Proof Tactics in Dedukti

Antoine Defourné
DEDUCI-EAM (INRIA) - ENSIMAG
antoine.defourne@inria.fr

Introduction

Dedukti is a type-checker for the λΠ-calculus modulo theories, a formal language combining the strengths of dependent types and rewriting systems to express and prove theorems in many different logical frameworks.

λΠ-calculus modulo theories

Why3 is a logical framework that is mainly intended to serve as a platform between external automated provers. Its language, Why3ML, can express programs, specifications and logical formulas in a first-order polymorphic langage.

More importantly, Why3 is mainly intended to serve as a platform between provers. One can use Why3 to solve a goal automatically by calling an external prover (19 are available to this day).

A why3 tactic would translate any goal Γ ⊢ G : T to a Why3 goal, then ask Why3 to solve this goal. The challenge is to translate as much Dedukti goals as possible in a way that is sound.

Up to now, the why3 tactic is able to:
- Translate terms of λΠ into the minimal first-order logic if possible;
- Recognize deep embeddings by relying on user declarations via a special command;
- Using those deep embeddings, it is actually possible to express formulas of first-order logic or arithmetic in Dedukti and still translate them to Why3.

Why3

Why3 code is structured in theories. Each theory contains declarations and one or more goals.
- There are several kinds of declarations: types, functions, axioms, etc.
- All terms are typed in Why3. In particular, predicates are functions with a special output type (a "Prop" type).

More generally, Why3 code is is structured in theories. Each theory contains declarations and one or more goals.

My Work

Demon is a clone of Dedukti to be used interactively, using tactics to build proofs and solve goals. This tool is an important step towards making Dedukti a complete proof assistant. As for now, there are a few tactics available in Demon; my work is to implement two new tactics in Demon to make it easier to solve some goals.

Theorem Proving in Dedukti

Shallow Embedding follows from the idea that types can be seen as propositions. For every term in a language of the minimal, intuitionistic first-order logic, there is a corresponding term in Dedukti.

Deep Embedding is necessary for more expressive logics. Traditionally, we define some type Prop : Type then an operator ε : Prop → Type, and work in Prop where we can define new operators using rewrite rules.

Below is an implementation of the conjunction ∧ in Dedukti:

\[ \text{Prop} : \text{Type} \]
\[ \text{and} : \text{Prop} \to \text{Prop} \to \text{Prop} \]
\[ (P, Q) \text{ and } P \to Q \to (\text{and } P \to Q) \to \text{and } R. \]

Rewrite

We are in the process of proving \( P \) under an equality hypothesis \( H_{eq} : \Gamma, H_{eq} : \forall x_1, \ldots, x_n, e_i = e_i \vdash G : P \).

Let us suppose there is an instance of \( e_i \) in \( P \), that is to say there exists a position \( p \in P \) and a substitution \( \sigma \) of domain \( x_1, \ldots, x_n \) so that \( P[\sigma] = e_i \sigma \). Then we would like to replace this instance by \( e_i \sigma \) in the goal, using \( H_{eq} \):

\[ \Gamma, H_{eq} \vdash \forall x_1, \ldots, x_n. e_i \sigma = e_i \sigma P(x) \]

To do this, we must start from a definition of equality. In this case, a variant of \textbf{Leibniz’ equality}, which can be seen as an induction principle: given a proposition \( P \) and an element \( x \), if \( P(x) \) then \( P(y) \) for all \( y \) equal to \( x \).

\[ \text{eq} \text{ind} : \forall A \text{ Type}, x:A, P(x). P(x) \Rightarrow (\forall y:A, x \equiv y \Rightarrow P(y)) \]

The proof term needed to get \( G \) from \( G' \) would then be:

\[ \text{eq} \text{ind} A (e_i, \sigma) (A x:A. P[x]) G' (e_i \sigma) (H_{eq} \sigma(x_1) \ldots \sigma(x_n)) \]

The remaining work is twofold:
- First, prove the soundness of this proof, that the term above has type \( P \) in all cases.
- Second, implement this as a rewrite tactic in Demon.