Considering the last rule different from Conversion used in such a derivation (which is necessarily Lam), and using the stratification theorem, there exists a derivable judgement $x : Prop \vdash q : t$ for some expression $t \equiv_{\beta *} x$. Hence, using Conversion, $x : Prop \vdash q : x$ is also derivable. We consider D a possible derivation of this judgement.

As q is a proof and is in normal form with respect to $\twoheadrightarrow_{\beta\sigma}$, we conclude from a careful case analysis that q has one of the following forms: $\lambda v : T.M$ or $e[v]$. We discard the first possibility as follows. If $q = \lambda v : T.M$, considering the last rule different from Conversion used in D (which is necessarily Lam), there exists some term of the form $\Pi v' : T'.U'$ such that $\Pi v' : T'.U' \equiv_{\beta *} x$. By the Church-Rosser property (Theorem 1), this conversion cannot hold. As a consequence, q has the form $e[v]$ for some elimination context e and some variable v.

Considering the last rule different from Conversion, Proj1, Proj2, or App used in D (which is necessarily Var), some judgement of the form $x : Prop \vdash v : T$ is derivable, and $v = x$. As q is a proof, $e[x] = q \neq x$. Hence, D admits some subderivation of judgement of the form $x : Prop \vdash xt' : T'$ or $x : Prop \vdash \pi_i(x) : T'$. Considering the last rule different from Conversion in such a derivation, and using the uniqueness of types (Theorem 2), this implies that there exists a term U of the form $\Pi v' : T_1.T_2$ or $\{v' : T_1 \mid T_2\}$ such that $U \equiv_{\beta *} Prop$. By the Church-Rosser property (Theorem 1), this conversion cannot hold. As a consequence, there exists no proof term p such that the judgement $\vdash p : \Pi x : Prop.x$ is derivable.

8 Type-checking in PVS-Cert

The purpose of this section is to present the main ideas leading to the definition of a type-checking algorithm for PVS-Cert. The decidability of type-checking is one of the most important result expected for PVS-Cert. In particular, it will be used in Section 10 together with the translation from PVS-Core derivations to PVS-Cert established in Section 9 to show that PVS-Cert can be used as verifiable certificates for PVS-Core (Definition 8).

This algorithm is mainly based on the type preservation theorem 6 and the strong normalization theorem 7 presented in the previous sections. In this section, we will only focus on the main specificities of the algorithm. Its precise definition is presented in Appendix A, together with proof sketches of soundness, termination, and completeness, which don't involve any specific difficulty – a more detailed version can be found in the author's PhD dissertation [1].

The algorithm is comparable to the algorithm presented in [6] for the general case of injective PTSs (which applies to λ-HOL). Besides the fact that our algorithm is extended to handle predicate subtypes, coercions $\langle M, N\rangle_T$ and projections $\pi_i(M)$, the main difference between the two is the use of both reductions $\rightarrow_{\beta *}$ and $\rightarrow_{\beta\sigma}$ in the case of PVS-Cert, while only $\rightarrow_{\beta}$ is used for injective PTSs.

On the one hand, $\rightarrow_{\beta *}$-normalization is used to check $\equiv_{\beta *}$-conversion on well-typed terms: by the Church-Rosser property and strong normalization, two well-typed terms are $\equiv_{\beta *}$-equivalent if and only if they admit the same normal form, which is unique. As in [6], this decision procedure for conversion on well-typed terms is used in turn together with the uniqueness of types (Theorem 2) to define type-checking from type inference, which is itself defined recursively.

Remark 3. In order to avoid redundant context well-formedness verifications in the multiple recursive calls of the type inference algorithm, we choose here to check the well-formedness of a context Γ beforehand when inferring a type for some term M in Γ. For this reason, type inference and type-checking are defined in two steps. First, we define auxiliary type inference and type-checking algorithms which are only ensured to operate soundly with well-formed contexts. Then, we use these auxiliary functions to define context well-formedness verification as well as complete type inference and type-checking algorithms, which operate soundly with any context.

