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, $\infty$-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 $\lambda x : ? : ?$. 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


