Encoding of Predicate Subtyping with Proof Irrelevance in the λΠ-Calculus Modulo Theory

Gabriel Hondet, Frédéric Blanqui

Encoding of Predicate Subtyping with Proof Irrelevance in the λΠ-Calculus Modulo Theory

Gabriel Hondet
Université Paris-Saclay, ENS Paris-Saclay, CNRS, Inria, Laboratoire Méthodes Formelles, Gif-sur-Yvette, France

Frédéric Blanqui
Université Paris-Saclay, ENS Paris-Saclay, CNRS, Inria, Laboratoire Méthodes Formelles, Gif-sur-Yvette, France

Abstract

The λΠ-calculus modulo theory is a logical framework in which various logics and type systems can be encoded, thus helping the cross-verification and interoperability of proof systems based on those logics and type systems. In this paper, we show how to encode *predicate subtyping* and *proof irrelevance*, two important features of the PVS proof assistant. We prove that this encoding is correct and that encoded proofs can be mechanically checked by Dedukti, a type checker for the λΠ-calculus modulo theory using rewriting.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Higher order logic; Theory of computation → Equational logic and rewriting

Keywords and phrases Predicate Subtyping, Logical Framework, PVS, Dedukti, Proof Irrelevance

Digital Object Identifier 10.4230/LIPIcs.TYPES.2020.6

Acknowledgements The authors thank Gilles Dowek and the anonymous referees very much for their remarks.

1 Introduction

A substantial number of proof assistants can be used to develop formal proofs, but a proof developed in an assistant cannot, in general, be used in another one. This impermeability generates redundancy since theorems are likely to have one proof per proof assistant. It also prevents adoption of formal methods by industry because of the lack of standards and the difficulty to use adequately formal methods.

Logical frameworks are a part of the answer. Because of their expressiveness, different logics and proof systems can be stated in a common language. The λΠ-calculus modulo theory, or $\lambda\Pi/\equiv$, is such a logical framework. It is the simplest extension of simply typed λ-calculus with dependent types and arbitrary computation rules. Fixed-length vectors are a common example of dependent type, that can be represented in the λΠ-calculus as $\forall n : \mathbb{N}, \mathrm{Vec}(n)$. The λΠ-calculus modulo theory already allows to formulate first order logic, higher order logic [5] or proof systems based on *Pure Type Systems* [12] such as Matita [3], Coq [10] or Agda [16].

PVS [28] is a proof assistant that has successfully been used in collaboration by academics and industrials to formalise and specify real world systems [27]. More precisely, PVS is an environment comprising a specification language, a type checker and a theorem prover. One of the specificities of PVS is its ability to blend type checking with theorem proving by requiring terms to validate arbitrary predicates in order to be attributed a certain type. This ability is a consequence of *predicate subtyping* [30]. It facilitates the development of specifications and provides a more expressive type system which allows to encode more constraints. For instance, one can define the inverse function $\mathrm{inv} : \mathbb{R}^* \to \mathbb{R}$, where $\mathbb{R}^*$ is a predicate subtype defined as reals which are not zero.

If predicate subtyping provides a richer type system, it also makes type checking of specifications undecidable. In [17], F. Gilbert paved the way of the expression of PVS into $\lambda\Pi/\equiv$: he formalised the core of PVS and provided a language of certificates for PVS whose type checking is decidable. However, the encoding in $\lambda\Pi/\equiv$ of this language of certificates relies on *proof irrelevance*.

The following work proposes an encoding of proof irrelevant equivalences into the $\lambda\Pi$-calculus modulo theory. It also inspects the completion of such equations into a confluent rewrite system. The resulting rewrite system can be used to provide an encoding of PVS into Dedukti, a type-checker for the $\lambda\Pi$-calculus modulo theory based on rewriting [4].

Related work

An encoding or "simulation" of predicate subtyping *à la* PVS into HOL can be found in [20]. The objective of that work was to get some facilities provided by predicate subtyping into HOL rather than providing a language of certificates, and proof checking hence remains undecidable. Moreover, predicate subtypes are not represented by types but by theorems.

In [32], predicate subtyping is weakened into a language named Russell to be then converted into CIC. This conversion amounts to the insertion of coercions and unsolved meta-variables, the latter embody PVS *type correctness conditions* (TCC). The equational theory used in the CIC encoding is richer than ours since it includes surjective pairing $e = \mathrm{pair}\, T\ U\, (\mathrm{fst}\, T\, U\, e)\, (\mathrm{snd}\, T\, U\, e)$ and η-equivalence $f = \lambda x,\ f\ x$ in addition to proof irrelevance.

In [36], proof irrelevance is embedded into Luo's ECC [25] and its dependent pairs. Pairs and dependent pair types come in two flavours, the proof irrelevant one and the normal one. The flavour is noted by an annotation, and proof irrelevance is implemented by a reduction which applies only on annotated pairs. The article presents as well an application to pvs.

On a slightly more practical side, the automated first-order prover ACL2 [21] reproduces the system of "guards" provided by predicate subtyping into its logic based on Common Lisp with the concept of *gold symbols*. Approximately, a symbol is gold if all its TCC have been solved.

Some theories – often based on Martin-Löf's Type Theory – blend together a decidable (called *definitional* or *intensional*) equality with an undecidable (said *extensional*) equality. In [29], a judgement "A is provable" is introduced, to say that a proof of A exists, but no attention is paid to what it is. Similarly, [1] introduces proof irrelevance in Martin-Löf's logical framework using a function to distinguish propositions A from "proof-irrelevant propositions $\mathsf{Prf}(A)$". While A can be inhabited by several normal terms, $\mathsf{Prf}(A)$ is inhabited by only one normal form noted $\star$, to which all terms of $\mathsf{Prf}(A)$ reduce. Still in Martin-Löf's type theory, [31] provides proof irrelevance for predicate subtyping (here called *subset types*) for two different presentations, one is intensional, and the other extensional. The interested reader may have a look at Nuprl [11], an implementation of Martin-Löf's Type Theory with extensional equality and subset types.

Proof irrelevance has also been added to LF to provide a new system LFI in [24], where proof irrelevance is used in the context of *refinement types*. In LFI, proof irrelevance is not limited to propositions, nor it is attached to a certain type: terms are irrelevant based on the function they are applied to. A similar system is implemented in Agda [33].

More generally, concerning proof irrelevance in proof assistants, Coq and Agda [18] each have a sort for proof irrelevant propositions (`SProp` for Coq and `Prop` for Agda [33]). Lean [14] is by design proof irrelevant, and Matita supports proof irrelevance as well [2, Section 9.3].

Outline

Encoding predicate subtyping requires a clear definition of it, which is done in Section 2. Predicate subtyping is encoded into $\lambda\Pi/\equiv$ using the signatures provided in Section 3. This encoding is put in use into some examples as well. The encoding is proved correct in Section 4: any well typed term of the source language can be encoded into $\lambda\Pi/\equiv$, and its type in $\lambda\Pi/\equiv$ is the encoding of its type in the source language. Finally, we show that a type checker for the $\lambda\Pi$-calculus modulo rewriting can be used to type check terms that have been encoded as described in Section 3.

2 PVS-Cert: A Minimal System With Predicate Subtyping

Because of its size, encoding the whole of PVS cannot be achieved in one step. Consequently, F. Gilbert in his PhD [17] extracted, formalised and studied a subsystem of PVS which captures the essence of predicate subtyping named PVS-Cert. Unlike PVS, PVS-Cert contains proof terms, which has for consequence that type checking is decidable in PVS-Cert while it is not in PVS. Hence PVS-Cert is a good candidate to be a logical system in which PVS proofs and specifications can be encoded to be rechecked by external tools.

In this paper, we use an equational presentation of PVS-Cert, that is, we use equations rather than reduction rules and slightly change the syntax of terms. We describe PVS-Cert, as done in [17], namely the addition of predicate subtyping over simple type theory.

2.1 Type Systems Modulo Theory

To describe PVS-Cert and $\lambda\Pi/\equiv$ in a uniform way, we will use the notion of *Type Systems Modulo* described in [8]. Type Systems Modulo are an extension of *Pure Type Systems* [7] with symbols of fixed arity whose types are given by a *typing signature* Σ, and an arbitrary conversion relation $\equiv$ instead of just β-conversion $\equiv_\beta$.

The terms of such a system are characterised by a finite set of *sorts* $\mathcal{S}$, a countably infinite set of variables $\mathcal{V}$ and a signature Σ. The set of terms $\mathcal{T}(\Sigma, \mathcal{S}, \mathcal{V})$ is inductively defined in Figure 1.

$$M, N, T, U ::= s \in \mathcal{S} \mid x \in \mathcal{V} \mid M\ N \mid \lambda x : T,\ M \mid (x : T) \to U \mid f(\overrightarrow{M})$$
$$\text{with } \Sigma(f) = \left(\overrightarrow{x, T}, U, s\right)$$

Figure 1 Terms of the type system characterised by $\mathcal{S}, \mathcal{V}$ and Σ.

The contexts are noted $\Gamma ::= \varnothing \mid \Gamma, v : T$ and the judgements $\Gamma \vdash WF$ or $\Gamma \vdash M : T$. The typing rules are given in Figure 2 and depend on

- *axioms* $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ to type sorts;
- *product rules* $\mathcal{P} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ to type dependent products;
- a typing signature Σ which defines the function symbols and how to type their applications;
- a convertibility relation $\equiv$.