On the other hand, $\rightarrow_{\beta\sigma}$ is used in type inference to handle applications:

$$\frac{\mathit{\Gamma} \vdash M : \mathit{\Pi} v : T_1.T_2 \qquad \mathit{\Gamma} \vdash N : T_1}{\mathit{\Gamma} \vdash MN : T_2[N/v]}\ \textsc{App}$$

In this situation, the recursive call on the first premise may produce a term U such that $\mathit{\Gamma} \vdash M : U$ is derivable, but U is not ensured to have the form $\mathit{\Pi} v : U_1.U_2$ – counterexamples can be easily found when M is a proof and U is a proposition. The usual solution to this issue, used e.g. in [6], is to reduce U using the reduction underlying conversion (or more specifically its restriction to weak head reduction, which is more economic): indeed, using the uniqueness of types as well as strong normalization, type preservation, and the Church-Rosser property, it can be proved that a term U' will be obtained, that M admits the type U', and that U' has the form $\mathit{\Pi} v : U_1.U_2$ if M admits a type of this form.

However, in the case of PVS-Cert, this approach cannot be followed directly, as the reduction underlying conversion, which is $\rightarrow_{\beta*}$, is not type preserving: U' is not necessary a valid type for M. For this reason, we use instead the type preserving reduction $\rightarrow_{\beta\sigma}$ (again, we use more specifically its restriction to weak head reduction, which is more economic). Using the strong normalization theorem, this operation terminates and yields some term U''. As a direct corollary of type preservation (based on Theorem 3 and Theorem 5), M admits the type U''. What is left is to prove that U'' has the form $\mathit{\Pi} v : U_1.U_2$ if M admits a type of this form, which is done as follows. If M admits a type of the form $\mathit{\Pi} v : T_1.T_2$, then $U'' \equiv_{\beta*} \mathit{\Pi} v : T_1.T_2$ by the uniqueness of types. Hence, analyzing the possible forms of the weak head normal form U'' and using the Church-Rosser property, we conclude that U'' has the form $\mathit{\Pi} v : U_1.U_2$, as expected.

Compared to [6], new cases must be added for predicate subtypes, coercions $\langle M, N\rangle_T$, and projections $\pi_i(M)$. These cases are handled in a similar way as in the case of PTSs with dependent pairs (see for instance ECC [16]), and don't involve any specific difficulty. Instead, a more distinctive specificity of the algorithm lies in the case of λ-abstraction:

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

As in the case of injective PTSs studied in [6], 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.

A general solution for this issue, applicable to any injective PTSs, is presented in [6] using some classification of terms to avoid this unwanted recursive call. The solution selected for PVS-Cert follows the same approach, adapted to the stratified terms of PVS-Cert. It relies on a classifying algorithm $\textsc{Level}(\cdot)$, which ensures that whenever M is either an expression, a type, $Type$, or $Kind$, then $\textsc{Level}(M)$ is either 1, 2, 3, or 4 respectively. As it is specifically suited to PVS-Cert, this definition is simpler than the classification presented in [6], which is intended to be applicable to a wide family of type systems. The algorithm is defined as follows:

Definition 5. *We define the algorithm* $\textsc{Level}(\cdot)$ *by recursion on its argument. The possible cases are the following.*

- $\textsc{Level}(Kind) = 4$, $\textsc{Level}(Type) = 3$, $\textsc{Level}(Prop) = 2$
- $\textsc{Level}(\Pi v : T.U) = \textsc{Level}(U)$, $\textsc{Level}(\{v : T \mid U\}) = 2$, $\textsc{Level}(X) = 2$
- *In all other cases,* $\textsc{Level}(M) = 1$

9 Expressing PVS-Core in PVS-Cert

The final purpose of PVS-Cert is to encode PVS-Core derivations as PVS-Cert judgements, and to use the type-checking algorithm presented in Section 8 to use these judgements 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.

9.1 An erasing function from PVS-Cert to PVS-Core

We begin the description of this correspondence with a translation from PVS-Cert to PVS-Core, referred to as *erasing.* This translation mainly consists in the erasure of PVS-Cert explicit coercions $\langle \cdot, M \rangle_A$ and $\pi_1(\cdot)$.

