Development of a proof assistant for Dedukti

- Lab: LSV, ENS Cachan, France
- Team: Deducteam
- Advisor: Frédéric Blanqui

Dedukti [11] is a formal proof checker based on a logical framework called the $\lambda\Pi$-calculus modulo, which is an extension of the simply-typed lambda-calculus with dependent types (e.g. lists of size n) and an equivalence relation on types generated by user-defined rewrite rules (like in Agda or Haskell). Proofs obtained by some proof assistants (e.g. HOL, Coq, Matita) can be checked in Dedukti by encoding function definitions and axioms by rewrite rules [8, 6, 5].

But, currently, no proof assistant fully uses all the capabilities of Dedukti, which allows a priori arbitrary user-defined rewrite rules. This is for instance necessary if one wants to ease the use of dependent types and be able to define types for representing simplicial sets of arbitrary dimensions, ∞-categories or models of Voevodsky's homotopy type theory.

The goal of this internship is to develop a front-end, that is, a proof assistant, for Dedukti that takes advantage of defining arbitrary rewrite rules for defining functions and types. Developing a proof assistant includes to develop a language and interpretation tool for building proofs interactively.

A key feature to scale up, especially with dependent and polymorphic types, is to allow the user to write down terms with missing information (e.g. the type of the elements of a list) and provide an inference engine for deducing it. To start with, the student could adapt the refinement engine of Matita [3].

Such a refinement engine is based on a unification algorithm. To start with, the student could implement a simple first-order unification algorithm.

A refinement engine also provides the basis on which to implement basic tactics. For instance, applying the logical introduction rule for implication consists in refining the current proof by the incomplete term λx :?.?. The student will implement a basic set of such tactics.

Then, several directions can be considered:

- Take into account in the unification algorithm of user-defined rules and unification hints like in Matita and Coq for handling type classes and canonical structures [1, 12, 10].
- Provide a tactic to export the current goal into the standard TPTP format and call any state-of-the-art automated theorem provers to solve it (with

a preference for those that can output some proof that can be checked by Dedukti like Zenon and iProver).

- Define a general tactic language following works like Isar [14], Tinycals [7, 2], Ssreflect [9], MTac [17] or the new implementation of LTac [13].
- Develop a user interface by developing a ProofGeneral mode [4] or a jEdit mode [16, 15].

Expected abilities: basic knowledge of typed lambda-calculus and OCaml.

References

[1] A. Asperti, W. Ricciotti, C. Sacerdoti Coen, and E. Tassi. Hints in unification. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 5674, 2009.

[2] A. Asperti, W. Ricciotti, C. Sacerdoti Coen, and E. Tassi. A new type for tactics. In *Proceedings of the ACM SIGSAM International Workshop on Programming Languages for Mechanized Mathematics Systems*, 2009.

[3] A. Asperti, W. Ricciotti, C. Sacerdoti Coen, and E. Tassi. A bi-directional refinement algorithm for the calculus of (co)inductive constructions. *Logical Methods in Computer Science*, 8:1–49, 2012.

[4] D. Aspinall. Proof General: a generic tool for proof development. In *Proceedings of the 6th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1785, 2000.

[5] A. Assaf. *A framework for defining computational higher-order logics*. PhD thesis, École Polytechnique, France, 2015.

[6] M. Boespflug, Q. Carbonneaux, and O. Hermant. The lambda-pi-calculus modulo as a universal proof language. In *Proceedings of the 2nd International Workshop on Proof eXchange for Theorem Proving*, CEUR Workshop Proceedings 878, 2012.

[7] C. Sacerdoti Coen, E. Tassi, and S. Zacchiroli. Tinycals: step by step tacticals. In *Proceedings of the 7th*, Electronic Notes in Theoretical Computer Science 174(2), 2006.

[8] 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.

[9] G. Gonthier, A. Mahboubi, and E. Tassi. A small scale reflection extension for the Coq system. Technical Report 6455 version 15, INRIA and Microsoft Research, 2014.

[10] A. Mahboubi and E. Tassi. Canonical Structures for the working Coq user. In *Proceedings of the 4th International Conference on Interactive Theorem Proving*, Lecture Notes in Computer Science 7998, 2013.

[11] R. Saillard. Dedukti 2.3, 2014.

[12] M. Sozeau. A new look at generalized rewriting in type theory. *Journal of Formalized Reasoning*, 2(1):41–62, 2009.

[13] A. Spiwack. An abstract type for constructing tactics in Coq, 2010.

[14] M. Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 1690, 1999.

[15] M. Wenzel. Isabelle/jEdit – a prover IDE within the PIDE framework. In *Proceedings of the 11th International Conference on Intelligent Computer Mathematics*, Lecture Notes in Computer Science 7362, 2012.

[16] M. Wenzel. READ-EVAL-PRINT in parallel and asynchronous proof-checking. In *Proceedings of the 10th*, Electronic Proceedings in Theoretical Computer Science 118, 2012.

[17] B. Ziliani, D. Dreyer, N. R. Krishnaswami, A. Nanevski, and V. Vafeiadis. Mtac: a monad for typed tactic programming in Coq. In *Proceedings of the 18th ACM International Conference on Functional Programming*, SIGPLAN Notices 48(9), 2013.