Notations. Rewriting relations are noted $\hookrightarrow_R$, where R is a set of rewriting rules. $\hookrightarrow_R$ is the closure of R by substitution and context. $\equiv_R$ is the symmetric, reflexive and transitive closure of $\hookrightarrow_R$. The substitution of x by N in M is noted $\{x \mapsto N\}\, M$. We use a vectorised notation

$$\text{empty}\,\frac{}{\varnothing \vdash WF} \qquad \text{decl}\,\frac{\Gamma \vdash T : s}{\Gamma, v : T \vdash WF}\, v \notin \Gamma \qquad \text{var}\,\frac{\Gamma \vdash WF}{\Gamma \vdash v : T}\, v : T \in \Gamma$$

$$\text{conv}\,\frac{\Gamma \vdash M : U \quad \Gamma \vdash T : s \quad T \equiv U}{\Gamma \vdash M : T} \qquad \text{sort}\,\frac{\Gamma \vdash WF}{\Gamma \vdash s_1 : s_2}\,(s_1, s_2) \in \mathcal{A}$$

$$\text{prod}\,\frac{\Gamma \vdash T : s_1 \quad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash (x : T) \to U : s_3}\,(s_1, s_2, s_3) \in \mathcal{P}$$

$$\text{abst}\,\frac{\Gamma \vdash (x : T) \to U : s \quad \Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x : T,\, M : (x : T) \to U} \qquad \text{app}\,\frac{\Gamma \vdash M : (x : T) \to U \quad \Gamma \vdash N : T}{\Gamma \vdash M\ N : \{x \mapsto N\}\, U}$$

$$\text{sig}\,\frac{\overrightarrow{x : T} \vdash U : s \quad \left(\Gamma \vdash t_i : \left\{(x_j \mapsto t_j)_{j<i}\right\} T_i\right)_i}{\Gamma \vdash f(\vec{t}) : \left\{\overrightarrow{x \mapsto t}\right\} U}\,\Sigma(f) = \left(\overrightarrow{x, T}, U, s\right)$$

Figure 2 Typing rules of a Type System Modulo.

for products $(\overrightarrow{x : T}) \to U$ to represent the dependent product $(x_1 : T_1) \to (x_2 : T_2) \to \cdots (x_n : T_n) \to U$; and more generally for any construction that can be extended to a finite sequence, such as a parallel substitution $\left\{\overrightarrow{x \mapsto N}\right\} M$. A mapping $\Sigma(f) = (\overrightarrow{x : T}, U, s)$ can also be written $\overrightarrow{x : T} \vdash_\Sigma f(\vec{x}) : U : s$. For all relations on terms R and S, we write $RS = \{(t, u) \mid \exists v, tRv \wedge vSu\}$ the composition of R and S, and R^* the reflexive and transitive closure of R.

2.2 Simple Type Theory

PVS and PVS-Cert are both based on simple type theory, which can be represented by the PTS λHOL [7]:

- $\mathcal{S}^{\lambda\text{HOL}} = \{Prop, Type, Kind\}$,
- $\mathcal{A}^{\lambda\text{HOL}} = \{(Prop, Type), (Type, Kind)\}$,
- $\mathcal{P}^{\lambda\text{HOL}} = \{(Prop, Prop, Prop), (Type, Type, Type), (Type, Prop, Prop)\}$,
- $\Sigma^{\lambda\text{HOL}} = \emptyset$,
- $\equiv^{\lambda\text{HOL}}$ is the reflexive, transitive and symmetric closure of the β-equation

$$((\lambda x,\, M)\ N) = \{x \mapsto N\}\, M \qquad (\beta)$$

2.3 Predicate Subtyping

Predicate subtyping has two main benefits for a specification language. The first is to provide a richer type system thanks to the entanglement of type-checking and proof-checking. In consequence, any property can by encoded in the type system, which allows to easily create "guards" such as `tail : nonempty_stack → stack` where `nonempty_stack` is a predicate subtype defined from a predicate `empty?`. It is also essential in the expression of mathematics: the judgement $M : T$ is akin to the statement $M \in T$ in the usual language of mathematics when T is a set defined by comprehension such as $E = \{n : \mathbb{N} \mid P(n)\}$. With predicate subtyping, we can represent the set E by the type $(\text{psub}\,\mathbb{N}\ P)$, and the judgement $\Gamma \vdash M : \text{psub}\,\mathbb{N}\ P$ is derivable if term M contains a proof of $P(n)$ for some

n. The other benefit of predicate subtyping, which is essential in PVS developments, is that it separates the process of writing specifications from the proving phase. In PVS, this separation appears through *type correctness conditions* (TCC): the development of specifications creates proof obligations that may be solved at any time. This separation is also visible in usual mathematical developments, where if we want to prove that $t \in E$, we prove once that $P(t)$ is valid to then forget the proof and simply use t.

The type system of PVS-Cert can be seen as λHOL with a non empty signature Σ^{PVS} defined in Figure 3 and a richer equivalence $\underset{\text{PVS}}{\equiv}$ that will be discussed in the next paragraph.

$$T : \mathit{Type}, p : T \to \mathit{Prop} \vdash \text{psub}\, T\, p \quad : \mathit{Type} \quad : \mathit{Kind} \tag{1}$$

$$T : \mathit{Type}, p : T \to \mathit{Prop}, m : T, h : p\, m \vdash \text{pair}\, T\, p\, m\, h : \text{psub}\, T\, p \quad : \mathit{Type} \tag{2}$$

$$T : \mathit{Type}, p : T \to \mathit{Prop}, m : \text{psub}\, T\, p \vdash \text{fst}\, T\, p\, m \quad : T \quad : \mathit{Type} \tag{3}$$

$$T : \mathit{Type}, p : T \to \mathit{Prop}, m : \text{psub}\, T\, p \vdash \text{snd}\, T\, p\, m \quad : p\, (\text{fst}\, T\, p\, m) \quad : \mathit{Type} \tag{4}$$

Figure 3 Signature Σ^{PVS} of PVS-Cert.

A predicate subtype $(\text{psub}\, T\, U)$ is defined from a *supertype* T and predicate U which binds a variable of type T to a proposition. Terms inhabiting a predicate subtype $(\text{psub}\, T\, U)$ are built with the pair construction $(\text{pair}\, T\, U\, M\, N)$ where M is a term of the supertype T and N is a proof of $(U\, M)$. While the pair construction allows to coerce a term from any type to a predicate subtype, the converse, that is the coercion from a type to its supertype is done with fst, the left projection of the pair. The right projection, snd, provides a witness that the left projection of the pair validates the predicate defining the subtype. Unlike PVS-Cert, PVS does not use coercions pair, fst and snd. In PVS, subtyping is implicit: terms do not have a unique type, and the choice of this type is left to the type checker.

▶ Remark 1. Unlike the original presentation of PVS-Cert in [17], this one annotates fst and snd, using $\text{fst}\, T\, p\, m$ instead of $\text{fst}\, m$ to ease the well-definedness proof of the translation of PVS-Cert terms (Proposition 4).

Equations and Proof Irrelevant Pairs

So far, no real difference has been evinced between PVS-Cert and dependent pairs: predicate subtype $(\text{psub}\, T\, p)$ may be encoded as the dependent pair type $\Sigma x : T, p\, x$ [17, Definition 4.2.3]. The difference lies in the equivalence relations and the fact that PVS-Cert implements *proof irrelevance* in pairs.

The equivalence of PVS-Cert is noted $\underset{\text{PVS}}{\equiv}$ and contains Equations (5), (6), and (β) which provide *proof irrelevance*:

$$\text{pair}\, t\, u\, m\, h_0 = \text{pair}\, t\, u\, m\, h_1 \tag{5}$$

$$\text{fst}\, t_0\, u_0\, (\text{pair}\, t_1\, u_1\, m\, h) = m \tag{6}$$

We will now motivate the use of these equations in PVS-Cert. Proofs contained in terms are essential for typing purposes. On the other hand, these proofs are a burden regarding equivalence of terms. Were these proofs taken into account (as $\equiv_\beta$ does), too many terms would be distinguished. For example, consider two terms $t = \text{pair}\, \mathbb{N}\, \mathit{Even}\, 2\, h$ and $t' = \text{pair}\, \mathbb{N}\, \mathit{Even}\, 2\, h'$ typed as even numbers. Then t and t' are not considered equal because they don't have the same proof (h and h') that 2 is even. We end up with one even number 2 per proof that 2 is even.

As stated in [13], most mathematicians seek convertibility of t and t' and care more about what h and h' prove than the proofs themselves. To this end, PVS-Cert has *proof irrelevant* pairs: proofs attached to terms are not taken into account when checking the equivalence of two pairs. This property is embedded in the equivalence relation $\underset{\text{PVS}}{\equiv}$ used in the conversion rule of PVS-Cert which must verify Equation (5).

Equation (6) allows the projection to compute, but because of proof irrelevance, we cannot allow the right projection to compute, otherwise, all terms of type *Prop* would be considered equivalent.

A proof of $T \equiv_\beta U$ or $T \underset{\text{PVS}}{\equiv} U$ can use untyped intermediate terms, which can be problematic when one wants to prove some property on typed terms only. In the case of $\equiv_\beta$, the problem is solved by using the fact that $\hookrightarrow_\beta$ is confluent, that is $\equiv_\beta = \hookrightarrow_\beta^* \hookleftarrow_\beta^*$. We now prove a similar property for $\underset{\text{PVS}}{\equiv}$:

▶ **Lemma 2** (Properties of the PVS-Cert conversion). *Let* $\hookrightarrow_{\beta\text{fst}} = \hookrightarrow_\beta \cup \hookrightarrow_{\text{fst}}$ *where* $\hookrightarrow_{\text{fst}}$ *is the closure by substitution and context of Equation* (6) *oriented from left to right, and let* $\leftrightarrow_{pi}$ *be the closure by substitution and context of Equation* (5) *and* $=_{pi} = \leftrightarrow_{pi}^*$.