Definition 6. *We define an erasure function* $[\![\cdot]\!]$ *from PVS-Cert expressions, types, and Type to PVS-Core terms recursively as follows.*

$$\begin{array}{lll}
[\![Type]\!] = Type & [\![x]\!] = x & [\![\langle t, M \rangle_A]\!] = [\![t]\!] \\
[\![Prop]\!] = Prop & [\![\lambda x : A.t]\!] = \lambda x : [\![A]\!].[\![t]\!] & [\![\pi_1(t)]\!] = [\![t]\!] \\
[\![X]\!] = X & [\![t\ u]\!] = [\![t]\!][\![u]\!] & \\
[\![\Pi x : A.B]\!] = \Pi x : [\![A]\!].[\![B]\!] & [\![\Pi x : A.P]\!] = \forall x : [\![A]\!].[\![P]\!] & \\
[\![\{x : A \mid P\}]\!] = \{x : [\![A]\!] \mid [\![P]\!]\} & [\![\Pi h : P.Q]\!] = [\![P]\!] \Rightarrow [\![Q]\!] &
\end{array}$$

Then, we extend straightforwardly $[\![\cdot]\!]$ *from PVS-Cert stratified contexts to PVS-Core contexts: for instance,* $[\![P, x : A, X : Type]\!] = [\![P]\!], x : [\![A]\!], X : Type$.

Last, we extend straightforwardly $[\![\cdot]\!]$ *from all PVS-Cert stratified judgements except those of the form* $\Gamma \vdash Type : Kind$ *to PVS-Core judgements. For instance,* $[\![x : A, X : Type \vdash p : P]\!] = x : [\![A]\!], X : Type \vdash [\![P]\!]$. *The PVS-Cert judgements of the form* $\Gamma \vdash Type : Kind$ *are not translated.*

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 10 that they are derivable in PVS-Core. This theorem relies in particular on the fact that conversion in PVS-Cert and PVS-Core are related through the erasure function $[\![\cdot]\!]$, established in the following proposition. The corresponding proof does not involve any specific difficulty.

Proposition 3. *For all terms M and N which are either expressions, types, or $Type$, whenever $M \equiv_{\beta*} N$, then $[\![M]\!] \equiv_{\beta} [\![N]\!]$.*

Using the two previous propositions and the stratification theorem in PVS-Cert, we conclude the following theorem, which allows to map PVS-Cert derivations to PVS-Core derivations.

Theorem 10. *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 proved by induction on the height of PVS-Cert derivations. All cases are straightforward, using the stratification theorem when necessary to establish a correspondence between stratified versions of PVS-Cert rules and PVS-Core rules. For instance:

- Decl corresponds either to TypeDecl, EltDecl, or Assumption
- Sort corresponds to Prop only (judgements of the form $\Gamma \vdash Type : Kind$ are not translated)
- Prod corresponds either to Pi, Forall, or Imply

9.2 Expressing PVS-Core derivations as PVS-Cert judgements

Theorem 10 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.

Definition 7. *For any PVS-Core derivation D, we define recursively the PVS-Cert stratified judgement* Certificate(D) *such that* $[\![\text{Certificate}(D)]\!]$ *corresponds to the conclusion of D.*

In this definition, we use an injective function $h(\cdot)$ mapping natural numbers to PVS-Cert proof variables, which can be chosen arbitrarily. We present two cases: Assumption*, which shows how $h(\cdot)$ is used, and* ImplyElim*. This latter case (as well as* ForallElim*) is more complex than others as it involves the computation of a normal form with respect to $\rhd_*$, i.e. the erasure of coercions at the head of a term. The other cases are detailed in Appendix B.*

- $\dfrac{\Gamma \vdash P : Prop}{\Gamma, P \vdash WF}$ Assumption

  We consider D_1 the derivation of $\Gamma \vdash P : Prop$. Certificate(D_1) *has the form $\Gamma_1 \vdash P_1 : Prop$. We consider n the number of declarations of the form $(h : Q)$ in Γ_1, and we define* Certificate$(D) = \Gamma_1, h(n) : P_1 \vdash WF$.

