Expressing predicate sub-typing in $\lambda\Pi$ modulo theory

Gabriel Hondet
Université Paris-Saclay, ENS Paris-Saclay, CNRS, Inria, Laboratoire Spécification et Vérification, 94235, Cachan, France
gabriel.hondet@inria.fr

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 engenders redundancy as each assistant is likely to ship its own version of a proof. It also prevents adoption of formal methods by industries because of the lack of standards and the difficulty to use adequately formal methods.

PVS is one of these proof assistants. More precisely, PVS is an environment comprising a specification language, a type checker and a theorem prover. One of the specificity of PVS is to be able to fuse type checking with theorem proving by requiring terms to validate arbitrary predicates in order to be attributed a certain type. This ability to type terms only if they validate some predicate is designated *predicate sub-typing* [6] and lifts the notion of set-theoretic subsets to type systems.

To bridge the gaps between formal method communities, the Logipedia project aims at providing an encyclopedia of formal proofs where proofs are independent from the system they were developed in. Logipedia relies on the encoding of proofs into $\lambda\Pi$-calculus modulo theory, an extension of Edinburgh logical framework abbreviated $\lambda\Pi/R$, and one of its implementations, Dedukti. The $\lambda\Pi$-calculus is itself an extension of simply typed λ calculus with dependent types allowing to write, for instance the type of vectors of a certain length, $\forall n : \mathbb{N}, V(n)$. $\lambda\Pi/R$ extends $\lambda\Pi$-calculus computation by the possibility to use arbitrary reduction rules instead of being limited to the β reduction of λ-calculus.

The present work proposes an encoding of the fundamental features of predicate sub-typing as used in PVS into Dedukti. All presented work is available as Dedukti files at `https://github.com/Deducteam/personoj`.

State of the art F. Gilbert formalised predicate sub-typing as a system named PVS-Cert in [3]. PVS-Cert can be decomposed in layers: the first layer is λ-HOL described in [1]. On top of it, predicate sub-typing is added as a particular case of dependent pairs [4]. In this system, the type of the dependent pair is noted $\{x : A \mid a\}$ where A is a type and a is a predicate that can depend on x. An element of this type is a pair $\langle t, h\rangle$ where t is of type A and h is a proof that $a[x \mapsto t]$ is true.

Contribution We are able to encode PVS-Cert following the same layered structure. The encoding of λ-HOL is taken from [2] and adding dependent pairs raised no difficulty. On the other hand, the reduction rules of PVS-Cert required more attention. Given a predicate `evenp` and two terms $\langle 2, h\rangle$ and $\langle 2, h'\rangle$ with h and h' two proofs that 2 validates `evenp`, we need *proof irrelevance* on the second argument of pairs (h and h') to avoid introducing two different terms representing the "even number 2" that differ only by the proof that 2 is even. Encoding proof irrelevance required to add a notion of *protected symbols* into Dedukti.

To appear in EPTCS.

The encoding is put to test on a specification of rational numbers[1]. In particular, we define $\mathbb{N}^* = \{n : \mathbb{N} \mid n \neq 0\}$ and $\cdot/\cdot : \mathbb{N} \to \mathbb{N}^* \to \mathbb{Q}^+$ to finally propose the proof of the theorem $\forall a \in \mathbb{N}, \forall b \in \mathbb{N}^*, \frac{a}{b} \times \frac{b}{1} = \frac{a}{1}$.

Programming with sub-typing usually involves sub-type polymorphism. PVS is no exception but while in programming languages the sub-typing relation is decidable, it is not in PVS. The use of sub-typing polymorphism requires the fulfillment of proof obligations —called *TCC* for Type-Correctness Conditions— that are not decidable in general. The typing rules for sub-type polymorphism are given in [5] and their encoding is not trivial. It requires the encoding of several notions of casting between types of PVS and the generation of TCCs. In the end, sub-type polymorphism is handled by a *cast* operator that allows to type a term t of type A as B provided that the two following TCCs are proved, (i) A and B have the same "top-type", (ii) all predicates that are required to be of type B are verified by t.

Type checking terms with predicate sub-typing can require to prove formulæ that may depend on the context they are in. For instance, the term $\forall x, \frac{1}{x} = \frac{1}{x}$ is not type correct in PVS since we cannot prove that $x \in \mathbb{N}^*$ while $\forall x, x \neq 0 \Rightarrow \frac{1}{x} = \frac{1}{x}$ is. PVS uses a notion of *logical context* in which terms are type-checked that is transcribed into our encoding.

Encoding PVS-Cert, sub-type polymorphism and logical contexts lay the foundations to the automatic translation of PVS specifications with their proofs to Dedukti. Such a translator is currently being developed[2]. Further work is needed to translate specifications in their entirety. The proofs of PVS are performed in a classical sequent calculus and use complex tactics that cannot be translated directly into Dedukti. Many TCCs generated during type-checking appear to be, if not trivial, automatically solvable by looking at already defined theorems. While Dedukti 3 has a notion of unsolved goals, it is not capable yet of solving automatically these goals. A first step would be to implement a Prolog-like search among defined symbols to instantiate these goals.

References

[1] Henk Barendregt & Kees Hemerik (1990): *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*, *LNCS* 432, Springer, pp. 1–35, doi:10.1007/3-540-52592-0_53.

[2] Ali Assaf and Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant & Ronan Saillard: *Dedukti: a Logical Framework based on the λΠ-Calculus Modulo Theory*.

[3] Frederic Gilbert (2018): *Extending higher-order logic with predicate subtyping : application to PVS*. Theses, Université Sorbonne Paris Cité. Available at `https://tel.archives-ouvertes.fr/tel-02058937`.

[4] Zhaohui Luo (1990): *An extended calculus of constructions*. Ph.D. thesis, University of Edinburgh, UK. Available at `http://hdl.handle.net/1842/12487`.

[5] Sam Owre & Natarajan Shankar (1997): *The Formal Semantics of PVS*. Technical Report SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA.

[6] John M. Rushby, Sam Owre & Natarajan Shankar (1998): *Subtypes for Specifications: Predicate Subtyping in PVS*. *IEEE Trans. Software Eng.* 24(9), pp. 709–720, doi:10.1109/32.713327.

[1] `https://github.com/Deducteam/personoj/blob/master/paper/rat.lp`

[2] `https://github.com/Deducteam/PVS/tree/prettyprint-dedukti`