For all relation on terms R, *let* R^{ty} *be the restriction of* R *to typable terms. Then:*

- $\underset{\text{PVS}}{\equiv} \subseteq \hookrightarrow_{\beta\text{fst}}^* =_{pi} \hookleftarrow_{\beta\text{fst}}^*$
- $\hookrightarrow_{\beta\text{fst}}$ *preserves typing: if* $\Gamma \vdash_{\text{PVS}} M : T$ *and* $M \hookrightarrow_{\beta\text{fst}} M'$, *then* $\Gamma \vdash_{\text{PVS}} M' : T$
- $\underset{\text{PVS}}{\equiv}^{ty} \subseteq \left(\hookrightarrow_{\beta\text{fst}}^{ty}\right)^* \left(\leftrightarrow_{pi}^{ty}\right)^* \left(\hookleftarrow_{\beta\text{fst}}^{ty}\right)^*$,

Proof. A relation $\hookrightarrow$ is confluent modulo some relation E if $\hookleftarrow^* \hookrightarrow^* \subseteq \hookrightarrow^* E \hookleftarrow^*$. If $E = \emptyset$, we simply say that $\hookrightarrow$ is confluent.

First note that $\hookrightarrow_{\beta\text{fst}}$ is confluent since it can be seen as a Combinatory Reduction System that is orthogonal (i.e. whose rules are left-linear and non-overlapping) [22].

We now prove that $\leftrightarrow_{pi}$ steps can be postponed: $\leftrightarrow_{pi} \hookrightarrow_{\beta\text{fst}} \subseteq \hookrightarrow_{\beta\text{fst}}^{=} =_{pi}$, where $\hookrightarrow_{\beta\text{fst}}^{=}$ is the reflexive closure of $\hookrightarrow_{\beta\text{fst}}$. Assume that the $\leftrightarrow_{pi}$ step is at position p and the $\hookrightarrow_{\beta\text{fst}}$ step is at position q. If p and q are disjoint, this is immediate. If p is above q, we have $\text{pair}\, T\ U\, M\, N_1 \leftrightarrow_{pi} \text{pair}\, T\ U\, M\, N_2$ and either $\text{pair}\, T\ U\, M\, N_2 \hookrightarrow_{\text{fst}} M$ or $\text{pair}\, T\ U\, M\, N_2 \hookrightarrow_{\beta\text{fst}} \text{pair}\, T'\ U'\, M'\, N_2'$. In the first case, $\text{pair}\, T\ U\, M\, N_1 \hookrightarrow_{\text{fst}} M$. In the second case, $\text{pair}\, T\ U\, M\, N_1 \hookrightarrow_{\beta\text{fst}}^{=} \text{pair}\, T'\ U'\, M'\, N_1 \leftrightarrow_{pi} \text{pair}\, T'\ U'\, M'\, N_2'$. Finally, if q is above p, we have $(\lambda x : T,\, M)N \leftrightarrow_{pi} (\lambda x : T',\, M')N' \hookrightarrow_{\beta\text{fst}} \{x \mapsto N'\}\, M'$ and $(\lambda x : T,\, M)N \hookrightarrow_{\beta\text{fst}} \{x \mapsto N\}\, M =_{pi} \{x \mapsto N'\}\, M'$, and similarly in the case of a fst step.

Hence, (1) $\hookrightarrow_{\beta\text{fst}}$ is confluent modulo $=_{pi}$, that is, $\underset{\text{PVS}}{\equiv} \subseteq \hookrightarrow_{\beta\text{fst}}^* =_{pi} \hookleftarrow_{\beta\text{fst}}^*$.

We now prove that (2) $\hookrightarrow_\beta$ preserves typing. To this end, it suffices to prove that, if $(x : T) \to U$ and $(x : T') \to U'$ are typable, and $(x : T) \to U \underset{\text{PVS}}{\equiv} (x : T') \to U'$, then $T \underset{\text{PVS}}{\equiv} T'$ and $U \underset{\text{PVS}}{\equiv} U'$ (see [9] for more details), which follows from (1).

We now prove that (3) $\hookrightarrow_{\text{fst}}$ preserves typing. Assume that $\text{fst}\, T_0\ P_0\, (\text{pair}\, T_1\ P_1\, M\, N)$ is of type C. By inversion of typing rules, $\text{pair}\, T_1\ P_1\, M\, N$ is of type $\text{psub}\, T_0\ P_0$ and $T_0 \underset{\text{PVS}}{\equiv} C$. By inversion again, M is of type T_1 and $\text{psub}\, T_0\ U_0 \underset{\text{PVS}}{\equiv} \text{psub}\, T_1\ P_1$. By (1), $T_0 \underset{\text{PVS}}{\equiv} T_1$ and $P_0 \underset{\text{PVS}}{\equiv} P_1$. Therefore, M is of type C.

Next, note that (4) $=_{pi} = \Leftrightarrow_{pi}$ where $\Leftrightarrow_{pi}$ consists in applying several $\leftrightarrow_{pi}$ steps at disjoint positions. Indeed, if $t = \text{pair}\, T\ P\, M\, N_1 \leftrightarrow_{pi} u = \text{pair}\, T\ P\, M\, (\ldots(\text{pair}\, T'\ P'\, M'\, N_1')\ldots) \leftrightarrow_{pi} v = \text{pair}\, T\ P\, M\, (\ldots(\text{pair}\, T'\ P'\, M'\, N_2')\ldots)$, then $t \leftrightarrow_{pi} v$ as well.

Moreover, we have (5) $\Leftrightarrow_{pi}^{ty} = (\leftrightarrow_{pi}^{ty})^*$. Indeed, $A \Leftrightarrow_{pi}^{ty} B$ means that we can obtain B from A by replacing some subterms of A, that are typable since A is typable, by some subterms of B, that are typable since B is typable.

We can now conclude as follows. Assume that $A \underset{\text{PVS}}{\equiv}^{ty} B$. By (1), there are A' and B' such that $A \hookrightarrow^*_{\beta\text{fst}} A' =_{pi} B' \hookleftarrow^*_{\beta\text{fst}} B$. By (2), (3), (4) and (5), $A(\hookrightarrow^{ty}_{\beta\text{fst}})^* A'(\leftrightarrow^{ty}_{pi})^* B'(\hookleftarrow^{ty}_{\beta\text{fst}})^* B$. ◀

3 Encoding PVS-Cert in $\lambda\Pi/\equiv$

We provide an encoding of PVS-Cert into the logical framework $\lambda\Pi/\equiv$. This encoding allows to express terms of PVS-Cert into $\lambda\Pi/\equiv$. Because logical frameworks strive to remain minimal, constructions such as pair or psub are not built-in: they must be expressed into the language of the logical framework through an encoding. We hence define the symbols allowing to emulate predicate subtyping using the terms of $\lambda\Pi/\equiv$.

Definition of $\lambda\Pi/\equiv$

$\lambda\Pi/\equiv$ is the family of Type Systems Modulo whose sorts, axioms and product rules are:

- sorts $\mathcal{S}^{\lambda\Pi} = \{\texttt{TYPE}, \texttt{KIND}\}$,
- axiom $\mathcal{A}^{\lambda\Pi} = \{(\texttt{TYPE}, \texttt{KIND})\}$,
- product rules $\mathcal{P}^{\lambda\Pi} = \{(\texttt{TYPE}, \texttt{TYPE}, \texttt{TYPE}), (\texttt{TYPE}, \texttt{KIND}, \texttt{KIND})\}$.

3.1 Encoding Simple Type Theory

The encoding of λHOL given in Figures 4 and 5 follows the method settled in [12] for pure type systems.

In the following, we write the function symbols of a signature in blue and the other constructions of $\lambda\Pi/\equiv$ in black, to better distinguish them.

The general idea is to manipulate types and terms of λHOL as terms of $\lambda\Pi/\equiv$. Sorts are both objectified as type and prop and encoded as types by Kind, Type and Prop in Equations (7)–(11). Sorts as types are used to type sorts as objects to encode the axioms in $\mathcal{A}$. Terms of type $Type$ are encoded as terms of type Type. These encoded types can be interpreted as $\lambda\Pi/\equiv$ types with function El (12). Similarly, propositions are reified as terms of type prop and interpreted by function Prf. For instance, given a λHOL type T and a λHOL proposition P both encoded as $\lambda\Pi/\equiv$ terms, the abstractions $\lambda x : \text{El}\, T$, x and $\lambda h : \text{Prf}\, P$, h are valid $\lambda\Pi/\equiv$ terms. The signature exposed in Figure 4 is noted $\Sigma^{\lambda\text{HOL}}$.

Equations (18)–(20) are used to map encoded products to $\lambda\Pi/\equiv$ products. Equation (17) makes sure that the objectified sort prop is the same as the sort Prop when interpreted as a type.

3.2 Encoding Predicate Subtyping

Predicate subtypes are defined in Equation (21) as encoded types (i.e. terms of type Type) built from encoded type t and predicate defined on t. Pairs are encoded in Equation (22), where the second argument is the predicate that defines the type of the pair. The two projections are encoded in Equations (23) and (24), and we note the signature of Figure 6 Σ^{psub}.