- $\dfrac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}$ ImplyElim

  We consider D_1 and D_2 the respective derivations of $\Gamma \vdash P \Rightarrow Q$ and $\Gamma \vdash P$. Certificate(D_2) *has the form $\Gamma_2 \vdash p_2 : P_2$ and* Certificate(D_1) *has the form $\Gamma_1 \vdash p_1 : Q_1'$. As $[\![Q_1']\!] = (P \Rightarrow Q)$, its normal form with respect to $\rhd_*$ has the form $\Pi h : P_1.Q_1$. We define* Certificate$(D) = \Gamma_1 \vdash p_1 p_2 : Q_1[p_2/h]$. *As all proof terms are deleted through the erasure function, $[\![Q_1[p_2/h]]\!] = [\![Q_1]\!]$. On the other hand, by induction hypothesis, $[\![Q_1]\!] = Q$, hence the erasure of this judgement is $\Gamma \vdash Q$, as expected.*

9.3 Relating conversion in PVS-Core and PVS-Cert

In order to prove that the outputs of the algorithm Certificate are derivable in PVS-Cert (presented in Theorem 11), the main required lemma is the fact that is the converse of Proposition 3: for any terms M and N which are either expressions, types, or *Type* and which verify $[\![M]\!] \equiv_\beta [\![N]\!]$, then $M \equiv_{\beta*} N$. More precisely, this property will be used in the proof Theorem 11 to handle the cases of conversion rules TypeConversion and PropConversion.

We first establish a modified version of this expected result, using equality and $\equiv_*$ instead of $\equiv_\beta$ and $\equiv_{\beta*}$ respectively. The proof is straightforward by induction on the two involved terms.

Proposition 4. *For all terms M and N which are either expressions, types, or Type, whenever $[\![M]\!] = [\![N]\!]$, then $M \equiv_* N$.*

Then, we establish the expected converse of Proposition 3 as follows.

Proposition 5. *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 present a proof based on the definition of a simple translation of PVS-Core terms as PVS-Cert expressions, types, or Type, which does not introduce any explicit coercion: for instance,

- $[\Pi x : A.B] = \Pi x : [A].[B]$
- $[P \Rightarrow Q] = \Pi h : [P].[Q]$ for an arbitrary proof variable h

We first show straightforwardly that the respective images through $[\cdot]$ of two terms related by $\equiv_\beta$ are also related by $\equiv_\beta$. As a consequence, $[[\![M]\!]] \equiv_\beta [[\![N]\!]]$.

On the other hand, it is straightforward to show that $[\cdot]$ is a right inverse of the erasure function $[\![\cdot]\!]$. Hence, $[\![[[\![M]\!]]]\!] = [\![M]\!]$. By Proposition 4, we conclude that $[[\![M]\!]] \equiv_* M$. Following the same reasoning, $[[\![N]\!]] \equiv_* N$.

As a consequence, $M \equiv_{\beta*} [[\![M]\!]] \equiv_{\beta*} [[\![N]\!]] \equiv_{\beta*} N$.

9.4 Soundness of the synthesis of certificates

The last proposition needed to prove the soundness of the algorithm CERTIFICATE is the following. It shows that the operation of normalization through $\rhd_*$ (which erases the coercions $\pi_1(\cdot)$ and $\langle \cdot, M\rangle_T$ at the head of a term) is safely used in the definition of CERTIFICATE.

Proposition 6. *For any derivable PVS-Cert judgement of the form $\Gamma \vdash t : \{x_n...\{x_1 : Prop \mid Q_1\}... \mid Q_n\}$, if t admits a normal form with respect to $\rhd_*$ which has the form $\Pi v : M.T$, then $\Gamma \vdash \Pi v : M.T : Prop$ is derivable.*

In fact, only the specific case $n = 0$ is used in the proof of soundness of CERTIFICATE, but this generalization is preferred as it admits a direct proof by induction on t, which does not involve any specific difficulty.

Last, we present the expected soundness property for CERTIFICATE:

Theorem 11. *For any PVS-Core derivation D,* CERTIFICATE(D) *is derivable in PVS-Cert.*

Proof. The proof is done by induction on D. Most cases are proved without any specific difficulty. In particular, the cases of conversion rules TYPECONVERSION and PROPCONVERSION are straightforward using Proposition 5.

