Dedukti: a Universal Proof Checker

Ronan Saillard
MINES ParisTech
ronan.saillard@cri.ensmp.fr

Context The success of formal methods both as tools of practical importance and as objects of intellectual curiosity, has spawned a bewildering variety of software systems to support them. While the field has developed to maturity in academia and has registered some important successes in the industry, the full benefit of formal methods in an industrial setting remains largely untapped. We submit that a lack of standards and easy interoperability in the field is one explanation to this state of affairs.

The $\lambda\Pi$-calculus modulo To address this issue we propose the $\lambda\Pi$-calculus modulo as a universal proof language. This calculus, introduced by Cousineau and Dowek [5], is a dependent typed λ-calculus where the definitional equality has been generalized to an arbitrary congruence generated by rewrite rules. Our opinion is that this formalism is well suited for encoding foreign logics and make them cooperate. Indeed, by allowing the addition of user-defined rewrite rules, logics benefit from shallower encodings and do not lose their computational content.

To support this language we have developed a new type checker called Dedukti [3].

A type-checker generator In order to provide an efficient and yet simple type checker for the $\lambda\Pi$-calculus modulo we chose to implement it as a type checker generator: a parser/front-end generates a specialized type checker in a target language which is then run. This scheme has two key advantages:

First it conveniently allows us to sidestep the problem of implementing alpha-equivalence and substitution by making use of *Higher Order Abstract Syntax* (HOAS).

Second, it permits us to make use of *normalization by evaluation* (NbE) which is one particularly efficient way to implement normalization scheme that alternates phases of weak reduction (*i.e.* evaluation) and reification phases (or readback phases [6]). The idea is to use the application of the target language to code the β-reduction or any other rewrite rules.

A Just-in-Time compiler as our backend As we intend to be a universal proof checker we need to be versatile. In particular some logics allow proofs to embed computational content and some do not. This is a real challenge because the efficiency of the normalization strategy highly depends on the computational content of the proof term. Indeed in the first case λ-terms should be compiled and in the latter λ-terms should be interpreted. The matter of choosing the right strategy is very difficult but it is already adressed by the *Just-in-Time* compiler community, so we chose to rely on the state-of-the-art *LuaJIT* [8] compiler, as our backend.

A Bidirectional and Context-Free type checking We use a domain-free representation of terms as introduced in [1], meaning that abstractions may not be annotated by the type of the variable that they bind. This information is not necessary if one adopts a bidirectional typing discipline, a technique going back to [4] which splits the usual type judgments into two forms: checking judgments and synthesis judgments. Hence, input terms are smaller at no cost to the complexity or size of the type checker.

When using HOAS, abstractions are encoded as functions of the implementation language. They are therefore opaque structures that cannot be directly analyzed. Functions are black boxes whose only interface is that they can be applied. In HOAS, it is therefore necessary to draw the parameter/variable distinction common in presentations of first-order logic — variables are always bound and parameters are variables that are free. This distinction is made explicit by the use of context-free type checking [2].

Features and Translators Dedukti features also dot-pattern *à la Agda* [7] which permits to avoid type checking subterms well-typed by construction, opaque definitions and a basic notion of modules. We provide a translator to the $\lambda\Pi$-calculus modulo for the Calculus of Inductive Construction as implemented by the *Coq* proof assistant, and a translator for the Higher Order Logic as implemented by *HOL/Isabelle*. We plan to develop translations for the logic of *PVS* and the set theory of *Atelier B*.

References

[1] Gilles Barthe and Morten Heine Sørensen. Domain-free pure type systems. In *J. Funct. Program.*, pages 9–20. Springer, 1993.

[2] Mathieu Boespflug. *Conception d'un noyau de vérification de preuves pour le $\lambda\Pi$-calcul modulo*. PhD thesis, Ecole polytechnique, January 2011.

[3] Mathieu Boespflug, Quentin Carbonneaux, Olivier Hermant, and Ronan Saillard. Dedukti : https://www.rocq.inria.fr/deducteam/dedukti/index.html.

[4] Thierry Coquand. An algorithm for type-checking dependent types. *Sci. Comput. Program.*, 26(1-3):167–177, 1996.

[5] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *TLCA*, volume 4583 of *LNCS*, pages 102–117. Springer, 2007.

[6] B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In Mitchell Wand and Simon L. Peyton Jones, editors, *ICFP*, pages 235–246. ACM, 2002.

[7] Ulf Norell. Dependently typed programming in agda. In Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra, editors, *Advanced Functional Programming*, volume 5832 of *LNCS*, pages 230–266. Springer, 2008.

[8] The LuaJIT Project. http://www.http://luajit.org/.