The signature used to encode PVS-Cert into $\lambda\Pi/\equiv$ is $\Sigma^{\text{PC}} = \Sigma^{\lambda\text{HOL}} \cup \Sigma^{\text{psub}}$. The terms of the encoding are thus the terms of $\mathcal{T}(\Sigma^{\text{PC}}, \mathcal{S}^{\lambda\Pi}, \mathcal{V})$. The typing rules are those of $\lambda\Pi/\equiv$ with the signature Σ^{PC} and the congruence $\underset{\lambda\Pi}{\equiv}$ generated by Equations (5), (6), (17)–(20), and (β) where, in Equations (5) and (6), psub, pair and fst (PVS-Cert symbols in black) are replaced by psub, pair and fst ($\lambda\Pi/\equiv$ symbols in blue).

$$\vdash \mathrm{Kind} : \mathtt{TYPE} : \mathtt{KIND} \tag{7}$$
$$\vdash \mathrm{Type} : \mathtt{TYPE} : \mathtt{KIND} \tag{8}$$
$$\vdash \mathrm{Prop} : \mathtt{TYPE} : \mathtt{KIND} \tag{9}$$
$$\vdash \mathrm{type} : \mathrm{Kind} : \mathtt{TYPE} \tag{10}$$
$$\vdash \mathrm{prop} : \mathrm{Type} : \mathtt{TYPE} \tag{11}$$
$$t : \mathrm{Type} \vdash \mathrm{El}\, t : \mathtt{TYPE} : \mathtt{KIND} \tag{12}$$
$$p : \mathrm{Prop} \vdash \mathrm{Prf}\, p : \mathtt{TYPE} : \mathtt{KIND} \tag{13}$$
$$t : \mathrm{Type}, p : \mathrm{El}\, t \to \mathrm{Prop} \vdash \forall\, t\, p : \mathrm{Prop} : \mathtt{KIND} \tag{14}$$
$$p : \mathrm{Prop}, q : \mathrm{Prf}\, p \to \mathrm{Prop} \vdash p \Rightarrow q : \mathrm{Prop} : \mathtt{KIND} \tag{15}$$
$$t : \mathrm{Type}, u : \mathrm{El}\, t \to \mathrm{Type} \vdash t \rightsquigarrow u : \mathrm{Type} : \mathtt{KIND} \tag{16}$$

Figure 4 Signature $\Sigma^{\lambda\mathrm{HOL}}$ of the encoding of λHOL into $\lambda\Pi/\equiv$.

$$\mathrm{El}\,\mathrm{prop} = \mathrm{Prop} \tag{17}$$
$$\mathrm{Prf}(\forall\, t\, p) = (x : \mathrm{El}\, t) \to \mathrm{Prf}(p\, x) \tag{18}$$
$$\mathrm{Prf}(p \Rightarrow q) = (h : \mathrm{Prf}\, p) \to \mathrm{Prf}(q\, h) \tag{19}$$
$$\mathrm{El}(t \rightsquigarrow u) = (x : \mathrm{El}\, t) \to \mathrm{El}(u\, x) \tag{20}$$

Figure 5 Equations of the encoding of λHOL into $\lambda\Pi/\equiv$.

3.3 Translation of PVS-Cert Terms Into $\lambda\Pi/\equiv$ Terms

▶ **Definition 3** (Translation). *Let Γ be a well formed context.*

- *The* term translation *of the terms M typable in Γ, noted $[M]_\Gamma$, is defined in Figures 7 and 8.*
- *The* type translation *of Kind and the terms M typable by a sort in Γ, noted $[\![M]\!]_\Gamma$, is defined in Figure 9.*
- *The* context translation *$[\![\Gamma]\!]$ is defined by induction on Γ as*

$$[\![\emptyset]\!] = \emptyset; \quad [\![\Gamma, x : T]\!] = [\![\Gamma]\!], x : [\![T]\!]_\Gamma$$

▶ **Proposition 4.** *The translation function $[\cdot]_\cdot$ that maps a context and a PVS-Cert term typable in this context to a $\lambda\Pi/\equiv$ term is well-defined.*

Proof. After Lemma 2 and [8, Lemma 41], the types of a term are unique up to equivalence. Moreover, the arguments of the translation function are decreasing with respect to the (strict) subterm relation. ◀

3.4 Examples of Encoded Theories

We provide here some examples that take advantage of proof irrelevance or predicate subtyping. While these examples could have been presented in PVS-Cert, we unfold them into the encoding of PVS-Cert into $\lambda\Pi/\equiv$ to show how it can be used in practice. All examples are

$$t : \mathrm{Type}, p : \mathrm{El}\, t \to \mathrm{Prop} \vdash \mathrm{psub}\, t\, p \quad : \mathrm{Type} \quad : \mathtt{TYPE} \tag{21}$$

$$t : \mathrm{Type}, p : \mathrm{El}\, t \to \mathrm{Prop}, m : \mathrm{El}\, t, h : \mathrm{Prf}(p\, m) \vdash \mathrm{pair}\, t\, p\, m\, h : \mathrm{El}(\mathrm{psub}\, t\, p) \quad : \mathtt{TYPE} \tag{22}$$

$$t : \mathrm{Type}, p : \mathrm{El}\, t \to \mathrm{Prop}, m : \mathrm{El}(\mathrm{psub}\, t\, p) \vdash \mathrm{fst}\, t\, p\, m \quad : \mathrm{El}\, t \quad : \mathtt{TYPE} \tag{23}$$

$$t : \mathrm{Type}, p : \mathrm{El}\, t \to \mathrm{Prop}, m : \mathrm{El}(\mathrm{psub}\, t\, p) \vdash \mathrm{snd}\, t\, p\, m \quad : \mathrm{Prf}(p\, (\mathrm{fst}\, t\, p\, m)) : \mathtt{TYPE} \tag{24}$$

Figure 6 Signature Σ^{psub} of the encoding of predicate subtyping into $\lambda\Pi/\equiv$.

$$\begin{aligned}
[x]_\Gamma &= x \\
[Prop]_\Gamma &= \mathrm{prop} \\
[Type]_\Gamma &= \mathrm{type} \\
[M\ N]_\Gamma &= [M]_\Gamma\ \ [N]_\Gamma \\
[\lambda x : T,\ M]_\Gamma &= \lambda x : \mathrm{El}\ \ [T]_\Gamma,\ [M]_{\Gamma, x:T} \\
[(x : T) \to U]_\Gamma &= [T]_\Gamma \rightsquigarrow \left(\lambda x : [\![T]\!]_\Gamma,\ [U]_{\Gamma, x:T}\right) \\
&\qquad \text{when } \Gamma \vdash_{\mathrm{PVS}} T : Type \text{ and } \Gamma, x : T \vdash_{\mathrm{PVS}} U : Type \\
[(x : T) \to P]_\Gamma &= \forall\, [T]_\Gamma\ \left(\lambda x : [\![T]\!]_\Gamma,\ [P]_{\Gamma, x:T}\right) \\
&\qquad \text{when } \Gamma \vdash_{\mathrm{PVS}} T : Type \text{ and } \Gamma, x : T \vdash_{\mathrm{PVS}} P : Prop \\
[(h : P) \to Q]_\Gamma &= [P]_\Gamma \Rightarrow \left(\lambda h : [\![P]\!]_\Gamma,\ [Q]_{\Gamma, h:P}\right) \\
&\qquad \text{when } \Gamma \vdash_{\mathrm{PVS}} P : Prop \text{ and } \Gamma, h : P \vdash_{\mathrm{PVS}} Q : Prop
\end{aligned}$$

Figure 7 Translation from λHOL to $\lambda\Pi/\equiv$.

$$\begin{aligned}
[\mathrm{psub}\, T\ P]_\Gamma &= \mathrm{psub}\ \ [T]_\Gamma\ [P]_\Gamma & [\mathrm{fst}\, T\ P\ M]_\Gamma &= \mathrm{fst}\, [T]_\Gamma\ [P]_\Gamma\ [M]_\Gamma \\
[\mathrm{pair}\, T\ P\ M\ N]_\Gamma &= \mathrm{pair}\, [T]_\Gamma\ [P]_\Gamma\ [M]_\Gamma\ \ [N]_\Gamma & [\mathrm{snd}\, T\ P\ M]_\Gamma &= \mathrm{snd}\, [T]_\Gamma\ [P]_\Gamma\ [M]_\Gamma
\end{aligned}$$

Figure 8 Translation from PVS-Cert to $\lambda\Pi/\equiv$.

$$\begin{aligned}
[\![T]\!]_\Gamma &= \mathrm{El}\ \ [T]_\Gamma \quad \text{when } \Gamma \vdash_{\mathrm{PVS}} T : Type; & [\![Kind]\!] &= \mathrm{Kind} \\
[\![T]\!]_\Gamma &= \mathrm{Prf}\ \ [T]_\Gamma \quad \text{when } \Gamma \vdash_{\mathrm{PVS}} T : Prop; & [\![Type]\!] &= \mathrm{Type}
\end{aligned}$$

Figure 9 Translation of types from PVS-Cert to $\lambda\Pi/\equiv$.

```
symbol stack : Type ;          symbol empty : El stack;          symbol t : Type ;
symbol nonempty_stack?(s : El stack) ≔ s ≠ empty;
symbol nonempty_stack ≔ psub nonempty_stack?;
symbol  push : El stack → El t → El nonempty_stack;
symbol  pop : El nonempty_stack → El stack;
symbol  pop_push(x : El t)(s : El stack) : Prf(pop(push x s) = s);
symbol  pop2push2(x y : El t)(s : El stack)
        : Prf(pop(pair (pop(push x (fst(push y s)))) ?0) = s) ≔ . . . ;
```

Figure 10 Specification for stacks.

available as Dedukti files[1] and can be type-checked with Lambdapi[2]. In the examples, the first two arguments of fst, pair and snd are implicit.