The most complex cases correspond to the rules IMPLYELIM and FORALLELIM which involve, by definition of CERTIFICATE, some normalization with respect to $\rhd_*$. In such cases, Proposition 6 is used to handle the specific difficulties related to this normalization. We present the case IMPLYELIM:

$$\frac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}\ \textsc{ImplyElim}$$

We consider D_1 and D_2 the respective derivations of $\Gamma \vdash P \Rightarrow Q$ and $\Gamma \vdash P$. CERTIFICATE(D_2) has the form $\Gamma_2 \vdash p_2 : P_2$ and CERTIFICATE(D_1) has the form $\Gamma_1 \vdash p_1 : Q_1'$. As $[\![Q_1']\!] = (P \Rightarrow Q)$, its normal form with respect to $\rhd_*$ has the form $\Pi h : P_1.Q_1$. In this setting, 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 3 and the stratification theorem, $\Gamma_1 \vdash Q_1' : Prop$ is derivable in PVS-Cert. Hence, by Proposition 6, $\Gamma_1 \vdash \Pi h : P_1.Q_1 : Prop$ is derivable as well. As $Q_1' \equiv_{\beta *} \Pi h : P_1.Q_1$, we conclude applying the CONVERSION rule that $\Gamma_1 \vdash p_1 : \Pi h : P_1.Q_1$ is derivable.

On the other hand, using Proposition 4, we can conclude from $[\![\Gamma_1]\!] = \Gamma = [\![\Gamma_2]\!]$ that $\Gamma_1 \equiv_* \Gamma_2$ as long as both contexts admit the list of declared proof variables, in the same order. This is the case as, by straightforward induction on PVS-Core derivations, this list is $h(1), h(2), ..., h(n)$, where $h(\cdot)$ is the injective function used in the definition of CERTIFICATE and n is the number of proof variable declarations in Γ_1 and Γ_2. Hence, $\Gamma_1 \equiv_* \Gamma_2$.

As $\Gamma_1 \vdash p_1 : \Pi h : P_1.Q_1$ is derivable, by Theorem 3 and the stratification theorem, $\Gamma_1 \vdash \Pi h : P_1.Q_1 : Prop$ is derivable. Hence, considering the last rule different from CONVERSION used in such a derivation (which is necessarily

Prod), and using the stratification theorem, $\Gamma_1 \vdash P_1 : Prop$ is derivable as well. As a consequence, using context conversion (mentioned in Section 4), $\Gamma_1 \vdash p_2 : P_1$ 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.

10 Using PVS-Cert as a system of verifiable certificates for PVS-Core

This final section shows how to use the different results presented in this paper to answer to the main question addressed in the current work: defining a system of verifiable certificates for PVS-Core.

The type-checking algorithm for PVS-Cert presented in Section 8 is combined in the following way with the encoding of PVS-Core derivations into PVS-Cert presented in Section 9 to use PVS-Cert proof judgements as proof certificates for PVS-Core.

Definition 8 (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 $[\![\Gamma]\!] \vdash [\![P]\!]$ (Definition 6), which is verifiable using the type-checking algorithm presented in Section 8 (whose complete definition is given in Appendix A). On the one hand, this approach is sound: whenever the type-checking algorithm succeed, $\Gamma \vdash p : P$ is derivable in PVS-Cert, hence $[\![\Gamma]\!] \vdash [\![P]\!]$ is derivable in PVS-Core by Theorem 10.*

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, the PVS-Cert judgement Certificate(D) *can be used 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 11, $\Gamma \vdash p : P$ is derivable, hence the execution of the type-checking algorithm on this judgement succeeds: this certificate is valid.*

We finally show that, through the construction of certificates, the PVS-Cert cut elimination theorem can be used to study meta-theoretical properties of PVS-Core. This possible use is illustrated with the case of consistency, proved in PVS-Cert in Theorem 9 using cut elimination.

Theorem 12. *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 consider $\vdash p : P =$ Certificate(D). By definition, $[\![P]\!] = \forall x : Prop.x = [\![\Pi x : Prop.x]\!]$. Hence, by proposition 5, $P \equiv_{\beta *} \Pi x : Prop.x$. 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 9.

