Dedukti: a universal proof checker

Mathieu Boespflug Chantal Keller

INRIA – LIX

October, 19th 2010

Dedukti: a proof checker for $\lambda\Pi$-modulo

Outline

Interoperable proof scripts

- Enable the cooperation of heterogeneous teams on large proof developments (Kepler's conjecture)
- Allow importing large libraries and frameworks from other systems. (ssreflect groups, HOL Light's floating point arithmetic, HOL's Hoare Logic)

A small and modular kernel

- Shared trusted base
- More users of the trusted means higher confidence
- Simple as possible
- Modularity to allow users the flexibility to choose the size of the trusted base
  $\longrightarrow$ *tradeoff size vs flexibility, speed*

The $\lambda\Pi$-modulo calculus

- It's a logical framework
- Very minimal theory: no polymorphism
- Still possible to represent binding using metalanguage binding.
- $\lambda\Pi = \lambda_{\to} +$ dependent types.

Typing rules for $\lambda\Pi$

$s \in \{\mathit{Type}, \mathit{Kind}\}$

$$\frac{}{[\,] \text{ well-formed}}$$

$$\frac{\Gamma \vdash A : s}{\Gamma[x : A] \text{ well-formed}}$$

$$\frac{\Gamma \text{ well-formed}}{\Gamma \vdash \mathit{Type} : \mathit{Kind}}$$

$$\frac{\Gamma \text{ well-formed} \quad x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma[x : A] \vdash B : s}{\Gamma \vdash \Pi x : A\ B : s}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma[x : A] \vdash B : s \quad \Gamma[x : A] \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}$$

$$\frac{\Gamma \vdash t : \Pi x : A\ B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : (u/x)B}$$

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : s \quad \Gamma \vdash t : A}{\Gamma \vdash t : B}\ A \equiv B$$

Why $\lambda\Pi$

- Why no polymorphism?
- Why no inductive types?
- Why no universes? (we'll come back to that later)
- Why not first order (defunctionalized) proof terms ?
- Why not propositional logic?

Why not first order?

Pros:

- First order is simple to implement
- Many more efficient theorem provers in first order setting.

Cons:

- Difficult to represent binding.
- No binding means implementing own substitution, own β-reduction, etc (via explicit substitutions).

See "A first-order representation of pure type systems using superdeduction" (Burel, 2008).

More generally

- Tension between simplicity of encoding and simplicity of proof checker.
- Simpler encoding makes it easier to prove correct, easier to reason about encoded proofs, easier to read encoded proofs.
- simpler proof checker respects de Bruijn criterion.
- simpler proof checker more “essential”.

Another tension

- Simplicity of the native logic of the proof checker and maintaining good properties under addition of encodings.
- In particular, if encoding the foreign logic requires asserting axioms, then in general we lose the witness property, disjunction property, etc, even for theories with cut elimination.
- Need to construct a new semantic model that validates the added axioms to ensure cut elimination, on which consistency is contingent.
- Asserting axioms breaks the computational behaviour of the encoding.

$\lambda\Pi$-modulo

- extends lambda-Pi with user defined rewrite rules.
- Many non-logical, so called "computational" axioms can be replaced with rewrite rules
- In particular, axioms of the form