▶ **Example 5** (Stacks with predicate subtypes)**.** This example comes from the language reference manual of PVS [26] and illustrates the use of predicate subtyping and the generation of TCC through a specification of stacks in Figure 10.

Predicate subtyping is used to define the type of nonempty stacks, which allows the function pop to be total. Symbol pop_push is an axiom that uses Leibniz equality = on stacks. In the definition of the theorem pop2push2, term $?_0$ is a meta-variable that must be instantiated with a proof that the first argument of the pair is not empty, and represents, in the encoding, the TCC generated by PVS. We can thus see that the concept of TCC of PVS has a clear and explicit representation in the encoding, allowing its benefits to be transported to λΠ/≡.

▶ **Example 6** (Bounded lists and proof irrelevance)**.** This example is inspired by sorted lists in the Agda manual [33][3]. Because we have not encoded dependent types, we cannot encode the type of lists bounded by a variable. We thus declare the bound in the signature. The specification is given in Figure 11.

We first notice that the predicate subtype allows to encode the proof $head \leq$ bound passed as a standalone argument in Agda in the type of an argument in our encoding, providing a shorter type for bcons. In Figure 12, we define two (non-convertible) axioms p_1 and p_2 as proofs of zero ≤ suc bound, and two lists containing zero but proved to be bounded by suc bound using p_1 for ℓ_1 and p_2 for ℓ_2. Type checking ℓ_i requires axioms p_i. These axioms are like TCC's in PVS. Assuming that one wants to prove $\ell_1 = \ell_2$, had we lacked proof irrelevance, we would have had to prove that $p_1 \equiv p_2$, which is not possible. In our case, the equality is simply the result of refl ℓ_1.

4 Correctness of the Encoding

In this section, we prove that the encoding is correct: if a PVS-Cert type is inhabited then its translation is inhabited too. Any type-checker for λΠ/≡ could thus be used to recheck PVS-Cert typings. However, to make sure that our encoding is faithful (the encoding that

[1] https://github.com/Deducteam/personoj/paper/
[2] https://github.com/Deducteam/lambdapi, commit 0875521
[3] https://agda.readthedocs.io/en/v2.5.4/language/irrelevance.html

```
symbol zero : El ℕ;                    symbol bound ≔ ...;
symbol suc(n : El ℕ) : El ℕ;           symbol blist : Type;
symbol ≤ (n m : El ℕ) : Prop;          symbol bnil : El blist;

symbol bounded ≔ psub(λn, n ≤ bound);
symbol bcons(head : El bounded)(tail : El blist) : El blist;
```

Figure 11 Specification of sorted lists.

```
symbol p₁ : Prf(zero ≤ suc bound);   symbol ℓ₁ ≔ bcons(pair zero p₁) bnil;
symbol p₂ : Prf(zero ≤ suc bound);   symbol ℓ₂ ≔ bcons(pair zero p₂) bnil;
```

Figure 12 Definition of two sorted lists with different proofs.

maps any PVS-Cert term to the same well-typed ground term is correct, but useless), completeness (also called conservativity) ought to be proved too: a PVS-Cert type is inhabited whenever its encoding is inhabited. However, as completeness is often difficult to establish (see [3, 34]), we leave it for future work.

In the following,

- s stands for $Type$, $Prop$ or $Kind$;
- T, U designate terms of type $Type$;
- M, N, t, u designate expressions that have a type $T : Type$;
- P, Q are propositions of type $Prop$, or predicates of type $T \to Prop$;
- h stands for a proof typed by a proposition.

Typing judgements in PVS-Cert are noted with $\vdash_{\text{PVS}}$, and typing judgements in $\lambda\Pi/\equiv$ are noted with $\vdash_{\lambda\Pi/\equiv}$.

▶ **Lemma 7** (Preservation of substitution). *If* $\Gamma, x : U, \Delta \vdash_{\text{PVS}} M : T$ *and* $\Gamma \vdash_{\text{PVS}} N : T$*, then* $[\{x \mapsto N\}\, M]_{\Gamma,\{x\mapsto N\}\Delta} = \{x \mapsto [N]_\Gamma\}\, [M]_{\Gamma,x:U,\Delta}$.

Proof. By structural induction on M. ◀

▶ **Lemma 8** (Preservation of equivalence). *Let* M *and* N *be two well typed terms in* Γ.

1. *If* $M \underset{\text{PVS}}{\leftrightarrow} N$*, then* $[M]_\Gamma \underset{\lambda\Pi}{\equiv} [N]_\Gamma$.
2. *If* $M \underset{\text{PVS}}{\equiv} N$*, then* $[M]_\Gamma \underset{\lambda\Pi}{\equiv} [N]_\Gamma$.

Proof. Each item is proved separately.

1. Taking back the notations of the proof of Lemma 2, we show that
   a. computational steps of $\hookrightarrow^{ty}_{\beta\text{fst}}$ are preserved,
   b. equational steps of $\leftrightarrow^{ty}_{pi}$ are preserved.

   These two properties are shown by induction on a context C such that $M = C[\hat{M}]\, R\, C[\hat{N}] = N$ where R is any of the two relations applied at the head of $\hat{M}$ and $\hat{N}$. We will only detail the base cases of inductions, the other cases being straightforward.

Preservation of Computation There are two possible cases,

Case $M = ((\lambda x,\, t)\ u) \hookrightarrow_\beta \{x \mapsto u\}\, t$, we have,

$$[(\lambda x : U,\, t)\ u]_\Gamma = ((\lambda x : [\![U]\!]_\Gamma,\, [t]_{\Gamma,x:U})\ [u]_\Gamma) = \{x \mapsto [u]_\Gamma\}\, [t]_\Gamma \underset{\lambda\Pi}{\equiv} [\{x \mapsto u\}\, t]_\Gamma$$

where the equivalence is given by Lemma 7.

Case $M = \mathrm{fst}\, T_1\ P_1\ (\mathrm{pair}\, T_0\ P_0\, t\, h) \hookrightarrow_{\mathrm{fst}} t$, we have the following equalities

$$\begin{aligned}[\mathrm{fst}\, T_1\ P_1\ (\mathrm{pair}\, T_0\ P_0\, t\, h)]_\Gamma &= \mathrm{fst}\, [T_1]_\Gamma\ [P_1]_\Gamma\ [\mathrm{pair}\, T_0\ P_0\, t\, h]_\Gamma \\ &= \mathrm{fst}\, [T_1]_\Gamma\ [P_1]_\Gamma\ (\mathrm{pair}\, [T_0]_\Gamma\ [P_0]_\Gamma\ [t]_\Gamma\ [h]_\Gamma) \\ &\underset{\lambda\Pi}{\equiv} [t]_\Gamma\end{aligned}$$

with the last equivalence provided by Equation (6).

Preservation of Proof Irrelevance Assume that $M = \mathrm{pair}\, T\ P\, t\, h \leftrightarrow_{pi} \mathrm{pair}\, T\ P\, t\, h'$

$$[\mathrm{pair}\, T\ P\, t\, h]_\Gamma = \mathrm{pair}\, [T]_\Gamma\ [P]_\Gamma\ [t]_\Gamma\ [h]_\Gamma \underset{\lambda\Pi}{\equiv} \mathrm{pair}\, [T]_\Gamma\ [P]_\Gamma\ [t]_\Gamma\ [h']_\Gamma = [\mathrm{pair}\, T\ P\, t\, h']_\Gamma$$

where the equivalence is given by Equation (5).

2. By Lemma 2, we know that there are H_0 and H_1 such that $M(\hookrightarrow^{ty}_{\beta\mathrm{fst}})^* H_0 (\leftrightarrow^{ty}_{pi})^* H_1 (\leftarrow^{ty}_{\beta\mathrm{fst}})^* N$. For $R \in \{\leftrightarrow_{pi}, \hookrightarrow_{\beta\mathrm{fst}}\}$, we have $t(R^{ty})^* u \Rightarrow [t] \underset{\lambda\Pi}{\equiv} [u]$ by induction on the number of R^{ty} steps, using Item 1 for the base case. Therefore, $[M]_\Gamma \underset{\lambda\Pi}{\equiv} [H_0]_\Gamma \underset{\lambda\Pi}{\equiv} [H_1]_\Gamma \underset{\lambda\Pi}{\equiv} [N]_\Gamma$, which gives, by transitivity of $\underset{\lambda\Pi}{\equiv}$, $[M]_\Gamma \underset{\lambda\Pi}{\equiv} [N]_\Gamma$.

◀

▶ **Theorem 9** (Correctness). *If* $\Gamma \vdash_{\mathrm{PVS}} M : T$, *then* $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [M]_\Gamma : [\![T]\!]_\Gamma$. *For all* Γ, *if* $\Gamma \vdash_{\mathrm{PVS}} WF$, *then* $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} WF$.