References

1. Frederic Gilbert *Extending higher-order logic with predicate subtyping: application to PVS*. PhD dissertation, Inria, 2018.
2. Andreas Abel and Gabriel Scherer. On irrelevance and algorithmic equality in predicative type theory. *arXiv preprint arXiv:1203.4716*, 2012.
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 barendregts cube. *Technica1 report, Carnegie-Me11on University (USA) and Universita di Torino (Ita1y)*, 1988.
8. Bruno Bernardo. *An implicit Calculus of Constructions with dependent sums and decidable type inference.* Phd thesis, École polytechnique, October 2015.
9. 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.
10. 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.
11. 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.
12. Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures de larithmétique dordre supérieur*. PhD thesis, PhD thesis, Université Paris VII, 1972.
13. Andrew M Kent, David Kempe, and Sam Tobin-Hochstadt. Occurrence typing modulo theories. In *ACM SIGPLAN Notices*, volume 51, pages 296–309. ACM, 2016.
14. Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1):279 – 308, 1993.
15. Kenneth Knowles and Cormac Flanagan. Hybrid type checking. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 32(2):6, 2010.
16. 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.
17. Sam Owre, John M Rushby, and Natarajan Shankar. Pvs: A prototype verification system. In *International Conference on Automated Deduction*, pages 748–752. Springer, 1992.
18. Sam Owre and Natarajan Shankar. The formal semantics of pvs. 1999.
19. Patrick M Rondon, Ming Kawaguci, and Ranjit Jhala. Liquid types. In *ACM SIGPLAN Notices*, volume 43, pages 159–169. ACM, 2008.

20. John Rushby, Sam Owre, and Natarajan Shankar. Subtypes for specifications: Predicate subtyping in pvs. *IEEE Transactions on Software Engineering*, 24(9):709–720, 1998.
21. Vincent Siles and Hugo Herbelin. Pure type system conversion is always typable. *Journal of Functional Programming*, 22(2):153–180, 2012.
22. Matthieu Sozeau. Subset coercions in coq. In *International Workshop on Types for Proofs and Programs*, pages 237–252. Springer, 2006.
23. William W Tait. A realizability interpretation of the theory of species. In *Logic Colloquium*, pages 240–251. Springer, 1975.
24. Jan Terlouw. Een nadere bewijstheoretische analyse van gstts. *Manuscript (in Dutch)*, 1989.
25. 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.
26. Jan Terlouw. Strong normalization in type systems: A model theoretical approach. *Annals of Pure and Applied Logic*, 73(1):53–78, 1995.
27. Benjamin Werner. On the strength of proof-irrelevant type theories. In *IJCAR*, volume 4130, pages 604–618. Springer, 2006.

A Appendix: full definition of the type-checking algorithm

As mentioned in Remark 3, the well-formedness of a context Γ is checked beforehand when inferring a type for some term M in the context Γ. For this reason, type inference and type-checking are defined in two steps. First, we define auxiliary type inference and type-checking algorithms which are only ensured to operate soundly with well-formed contexts. Then, we use these auxiliary functions to define complete type inference and type-checking algorithms, which operate soundly with any context.

These auxiliary functions are defined as follows.

Definition 9. *We define 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 weak head $\twoheadrightarrow_{\beta\sigma}$-reduction. 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 specifications expected from these auxiliary algorithms are the following.

Proposition 7. *The algorithms* Infer-type-aux *and* Check-type-aux *always terminate, and admit the following properties:*

- *Whenever $\Gamma \vdash WF$ is derivable,* Infer-type-aux($\Gamma \mid M$) *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*
- *Whenever $\Gamma \vdash T : s$ is derivable,* Check-type-aux($\Gamma \mid M \mid T$) *succeeds if and only if $\Gamma \vdash M : T$ is derivable*

Proof. The proof is decomposed in three successive step. The first one is the proof of soundness, which states that whenever $\Gamma \vdash WF$ (resp. $\Gamma \vdash T : s$) is derivable and Infer-type-aux($\Gamma \mid M$) outputs a term T successfully (resp. Check-type-aux($\Gamma \mid M \mid T$) succeeds), then $\Gamma \vdash M : T$ is derivable. In the case of λ-abstractions, we use the following classifying property of the algorithm Level($\cdot$) defined in Definition 5: whenever some term U is either an expression, a type, $Type$, or $Kind$, then Level(U) is either 1, 2, 3, or 4 respectively.

The second step is the proof of termination, which uses soundness in order to apply the expected strong normalization theorem. The final step is completeness, which states that whenever $\Gamma \vdash M : T$ is derivable (resp. $\Gamma \vdash T : s$ and $\Gamma \vdash M : T$ are derivable), then Infer-type-aux($\Gamma \mid M$) outputs some term successfully (resp. Check-type-aux($\Gamma \mid M \mid T$) succeeds). This final step uses in turn termination in order to ensure the success of the expected algorithms.