$$\forall x_1, \ldots, x_n.t = u$$

or

$$\forall x_1, \ldots, x_n.P \Leftrightarrow Q$$

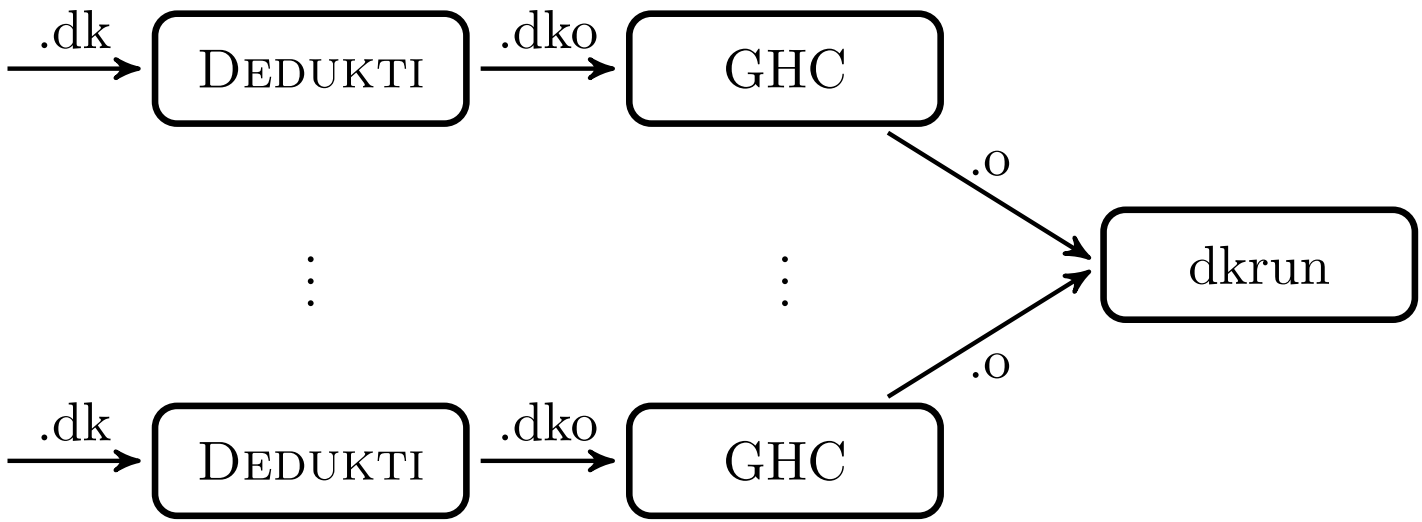
can be replaced by rewrite rules of the form $t \hookrightarrow u$ or $P \hookrightarrow Q$.

- conserve witness property, etc.
- For purely computational theories, have a single notion of cut.

Dedukti

- Dedukti: modular proof checker based on lambda-Pi-modulo.
  - Does only proof checking
  - No confluence checking, termination checking of rewrite rules.
- Dedukti: small proof checker based on lambda-Pi-modulo.
  - maps proof script to a functional program
  - No custom implementation of beta reduction necessary.
  - No custom implementation of rewrite rules necessary.

Proof checking $\lambda\Pi$-modulo scripts



Encoding the CIC

Encoding polymorphism

```
o : Type.

nat : Type.                    bool : Type.
O : nat.                       true : bool.
S : nat → nat.                 false : bool.
ṅat : o.                       ḃool : o.

ϵ : o → Type.
[] ϵ ṅat ↪ nat.
[] ϵ ḃool ↪ bool.

vec : A : o → nat → Type.
nil : A : o → vec A O.
cons : A : o → x : nat → ϵ A → vec A x →
     vec A (S x).
```

Embedding of PTS in $\lambda\Pi$-modulo

sort

s

universe type

$U_s : Type$

decoding function

$\epsilon_s : U_s \to Type$

axiom

$\langle s_1, s_2 \rangle$

declaration

$\dot{s_1} : U_{s_2}$

rule

$\langle s_1, s_2, s_3 \rangle$

declaration

$\dot{\Pi}_{\langle s_1, s_2, s_3 \rangle} : \Pi x : U_{s_1} : (((\epsilon_1\ x) \to U_{s_2}) \to U_{s_3})$

rewrite rule

$\epsilon_{s_3}\ (\dot{\Pi}_{\langle s_1, s_2, s_3 \rangle}\ x\ y) \hookrightarrow \Pi x : (\epsilon_{s_1}\ x)\ (\epsilon_{s_2}\ (y\ x))$

Example: Calculus of Constructions

$$
\begin{aligned}
\epsilon_{Kind}\ (\dot{Type}) &\hookrightarrow U_{Type} \\
\epsilon_{Type}\ (\dot{\Pi}_{\langle Type, Type, Type\rangle}\ x\ y) &\hookrightarrow \Pi x : (\epsilon_{Type}\ x)\ (\epsilon_{Type}\ (y\ x)) \\
\epsilon_{Kind}\ (\dot{\Pi}_{\langle Type, Kind, Kind\rangle}\ x\ y) &\hookrightarrow \Pi x : (\epsilon_{Type}\ x)\ (\epsilon_{Kind}\ (y\ x)) \\
\epsilon_{Kind}\ (\dot{\Pi}_{\langle Kind, Type, Kind\rangle}\ x\ y) &\hookrightarrow \Pi x : (\epsilon_{Kind}\ x)\ (\epsilon_{Type}\ (y\ x)) \\
\epsilon_{Kind}\ (\dot{\Pi}_{\langle Kind, Kind, Kind\rangle}\ x\ y) &\hookrightarrow \Pi x : (\epsilon_{Kind}\ x)\ (\epsilon_{Kind}\ (y\ x))
\end{aligned}
$$