Proof. By induction on the typing derivation of $\Gamma \vdash_{\mathrm{PVS}} M : T$ and case distinction on the last inference rule.

empty $\dfrac{}{\emptyset \vdash_{\mathrm{PVS}} WF}$

We have $[\![\emptyset]\!] = \emptyset$ and $\emptyset \vdash_{\lambda\Pi/\equiv} WF$.

decl $\dfrac{\Gamma \vdash_{\mathrm{PVS}} T : s}{\Gamma, v : T \vdash_{\mathrm{PVS}} WF}\, v \notin \Gamma$

We have $[\![\Gamma, v : T]\!] = [\![\Gamma]\!], v : [\![T]\!]_\Gamma$. By induction hypothesis, we have $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [T]_\Gamma : [\![s]\!]_\Gamma$, for $s \in \mathcal{S}$ and hence $[\![s]\!]_\Gamma$ is either Prop by conversion (because $\mathrm{El}\,\mathrm{prop} \underset{\lambda\Pi}{\equiv} \mathrm{Prop}$), Type or Kind. If s is $Kind$, then T is $Type$. Since $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \mathrm{Type} : \mathtt{TYPE}$ because $\Sigma^{\mathrm{PC}}(\mathrm{Type}) = (\vec{0}, (\mathtt{TYPE}, \mathtt{KIND}))$, we can derive with the declaration rule $[\![\Gamma, v : T]\!] \vdash_{\lambda\Pi/\equiv} WF$ because $[\![Type]\!] = \mathrm{Type}$. Otherwise, s is $Type$ or $Prop$ and $[\![T]\!] = \xi\ [T]_\Gamma$ where ξ is El or Prf. By typing of El or Prf (with the signature), $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [\![T]\!]_\Gamma : \mathtt{TYPE}$ and finally, $[\![\Gamma, v : T]\!] \vdash_{\lambda\Pi/\equiv} WF$ by application of the declaration rule.

var $\dfrac{\Gamma \vdash_{\mathrm{PVS}} WF}{\Gamma \vdash_{\mathrm{PVS}} v : T}\, v : T \in \Gamma$

By definition, $[v] = v$ and by induction hypothesis, $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} WF$. Since $v : T \in \Gamma$, by definition, there is $\Delta \subsetneq \Gamma$, $\Delta \vdash_{\mathrm{PVS}} WF$ such that, $v : [\![T]\!]_\Delta \in [\![\Gamma]\!]$. Hence $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} v : [\![T]\!]_\Delta$ and finally $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} v : [\![T]\!]_\Gamma$ because contexts are well formed.

sort $\dfrac{\Gamma \vdash_{\mathrm{PVS}} WF}{\Gamma \vdash_{\mathrm{PVS}} s_1 : s_2}\, (s_1, s_2) \in \mathcal{A}$

First, $[s_1]$ is either prop or type. In the former case, $[\![s_2]\!] = \mathrm{Type}$ and because $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} WF$ (by induction hypothesis) and $\Sigma^{\mathrm{PC}}(\mathrm{prop}) = (\vec{0}, (\mathrm{Type}, \mathtt{TYPE}))$, we have $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \mathrm{prop} : \mathrm{Type}$. The same procedure holds for $s_1 = Type$ and $s_2 = Kind$.

$$\textbf{prod}\quad \frac{\Gamma \vdash_{\text{PVS}} T : s_1 \quad \Gamma, x : T \vdash_{\text{PVS}} U : s_2}{\Gamma \vdash_{\text{PVS}} (x : T) \to U : s_3}\,(s_1, s_2, s_3) \in \mathcal{P}$$

We only detail for the product $(Type, Prop, Prop)$, others being processed similarly. We have $[(x : T) \to U]_\Gamma = \forall\,[T]_\Gamma\,\left(\lambda x : [\![T]\!]_\Gamma,\, [U]_{\Gamma,x:T}\right)$. By induction hypothesis, $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [T] : [\![Type]\!]$, and thus $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [T] : \text{Type}$ by definition. By induction hypothesis, $[\![\Gamma, x : T]\!] \vdash_{\lambda\Pi/\equiv} [U] : [\![Prop]\!]$, and thus $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_{\lambda\Pi/\equiv} [U] : \text{Prop}$ by definition of $[\![\cdot]\!]$ and conversion which yields $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \lambda x : [\![T]\!]_\Gamma,\, [U]_{\Gamma,x:T} : [\![T]\!]_\Gamma \to \text{Prop}$.

To finish, we obtain $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \lambda x : [\![T]\!]_\Gamma,\, [U]_{\Gamma,x:T} : (\text{El}\,[T]_\Gamma) \to \text{Prop}$ by conversion. Using the typing signature Σ^{PC}, $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \forall\,[T]_\Gamma\,\left(\lambda x,\, [\![T]\!]_\Gamma [U]_{\Gamma,x:T}\right) : \text{Prop}$ which becomes, by conversion $\text{Prop} \underset{\lambda\Pi}{\equiv} \text{El}\,\text{prop}$ and definition of $[\![\cdot]\!]_\Gamma$: $\text{El}\,\text{prop} = [\![Prop]\!]$, hence, $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \forall\,[T]_\Gamma\,\left(\lambda x,\, [\![T]\!]_\Gamma [U]_{\Gamma,x:T}\right) : [\![Prop]\!]$

$$\textbf{abst}\quad \frac{\Gamma, v : T \vdash_{\text{PVS}} M : U \quad \Gamma \vdash_{\text{PVS}} (v : T) \to U : s}{\Gamma \vdash_{\text{PVS}} \lambda v : T,\, M : (v : T) \to U}$$

We have $[\lambda v : T,\, M]_\Gamma = \lambda v : [\![T]\!]_\Gamma,\, [M]_\Gamma$. By induction hypothesis, $[\![\Gamma, v : T]\!] \vdash_{\lambda\Pi/\equiv} [M]_{\Gamma,v:T} : [\![U]\!]_{\Gamma,v:T}$ and by definition of $[\![\cdot]\!]$, $[\![\Gamma]\!], v : [\![T]\!]_\Gamma \vdash_{\lambda\Pi/\equiv} [M]_{\Gamma,v:T} : [\![U]\!]_{\Gamma,v:T}$. Applying the abstraction rule in $\lambda\Pi/\equiv$, we obtain $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \lambda v : [\![T]\!]_\Gamma,\, [M]_{\Gamma,v:T} : (v : [\![T]\!]_\Gamma) \to [\![U]\!]_{\Gamma,v:T}$ (with the product well typed in $\lambda\Pi/\equiv$ since $[\![U]\!]$ and $[\![T]\!]$ are both of type `TYPE` and thus the product is of type `TYPE` as well).

Finally, we proceed by case distinction on sorts s_T and s_U such that $\Gamma \vdash_{\text{PVS}} T : s_T$ and $\Gamma \vdash_{\text{PVS}} U : s_U$. We will detail the case $(s_T, s_U) = (Type, Prop)$. We have $(v : [\![T]\!]_\Gamma) \to [\![U]\!]_{\Gamma,v:T} \underset{\lambda\Pi}{\equiv} \text{Prf}(\forall\,[T]_\Gamma\,(\lambda x : [\![T]\!]_\Gamma,\, [U]_{\Gamma,v:T})) = [\![(v : T) \to U]\!]_\Gamma$ which allows to conclude.

$$\textbf{app}\quad \frac{\Gamma \vdash_{\text{PVS}} M : (v : T) \to U \quad \Gamma \vdash_{\text{PVS}} N : T}{\Gamma \vdash_{\text{PVS}} M\,N : \{v \mapsto N\}\,U}$$

By induction hypothesis and conversion, we have $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [M]_\Gamma : (v : [\![T]\!]_\Gamma) \to [\![U]\!]_{\Gamma,v:T}$ (shown by case distinction on the sorts of T and U) and $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [N]_\Gamma : [\![T]\!]_\Gamma$. Since $[M\,N]_\Gamma = [M]\ [N]$, we obtain using the application rule $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [M\,N] : \{v \mapsto [N]_\Gamma\}\,[\![U]\!]_{\Gamma,v:T}$ and by Lemma 7, we obtain $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [M\,N] : [\![\{v \mapsto N\}\,U]\!]_\Gamma$.

$$\textbf{conv}\quad \frac{\Gamma \vdash_{\text{PVS}} M : U \quad \Gamma \vdash_{\text{PVS}} T : s \quad T \underset{\text{PVS}}{\equiv} U}{\Gamma \vdash_{\text{PVS}} M : T}$$

By hypothesis, there is a type U such that $\Gamma \vdash_{\text{PVS}} M : U$, and $T \underset{\text{PVS}}{\equiv} U$, and there is a sort s such that $\Gamma \vdash_{\text{PVS}} T : s$. By induction hypothesis, $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [M]_\Gamma : [\![U]\!]_\Gamma$.

We now prove that if $T \underset{\text{PVS}}{\equiv} U$, then $[\![T]\!]_\Gamma \underset{\lambda\Pi}{\equiv} [\![U]\!]_\Gamma$ and $\Gamma \vdash_{\lambda\Pi/\equiv} [\![T]\!] : \text{TYPE}$: it will allow us to conclude using the conversion rule in $\lambda\Pi/\equiv$.