The proof of these three successive steps involves no further specific difficulty.

Last, complete algorithms of type-checking and type inference are defined as follows, together with well-formedness verification for contexts.

Definition 10. *We define the algorithms of well-formedness checking, type inference, and type checking using the two auxiliary algorithms of Definition 9. They are denoted* Check-wf(Γ)*,* Infer-type($\Gamma \mid M$)*, and* Check-type($\Gamma \mid M \mid T$) *respectively, 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 specifications expected from these algorithms are the following.

Theorem 13. *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. As in the proof of Proposition 7, we decompose the proof into a statement of soundness followed by a statement of termination and a statement of completeness. Using the results of proposition 7 at each step, the proof involves no specific difficulty.

B Appendix: full definition of the certificate synthesis algorithm

The full definition of the certificate synthesis algorithm Certificate(D) is the following.

Definition 11. *For any PVS-Core derivation D, we define recursively the PVS-Cert stratified judgement* $\textsc{Certificate}(D)$ *such that* $[\![\textsc{Certificate}(D)]\!]$ *corresponds to the conclusion of D.*

This definition involves some injective function $h(\cdot)$ mapping natural numbers to PVS-Cert proof variables, which can be chosen arbitrarily.

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 $\textsc{Certificate}(D) = \varnothing \vdash WF$.

- $\dfrac{\Gamma \vdash WF}{\Gamma, X : Type \vdash WF}$ TypeDecl $X \in \mathcal{V}_{types} \backslash DV(\Gamma)$

  We consider D_1 the direct subderivation of D. $\textsc{Certificate}(D_1)$ *has the form $\Gamma_1 \vdash WF$. We define* $\textsc{Certificate}(D) = \Gamma_1, X : Type \vdash WF$.

- $\dfrac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash WF}$ EltDecl $x \in \mathcal{V}_{expressions} \backslash DV(\Gamma)$

  We consider D_1 the direct subderivation of D. $\textsc{Certificate}(D_1)$ *has the form $\Gamma_1 \vdash A_1 : Type$. We define* $\textsc{Certificate}(D) = \Gamma_1, x : A_1 \vdash WF$.

- $\dfrac{\Gamma \vdash P : Prop}{\Gamma, P \vdash WF}$ Assumption

  We consider D_1 the direct subderivation of D. $\textsc{Certificate}(D_1)$ *has the form $\Gamma_1 \vdash P_1 : Prop$. We consider n the number of declarations of the form $(h : Q)$ in Γ_1, and we define* $\textsc{Certificate}(D) = \Gamma_1, h(n) : P_1 \vdash WF$.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash X : Type}$ TypeVar $(X : Type) \in \Gamma$

  We consider D_1 the direct subderivation of D. $\textsc{Certificate}(D_1)$ *has the form $\Gamma_1 \vdash WF$. We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash X : Type$.

- $\dfrac{\Gamma \vdash WF}{\Gamma \vdash Prop : Type}$ Prop

  We consider D_1 the direct subderivation of D. $\textsc{Certificate}(D_1)$ *has the form $\Gamma_1 \vdash WF$. We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash Prop : Type$.

- $\dfrac{\Gamma, x : A \vdash B : Type}{\Gamma \vdash \Pi x : A.B : Type}$ Pi

  We consider D_1 the direct subderivation of D. $\textsc{Certificate}(D_1)$ *has the form $\Gamma_1, x : A_1 \vdash B_1 : Type$. We define* $\textsc{Certificate}(D) = \Gamma_1 \vdash \Pi x :$

$A_1.B_1 : \mathit{Type}$.

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

  We consider D_1 the direct subderivation of D. Certificate(D_1) has the form $\Gamma_1, x : A_1 \vdash P_1 : \mathit{Prop}$. We define Certificate($D$) $= \Gamma_1 \vdash \{x : A_1 : P_1\} : \mathit{Type}$.

- $$\frac{\Gamma \vdash \mathit{WF}}{\Gamma \vdash x : A}\ \textsc{EltVar}\ (x : A) \in \Gamma$$

  We consider D_1 the direct subderivation of D. Certificate(D_1) has the form $\Gamma_1 \vdash \mathit{WF}$, 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 Certificate(D) $= \Gamma_1 \vdash x : A_1$.

