Proof Tactics in Dedukti

Antoine Defourné

DEDUC⊢EAM (INRIA) - ENSIMAG

antoine.defourne@inria.fr

Introduction

Dedukti is a **type-checker** for the $\lambda\Pi$-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.

$\lambda\Pi$-calculus modulo theories

```
#NAME example.

A : Type.

Nat : Type.
0 : Nat.
S : Nat -> Nat.

(; Vec is a dependent type: the type of vectors of some length. ;)
Vec : Nat -> Type.
nil : Vec 0.
cons : n:Nat -> A -> Vec n -> Vec (S n).

(; The tail function for vectors is defined with rewriting. ;)
def tail : n:Nat -> Vec (S n) -> Vec n.
[v] tail _ (cons _ _ v) --> v.
```

Theorem Proving in Dedukti

Shallow Embedding follows from the idea that *types can be seen as propositions*. For every term in a langage 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 $\epsilon : Prop \to Type$, and work in $Prop$ where we can define new operators using rewrite rules. Below is an implementation of the conjunction $\wedge$ in *Dedukti*:

```
Prop : Type.
def eps : Prop -> Type.

and : Prop -> Prop -> Prop.
[P,Q] eps (and P Q) --> R:Prop -> (eps P -> eps Q -> eps R) -> eps R.
```

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.

Why3

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

```
theory Task
        type set
        constant a : set
        predicate p set
        axiom h : forall x:set. p x
        goal g : exists x:set. p x
end
```

- *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 importantly, *Why3* is mainly intented to serve as a plaform 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 $\Gamma \vdash^? G : T$ 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 $\lambda\Pi$ 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*.

Rewrite

We are in the process of proving P under an equality hypothesis H_{eq} :

$$\Gamma, H_{eq} : \forall x_1, \dots, x_n, e_\ell = e_r \vdash^? G : P$$

Let us suppose there is an instance of e_ℓ in P, that is to say there exists a position $p \in \mathcal{P}\mathrm{os}(P)$ and a substitution σ of domain $x_1, \dots, x_n$ so that $P|_p = e_\ell\sigma$. Then we would like to replace this instance by $e_r\sigma$ in the goal, using H_{eq}.

$$\Gamma, H_{eq} : \dots \vdash^? G' : P[e_r\sigma \backslash e_\ell\sigma]$$

To do this, we must start from a definition of equality. In this case, a variant of **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.

$$\text{eq\_ind} : \forall A^{Type}, x^A, P^{\forall z^A, Prop}, P(x) \Rightarrow \left(\forall y^A, x =_A y \Rightarrow P(y)\right)$$

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

$$\text{eq\_ind}\ A\ (e_r\sigma)\ (\lambda z^A . P[z]_p)\ G'\ (e_\ell\sigma)\ (H'_{eq}\ \sigma(x_1)\ \dots\ \sigma(x_2))$$

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*.