By Lemma 2, we have $T \hookrightarrow^*_{\beta\text{fst}} T' =_{pi} U' \hookleftarrow^*_{\beta\text{fst}} U$ and $T(\hookrightarrow^{ty}_{\beta\text{fst}})^* T' (\leftrightarrow^{ty}_{pi})^* U' (\hookleftarrow^{ty}_{\beta\text{fst}})^* U$. Because $\hookrightarrow_{\beta\text{fst}}$ preserves typing (Lemma 2), we have $\Gamma \vdash_{\text{PVS}} U' : s$. By [8, Lemma 43], $\Gamma \vdash_{\text{PVS}} T : s$. By Lemma 8, $[T]_\Gamma \underset{\lambda\Pi}{\equiv} [U]_\Gamma$

If $s = Prop$, then $[\![T]\!]_\Gamma = \text{Prf}\,[T]_\Gamma \underset{\lambda\Pi}{\equiv} \text{Prf}\,[U]_\Gamma = [\![U]\!]_\Gamma$. Moreover we have $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [\![T]\!]_\Gamma : \text{TYPE}$ because, by induction hypothesis, $[T]_\Gamma : [\![Prop]\!] = \text{El}\,[Prop] = \text{El}\,\text{prop} = \text{Prop}$, and $(p : \text{Prop} \vdash_{\Sigma^{\text{PC}}} \text{Prf}\,p : \text{TYPE} : \text{KIND})$. If $s = Type$, $[\![T]\!]_\Gamma = \text{El}\,[T]_\Gamma \underset{\lambda\Pi}{\equiv} \text{El}\,[U]_\Gamma = [\![U]\!]_\Gamma$. By induction hypothesis, $[T]_\Gamma : [\![Type]\!]_\Gamma = \text{Type}$. If $s = Kind$, then $T = U = Type$ ($Type$ is the only inhabitant of $Kind$). Finally, $[\![Type]\!] = \text{Type} : \text{TYPE}$.

$$\textbf{sig}\quad \frac{\overrightarrow{x : T} \vdash U : s \quad \left(\Gamma \vdash t_i : \left\{(x_j \mapsto t_j)_{j<i}\right\} T_i\right)_i}{\Gamma \vdash f(\vec{t}) : \left\{\overrightarrow{x \mapsto t}\right\} U}\,\Sigma(f) = (\overrightarrow{x, T}, U, s)$$

We first observe from Figure 6 that for each $f \in \Sigma^{\mathrm{PVS}}$, we have a counterpart symbol $\hat{f} \in \Sigma^{\mathrm{PC}}$ such that if $\Sigma^{\mathrm{PVS}}(f) = \left(\overrightarrow{x:T}, U, s\right)$, then $\Sigma^{\mathrm{PC}}(\hat{f}) = \left(\overrightarrow{x, [\![T]\!]}, [\![U]\!]_{\overrightarrow{x:T}}, \mathtt{TYPE}\right)$. By induction hypothesis, for each i, we have $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [t_i]_\Gamma : [\![\{(x_j \mapsto t_j)_{j<i}\} T_i]\!]_\Gamma$ which we can write as, thanks to Lemma 7, $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} [t_i]_\Gamma : \left\{(x_j \mapsto [t_j]_\Gamma)_{j<i}\right\} [\![T_i]\!]_\Gamma$. Now, using the signature rule, we are able to conclude $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \hat{f}\,\overrightarrow{[t]_\Gamma} : \left\{\overrightarrow{x \mapsto [t]}\right\} [\![U]\!]$. By Lemma 7, we obtain $[\![\Gamma]\!] \vdash_{\lambda\Pi/\equiv} \hat{f}\,\overrightarrow{[t]_\Gamma} : \left[\!\left[\left\{\overrightarrow{x \mapsto t}\right\} U\right]\!\right]$. Moreover, we have taken care to define the translation in Figure 8 such that $\left[f(\overrightarrow{t})\right] = \hat{f}\,\overrightarrow{[t]}$. ◀

5 Mechanised Type Checking

The encoding of PVS-Cert into λΠ/≡ can be used to proof check terms of PVS-Cert using a type checker for λΠ/≡. But because of the rule

$$\frac{\Gamma \vdash t : B \quad \Gamma \vdash A : s \quad A \equiv B}{\Gamma \vdash t : A} \qquad (\lambda\Pi/\equiv\text{-conv})$$

type checking is decidable only if ≡ is. A decidable relation equivalent to ≡ can be obtained using the convertibility relation stemming from the rewriting relation of a convergent rewrite system, yielding the type system $\lambda\Pi/R$ (R for *rewriting*). Consequently, while type checkers cannot be provided for λΠ/≡ in general, they can for $\lambda\Pi/R$, as can be seen with Dedukti[4]. Such rewrite systems can be obtained through *completion procedures* [6]. However, completion procedures rely on a well-founded order that cannot be provided here because of Equation (5) which cannot be oriented since each side of the equation has a free variable which is not in the other side.

A possible solution would be to rewrite all proofs of a pair to a canonical proof with a rule of the form

$$\mathrm{pair}\, t\, p\, m\, h \hookrightarrow \mathrm{pair}\, t\, p\, m\, (\mathrm{canon}\, t\, p\, m)$$

where $t : \mathrm{Type}, p : \mathrm{El}\, t \to \mathrm{Prop}, m : \mathrm{El}\, t \vdash \mathrm{canon}\, t\, p\, m : \mathrm{Prf}(p\, m) : \mathtt{TYPE}$. But this creates a rewrite rule that duplicates three variables.

Otherwise, as noted in [23], the addition of a symbol to the signature can circumvent the issue. Hence, we add a symbol for proof irrelevant pairs, and make it equal to pairs

$$t : \mathrm{Type}, p : \mathrm{El}\, t \to \mathrm{Prop}, m : \mathrm{El}\, t \vdash \mathrm{pair}^\dagger\, t\, p\, m : \mathrm{El}(\mathrm{psub}\, t\, p) : \mathtt{TYPE} \tag{25}$$

$$\mathrm{pair}\, t\, p\, m\, h = \mathrm{pair}^\dagger\, t\, p\, m \tag{26}$$

thus $(\mathrm{pair}\ \ t\ p\ m\ h) \equiv (\mathrm{pair}^\dagger\ \ t\ p\ m) \equiv (\mathrm{pair}\ \ t\ p\ m\ h')$. The new set of identities given by Equations (6), (17)–(20), and (26) can be completed into a rewrite system R which is equivalent to the equations:

▶ **Proposition 10.** *Let $\hookrightarrow_R$ be the closure by context and substitution of the rewrite rules of Figure 13, and $\equiv_R$ be the smallest equivalence containing $\hookrightarrow_R$. Then, for all $M, N \in \mathcal{T}(\Sigma^{\mathrm{PC}}, \mathcal{S}^{\lambda\Pi}, \mathcal{V})$, if $M \underset{\lambda\Pi}{\equiv} N$ then $M \equiv_R N$.*

[4] `https://github.com/Deducteam/lambdapi.git`

$$(\lambda x : T,\, t)\, u \hookrightarrow \{x \mapsto u\}\, t \tag{27}$$

$$\mathrm{pair}\, t\, p\, m\, h \hookrightarrow \mathrm{pair}^{\dagger}\, t\, p\, m \tag{28}$$

$$\mathrm{fst}\, t_0\, p_0\, (\mathrm{pair}^{\dagger}\, t_1\, p_1\, m) \hookrightarrow m \tag{29}$$

$$\mathrm{El}\, \mathrm{prop} \hookrightarrow \mathrm{Prop} \tag{30}$$

$$\mathrm{Prf}(\forall\, t\ p) \hookrightarrow (x : \mathrm{El}\, t) \to \mathrm{Prf}(p\ x) \tag{31}$$

$$\mathrm{El}(t \rightsquigarrow u) \hookrightarrow (x : \mathrm{El}\, t) \to \mathrm{El}(u\ x) \tag{32}$$

$$\mathrm{Prf}(p \Rightarrow q) \hookrightarrow (h : \mathrm{Prf}\, p) \to (\mathrm{Prf}(q\ h)) \tag{33}$$

Figure 13 Rewrite system R resulting from the completion of the equations of the encoding of PVS-Cert in $\lambda\Pi/\equiv$.

Proof. It suffices to prove that every equation of PVS-Cert is included in $\equiv_R$. This is immediate for the Equations (17)–(20) and (β) since they are equal to the rules (27) and (30)–(33). For the Equation (5), we have $\mathrm{pair}\, t\, p\, m\, h_0 \hookrightarrow_R \mathrm{pair}^{\dagger}\, t\, p\, m \hookleftarrow_R \mathrm{pair}\, t\, p\, m\, h_1$. Finally, for the Equation (6), we have $\mathrm{fst}\, t_0\, p_0\, (\mathrm{pair}\, t_1\, p_1\, m\, h) \hookrightarrow_R \mathrm{fst}\, t_0\, p_0\, (\mathrm{pair}^{\dagger}\, t_1\, p_1\, m) \hookrightarrow_R m$. ◀

▶ Remark 11.

- Rewrite system R is confluent because it is orthogonal.
- Termination of R is required to obtain the decidability of $\equiv_R$. A possible approach to prove it would be to extend the termination model of λHOL described in [15].
- In order to prove the completeness of the encoding, that is, the fact that a type is inhabited whenever its encoding is, it could be useful to have the reciprocal implication, that is, if $M \equiv_R N$ and $M, N \in \mathcal{T}(\Sigma^{\mathrm{PC}}, \mathcal{S}^{\lambda\Pi}, \mathcal{V})$, then $M \underset{\lambda\Pi}{\equiv} N$. We leave this for future work too.

A priori, the introduction of $\mathrm{pair}^{\dagger}$ allows one to craft terms that cannot be proof checked in PVS-Cert. Indeed, given a predicate Even on natural numbers, the term $(\mathrm{pair}^{\dagger}\, \mathbb{N}\ \mathrm{Even}\ 3)$ is the encoding of $(\mathrm{pair}\, \mathbb{N}\ \mathrm{Even}\, 3\, h)$ which cannot be type checked in PVS-Cert since there is no proof h that 3 is even. However, Dedukti relies on a system of modules and tags attached to symbols to define where and how symbols can be used. A symbol tagged *protected* cannot be used to build terms outside of the module where it is defined, but it may appear during type checking because of conversion, a trick first introduced in [35] and used also for encoding Cumulative Type Systems in $\lambda\Pi/\equiv$ [34]. In our case, one may protect $\mathrm{pair}^{\dagger}$ in the module that defines the encoding of PVS-Cert, so that users of the encoding are forced to use pair.