- $$\frac{\Gamma, x : A \vdash P : \mathit{Prop}}{\Gamma \vdash \forall x : A.P : \mathit{Prop}}\ \textsc{Forall}$$

  We consider D_1 the direct subderivation of D. Certificate(D_1) has the form $\Gamma_1, x : A_1 \vdash P_1 : \mathit{Prop}$. We define Certificate($D$) $= \Gamma_1 \vdash \Pi x : A_1.P_1 : \mathit{Prop}$.

- $$\frac{\Gamma, P \vdash Q : \mathit{Prop}}{\Gamma \vdash P \Rightarrow Q : \mathit{Prop}}\ \textsc{Imply}$$

  We consider D_1 the direct subderivation of D. Certificate(D_1) has the form $\Gamma_1, h_1 : P_1 \vdash Q_1 : \mathit{Prop}$. We define Certificate($D$) $= \Gamma_1 \vdash \Pi h_1 : P_1.Q_1 : \mathit{Prop}$.

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

  We consider D_1 the direct subderivation of D. Certificate(D_1) has the form $\Gamma_1, x : A_1 \vdash t_1 : B_1$. We define Certificate($D$) $= \Gamma_1 \vdash \lambda x : A_1.t_1 : \Pi x : A_1.B_1$.

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

  We consider D_1 and D_2 the direct subderivations of D. Certificate(D_2) has the form $\Gamma_2 \vdash u_2 : A_2$ and Certificate(D_1) has the form $\Gamma_1 \vdash t_1 : \Pi x : A_1.B_1$. We define Certificate($D$) $= \Gamma_1 \vdash t_1 u_2 : B_1[u_2/x]$.
  By straightforward induction on B_1, $[\![B_1[u_2/x]]\!] = [\![B_1]\!][[\![u_2]\!]/x]$. On the other hand, by induction hypothesis, $[\![B_1]\!][[\![u_2]\!]/x] = B[u/x]$, hence the erasure of this judgement is $\Gamma \vdash tu : B[u/x]$, as expected.

- $$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash P[t/x] \qquad \Gamma \vdash \{x : A \mid P\} : \mathit{Type}}{\Gamma \vdash t : \{x : A \mid P\}}\ \textsc{SubtypeIntro}$$

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* Certificate($D_3$) *has the form* $\Gamma_3 \vdash \{x : A_3 \mid P_3\} : Type$. *We define* Certificate(D) $= \Gamma_1 \vdash \langle t_1, p_2 \rangle_{\{x:A_3 \mid P_3\}} : \{x : A_3 \mid P_3\}$.

- $\dfrac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash t : A}$ SubtypeElim1

  We consider D_1 *the direct subderivation of* D. Certificate(D_1) *has the form* $\Gamma_1 \vdash t_1 : \{x : A_1 \mid P_1\}$. *We define* 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* Certificate(D_1) *has the form* $\Gamma_1 \vdash p_1 : Q_1'$. *As* $[\![Q_1']\!] = (P \Rightarrow Q)$, *its normal form with respect to* $\rhd_*$ *has the form* $\Pi h : P_1.Q_1$. *We define* Certificate(D) $= \Gamma_1 \vdash p_1 p_2 : Q_1[p_2/h]$. *As all proof terms are deleted through the erasure function,* $[\![Q_1[p_2/h]]\!] = [\![Q_1]\!]$. *On the other hand, by induction hypothesis,* $[\![Q_1]\!] = Q$, *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* Certificate(D_1) *has the form* $\Gamma_1 \vdash p_1 : P_1'$. *As* $[\![P_1']\!] = (\forall x : A.P)$, *its normal form with respect to* $\rhd_*$ *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 straightforward induction on P_1, $[\![P_1[t_2/x]]\!] = [\![P_1]\!][[\![t_2]\!]/x]$. *On the other hand, by induction hypothesis,* $[\![P_1]\!][[\![t_2]\!]/x] = P[t/x]$, *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* $\Gamma_1 \vdash t_1 : \{x : A_1 \mid P_1\}$. *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* Certificate(D_1) *has the form* $\Gamma_2 \vdash Q_2 : Prop$. *We define* Certificate$(D) = \Gamma_1 \vdash p_1 : Q_2$.