Calculus of Inductive Constructions

```
Inductive T x_1 ... x_n : s :=
| C_1 : ... -> ... -> T t_1_1 ... t_1_n
|...
| C_m : ... -> ... -> T t_m_1 ... t_m_n
```

- Definition schema adds new declarations for each constructor to environment.
- In addition adds induction principles.
- Induction principles defined in termes of fixpoints.
  $\longrightarrow$ *Can be seen as adding extra reduction rules to the calculus for each induction principle.*

Encoding of constructors (first attempt)

```
Inductive T x_1 ... x_n : s :=
| C_1 : ... -> ... -> T t_1_1 ... t_1_n
|...
| C_m : ... -> ... -> T t_m_1 ... t_m_n
```

$$\mathsf{T} : \ldots \to \ldots \to U_s$$
$$C_1 : \ldots \to \ldots \to \mathsf{T}\ t_{11} \ \ldots \ t_{1n}$$
$$C_m : \ldots \to \ldots \to \mathsf{T}\ t_{m1} \ \ldots \ t_{mn}$$

Encoding of fixpoints and match constructs

```
Fixpoint F (x : T) := match x with
| C_1 x_1_1 ... x_1_n => E_1
|...
| C_1 x_m_1 ... x_m_n => E_m
end
```

$$[x_{11} : T_{11}, \ldots, x_{1n} : T_{1n}]$$
$$\mathsf{F}\ C_{11}\ x_{11}\ \ldots\ x_{1n} \hookrightarrow E_1$$
$$\ldots$$
$$[x_{m1} : T_{m1}, \ldots, x_{mn} : T_{mn}]$$
$$\mathsf{F}\ C_{m1}\ x_{m1}\ \ldots\ x_{mn} \hookrightarrow E_m$$

Solution: box all constructors

```
nat_constr : nat' → nat.
O : nat'.
S : nat → nat'.
```

Then, natural numbers are translated as:

$$|O| = nat_{constr}\, O \quad |Sx| = nat_{constr}(S|x|)$$

Presentation

Purpose:

- well-understood logical framework
- many LCF-like theorem provers
- in particular HOL Light

Joint work with Paul Brauner

Main idea

Main idea:

- embedding using HOAS
- translation of HOL propositions into Dedukti terms
- proofs as rewriting rules

HOL as a first order theory [Dowek et al. 1999]

Simple types:

- o
- $a \to b$

Simply-typed terms:

- x, $\lambda x.t$, $t\ u$
- constants: $=$, logical connectives

A set of inference rules:

$$\frac{}{\vdash t = t} \qquad \frac{\Gamma \vdash p \Leftrightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

Zoom on inference rules

Inference rules:

$$\frac{}{\vdash t = t} \qquad \frac{\Gamma \vdash p \Leftrightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

Theorem $\Gamma \vdash p$:

- Γ: set of terms of type o
- p: term of type o

- LCF : theorems as inhabitants of an abstract data-type

Types and terms [Schürmann and Stehr 2006]

Types:

```
htype : Type.
  o : htype.
  ↦ : htype → htype → htype.
```

Terms:

```
hterm : htype → Type.
  λ : (hterm a → hterm b) → hterm (a ↦ b).
  α : hterm (a ↦ b) → hterm a → hterm b.
  ·→ : hterm (o ↦ (o ↦ o)).
  ·∀ : hterm ((a ↦ o) ↦ o).
  ·= : hterm (a ↦ (a ↦ o)).
```

β-reduction

β-reduction:

$$[\mathsf{a}\ \mathsf{b}\ \mathsf{f}\ \mathsf{x}]\ \alpha\ (\lambda\ \mathsf{f})\ \mathsf{x} \hookrightarrow \mathsf{f}\ \mathsf{x}\,.$$

Notation:

$$\alpha\ \mathsf{f}\ \mathsf{t}\ \mathsf{u} \triangleq \alpha\ (\alpha\ \mathsf{f}\ \mathsf{t})\ \mathsf{u}$$