Conclusion

This work provides an encoding of predicate subtyping with proof irrelevance into the $\lambda\Pi$-calculus modulo theory, $\lambda\Pi/\equiv$ [4]. We first recall PVS-Cert, an extension of higher-order logic with predicate subtyping and proof irrelevance [17]. We then provide a $\lambda\Pi/\equiv$ signature to encode terms of PVS-Cert, and prove that the encoding is correct: if a PVS-Cert type is inhabited, then its translation in $\lambda\Pi/\equiv$ is inhabited too. Finally, we show that the equational theory of our encoding is equivalent to a confluent set of rewrite rules which enable us to use Dedukti to type check encoded specifications.

However, two important problems are left open. First, is our encoding complete, that is, is a PVS-Cert type inhabited if its translation is? Second, is the confluent rewrite system used in the encoding terminating? We believe that these two properties hold but leave their difficult study for future work.

Perspectives

The encoding of PVS-Cert in $\lambda\Pi/R$ is the stepping stone towards an automatic translator from PVS to Dedukti. Indeed, PVS does not have proof terms in its syntax, and consequently type checking is undecidable. The creation of PVS-Cert allows to convert PVS terms to a syntax whose type checking is decidable. This was the work of F. Gilbert in [17]. Now we are able to express this decidable syntax in $\lambda\Pi/R$ and hence in Dedukti. However, the type system proposed here only allows to coerce from a type to its direct supertype or a subtype, that is, we can go from $(\mathsf{psub}\,(\mathsf{psub}\,\iota\,P)\,Q)$ to $\mathsf{psub}\,\iota\,P$ in one coercion, but we cannot coerce from $(\mathsf{psub}\,(\mathsf{psub}\,\iota\,P)\,Q)$ to ι, whereas PVS can. Consequently, an algorithm to elaborate the correct sequence of coercions is needed to obtain terms that can be type checked in Dedukti.

Other features of PVS can be integrated into PVS-Cert and the encoding: dependent types like $(\mathsf{psub}\ \mathit{list}\ (\lambda\ell,\ \mathsf{length}\ \ell = n))$, recursive definitions of functions, and dependent records. With those features encoded, almost all the standard library[5] of PVS can be translated to Dedukti.

Finally, while the previous points were concerned with the translation of specifications from PVS, we may also want to translate proofs developed in PVS. These proofs are witnesses of *type correctness conditions* (TCC), which are required to type check terms. Since PVS is a highly automated prover, proof terms often come from application of complex tactics that cannot be mimicked into Dedukti. However, proof terms may either be provided by hand, emulating the interaction provided by TCC's, or we may call external solvers [19].

References

1. Andreas Abel, Thierry Coquand, and Miguel Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. *Log. Methods Comput. Sci.*, 7(2), 2011. doi:10.2168/LMCS-7(2:4)2011.
2. Andrea Asperti, Wilmer Ricciotti, and Claudio Sacerdoti Coen. Matita tutorial. *J. Formaliz. Reason.*, 7(2):91–199, 2014. doi:10.6092/issn.1972-5787/4651.
3. A. Assaf. *A framework for defining computational higher-order logics*. PhD thesis, École Polytechnique, France, 2015. URL: https://tel.archives-ouvertes.fr/tel-01235303/.
4. A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant, and R. Saillard. Dedukti: a logical framework based on the $\lambda\pi$-calculus modulo theory, 2019. Draft. URL: http://lsv.fr/~dowek/Publi/expressing.pdf.
5. Ali Assaf and Guillaume Burel. Translating HOL to dedukti. In Cezary Kaliszyk and Andrei Paskevich, editors, *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015*, volume 186 of *EPTCS*, pages 74–88, 2015. doi:10.4204/EPTCS.186.8.
6. Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
7. Henk Barendregt and Kees Hemerik. Types in Lambda Calculi and Programming Languages. In Neil D. Jones, editor, *ESOP'90, 3rd European Symposium on Programming, Copenhagen, Denmark, May 15-18, 1990, Proceedings*, volume 432 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 1990. doi:10.1007/3-540-52592-0_53.
8. F. Blanqui. *Théorie des types et récriture*. PhD thesis, Université Paris-Sud, France, 2001. URL: http://tel.archives-ouvertes.fr/tel-00105522.

[5] http://www.cs.rug.nl/~grl/ar06/prelude.html

9 Frédéric Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1), 2005. doi:10.1017/S0960129504004426.

10 M. Boespflug and G. Burel. CoqInE: translating the calculus of inductive constructions into the lambda-Pi-calculus modulo. In *Proceedings of the 2nd International Workshop on Proof eXchange for Theorem Proving*, CEUR Workshop Proceedings 878, 2012. URL: http://ceur-ws.org/Vol-878/paper3.pdf.

11 Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, Todd B. Knoblock, N. P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing mathematics with the Nuprl proof development system*. Prentice Hall, 1986. URL: http://dl.acm.org/citation.cfm?id=10510.

12 D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-Pi-calculus modulo. In *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 4583, 2007. doi:10.1007/978-3-540-73228-0_9.

13 N.G. de Bruijn. Some Extensions of Automath: The AUT-4 Family. In R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 283–288. Elsevier, 1994. doi:10.1016/S0049-237X(08)70209-X.

14 Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015. doi:10.1007/978-3-319-21401-6_26.

15 Gilles Dowek. Models and termination of proof reduction in the lambda pi-calculus modulo theory. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 109:1–109:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.109.

16 G. Genestier. ncoding agda programs using rewriting. In *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 167, 2020. doi:10.4230/LIPIcs.FSCD.2020.31.

17 Frederic Gilbert. *Extending higher-order logic with predicate subtyping : application to PVS*. Theses, Université Sorbonne Paris Cité, 2018. URL: https://tel.archives-ouvertes.fr/tel-02058937.

18 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without k. *Proc. ACM Program. Lang.*, 3(POPL), 2019. doi:10.1145/3290316.

19 Mohamed Yacine El Haddad, Guillaume Burel, and Frédéric Blanqui. EKSTRAKTO A tool to reconstruct dedukti proofs from TSTP files (extended abstract). In Giselle Reis and Haniel Barbosa, editors, *Proceedings Sixth Workshop on Proof eXchange for Theorem Proving, PxTP 2019, Natal, Brazil, August 26, 2019*, volume 301 of *EPTCS*, pages 27–35, 2019. doi:10.4204/EPTCS.301.5.

20 Joe Hurd. Predicate subtyping with predicate sets. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings*, volume 2152 of *Lecture Notes in Computer Science*, pages 265–280. Springer, 2001. doi:10.1007/3-540-44755-5_19.

21 Matt Kaufmann and J. Strother Moore. An industrial strength theorem prover for a logic based on common lisp. *IEEE Trans. Software Eng.*, 23(4):203–213, 1997. doi:10.1109/32.588534.

22 Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121(1&2):279–308, 1993. doi:10.1016/0304-3975(93)90091-7.

23 D. Knuth and P. Bendix. Simple word problems in universal algebras. In *Automation of Reasoning. Symbolic Computation (Artificial Intelligence)*. Springer, 1983.

24 William Lovas and Frank Pfenning. Refinement types for logical frameworks and their interpretation as proof irrelevance. *Log. Methods Comput. Sci.*, 6(4), 2010. doi:10.2168/LMCS-6(4:5)2010.

25 Zhaohui Luo. *An extended calculus of constructions*. PhD thesis, University of Edinburgh, UK, 1990. URL: http://hdl.handle.net/1842/12487.

26 S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, 1999.

27 Sam Owre, John Rushby, N. Shankar, and David Stringer-Calvert. PVS: an experience report. In Dieter Hutter, Werner Stephan, Paolo Traverso, and Markus Ullman, editors, *Applied Formal Methods—FM-Trends 98*, volume 1641 of *Lecture Notes in Computer Science*, pages 338–345, Boppard, Germany, October 1998. Springer-Verlag. URL: http://www.csl.sri.com/papers/fmtrends98/.

28 Sam Owre and Natarajan Shankar. The formal semantics of PVS. Technical Report SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA, 1997.

29 Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 221–230. IEEE Computer Society, 2001. doi:10.1109/LICS.2001.932499.

30 John M. Rushby, Sam Owre, and Natarajan Shankar. Subtypes for specifications: Predicate subtyping in PVS. *IEEE Trans. Software Eng.*, 24(9):709–720, 1998. doi:10.1109/32.713327.

31 Anne Salvesen and Jan M. Smith. The strength of the subset type in martin-löf's type theory. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88), Edinburgh, Scotland, UK, July 5-8, 1988*, pages 384–391. IEEE Computer Society, 1988. doi:10.1109/LICS.1988.5135.

32 Matthieu Sozeau. Subset coercions in coq. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, volume 4502 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2006. doi:10.1007/978-3-540-74464-1_16.

33 The Agda Team. *Agda Manual*. URL: https://agda.readthedocs.io/.

34 F. Thiré. *Interoperability between proof systems using the Dedukti logical framework*. PhD thesis, Université Paris-Saclay, France, 2020.

35 F. Thiré and G. Férey. Proof irrelevance and predicate subtyping in dedukti. https://eutypes.cs.ru.nl/eutypes_pmwiki/uploads/Main/books-of-abstracts-TYPES2019.pdf, p. 106, 2019. Abstract of a talk given at the TYPES conference.

36 Benjamin Werner. On the strength of proof-irrelevant type theories. *Log. Methods Comput. Sci.*, 4(3), 2008. doi:10.2168/LMCS-4(3:13)2008.