ε translation

ε: hterm o $\to$ Type

- unary predicate symbol
- transforms a term *t* of type *o* into a the proposition ε *t*
- defines a translation from the embedding into Dedukti terms
- gives an elimination scheme to each constant

Translation:

```
[a b] ε(α ⇢ a b) ↪ ε a → ε b.
[a f] ε (α ∀̇ (λ f)) ↪ b: hterm a → ε (f b).
[a x y] ε (α ≐ x y) ↪
  P: hterm (a ↦ o) → ε (α P x) → ε (α P y).
```

Example of an inference rule

$$\frac{}{\vdash t = t}$$

Encoding:

refl : a: htype $\rightarrow$ t: hterm a $\rightarrow \varepsilon$ $(\alpha \doteq$ t t).
[a t] refl a t $\hookrightarrow$?

Remember:

[a x y] ε $(\alpha \doteq$ x y) $\hookrightarrow$
P: hterm (a $\mapsto$ o) $\rightarrow \varepsilon$ (α P x) $\rightarrow \varepsilon$ (α P y).

Example of an inference rule

$$\frac{}{\vdash t = t}$$

Encoding:

$\mathsf{refl} : \mathsf{a} : \mathsf{htype} \to \mathsf{t} : \mathsf{hterm}\ \mathsf{a} \to \varepsilon\ (\alpha \doteq \mathsf{t}\ \mathsf{t}).$

$[\mathsf{a}\ \mathsf{t}]\ \ \mathsf{refl}\ \ \mathsf{a}\ \ \mathsf{t} \hookrightarrow \mathsf{P} : \mathsf{hterm}\ (\mathsf{a} \mapsto \mathsf{o}) \Rightarrow \mathsf{h} : \varepsilon\ (\alpha\ \ \mathsf{P}\ \mathsf{t}) \Rightarrow ?$

Remember:

$$[\mathsf{a}\ \ \mathsf{x}\ \ \mathsf{y}]\ \ \varepsilon\ \ (\alpha \doteq\ \mathsf{x}\ \ \mathsf{y}) \hookrightarrow$$
$$\mathsf{P} : \mathsf{hterm}\ \ (\mathsf{a} \mapsto \mathsf{o}) \to \varepsilon\ \ (\alpha\ \ \mathsf{P}\ \ \mathsf{x}) \to \varepsilon\ \ (\alpha\ \ \mathsf{P}\ \ \mathsf{y}).$$

Example of an inference rule

$$\frac{}{\vdash t = t}$$

Encoding:

$$\mathsf{refl} : \mathsf{a} : \mathsf{htype} \to \mathsf{t} : \mathsf{hterm}\ \mathsf{a} \to \varepsilon\ (\alpha \doteq \mathsf{t}\ \mathsf{t}).$$
$$[\mathsf{a}\ \mathsf{t}]\ \ \mathsf{refl}\ \ \mathsf{a}\ \ \mathsf{t} \hookrightarrow \mathsf{P} : \mathsf{hterm}\ (\mathsf{a} \mapsto \mathsf{o}) \Rightarrow \mathsf{h} : \varepsilon\ (\alpha\ \ \mathsf{P}\ \mathsf{t}) \Rightarrow \mathsf{h}.$$

Remember:

$$[\mathsf{a}\ \ \mathsf{x}\ \ \mathsf{y}]\ \ \varepsilon\ \ (\alpha \doteq\ \mathsf{x}\ \ \mathsf{y}) \hookrightarrow$$
$$\mathsf{P} : \mathsf{hterm}\ \ (\mathsf{a} \mapsto \mathsf{o}) \to \varepsilon\ \ (\alpha\ \ \mathsf{P}\ \ \mathsf{x}) \to \varepsilon\ \ (\alpha\ \ \mathsf{P}\ \ \mathsf{y}).$$

Inference rules

- Every inference rule can be proved that way
- Need axioms:
  - extensionality
  - classical logic
  - axiom of choice

A few words about HOL Light

HOL Light:

- LCF-style theorem prover
- theorems are ML objects of an abstract data-type thm
- constructed only using given primitives (inference rules)

Prototype under development:

- recording HOL Light proof terms [Obua 2006]
- exporting:
  - theorems as constants of a certain type
  - proofs as rewriting rules using the encoding of inference rules

Conclusion and future work

- Encode more theories!
- Encode universe hierarchies and universe polymorphism.
- Elaborate safe encoding methodologies for combining proofs, formulae from distinct theories.