Toward explicit rewrite rules in the $\lambda\Pi$-calculus modulo

Ronan Saillard (Olivier Hermant)

Deducteam INRIA

MINES ParisTech

December 14th, 2013

Motivation

Problem

- Checking, in a trustful way, that something is a valid proof in a given logic.
- Encoding proofs of any logic in a single compact format.
- Making these encodings interoperate.

Solution

Using the $\lambda\Pi$-calculus modulo, a language based on dependent types and rewriting, as a universal proof language.

In this Talk

- A new presentation of $\lambda\Pi$-calculus modulo with explicit rewrite rules.
- The lastest version of Dedukti, a type-checker for the $\lambda\Pi$-calculus modulo.

Table of Contents

Introduction

The $\lambda\Pi$-calculus modulo

Extending $\lambda\Pi$-calculus with Rewrite Rules

Typing the Rules

Dedukti

A Type-Checker for the $\lambda\Pi$-calculus modulo

Case Study: the OpenTheory Library

Future Work

Confluence and Termination

The $\lambda\Pi$-calculus modulo

Extending $\lambda\Pi$-calculus with Rewrite Rules

The λΠ-calculus modulo

The λΠ-calculus modulo

An extension of dependent typed λ-calculus ($\lambda\Pi$-calculus aka λP-calculus aka LF) with user-defined rewrite rules.

Example of a Rewrite Rules

[f:$\Pi x^A.B$] **Map** f **Nil** $\hookrightarrow$ **Nil**
[f:$\Pi x^A.B$, a:**A**, l:**ListA**] **Map** f (**Cons** a l) $\hookrightarrow$ **Cons** (f a) (**Map** f l)

Typing Rules for λΠ-calculus modulo

$$\text{(Empty)}\ \frac{}{\emptyset\ \mathbf{wf}} \qquad \text{(Dec)}\ \frac{\Gamma\ \mathbf{wf} \qquad \Gamma \vdash A : s \qquad x \notin \Gamma}{\Gamma(x : A)\ \mathbf{wf}}$$

$$\text{(Rewrite)}\ \frac{\Gamma\ \mathbf{wf} \qquad \text{"the rule is well-typed"}}{\Gamma([\Delta]l \hookrightarrow r)\ \mathbf{wf}}$$

$$\text{(Type)}\ \frac{\Gamma\ \mathbf{wf}}{\Gamma \vdash \mathbf{Type} : \mathbf{Kind}} \qquad \text{(Var)}\ \frac{\Gamma\ \mathbf{wf} \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A}$$

$$\text{(App)}\ \frac{\Gamma \vdash t : \Pi x^A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B[x/u]} \qquad \text{(Conv)}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s \qquad A \equiv_{\beta\Gamma} B}{\Gamma \vdash t : B}$$

$$\text{(Abs)}\ \frac{\Gamma \vdash A : \mathbf{Type} \qquad \Gamma(x : A) \vdash t : B \qquad B \neq \mathbf{Kind}}{\Gamma \vdash \lambda x^A.t : \Pi x^A.B}$$

$$\text{(Prod)}\ \frac{\Gamma \vdash A : \mathbf{Type} \qquad \Gamma(x : A) \vdash B : s}{\Gamma \vdash \Pi x^A.B : s}$$

The $\lambda\Pi$-calculus modulo

Typing the Rules

Subject Reduction

Subject Reduction

If $\Gamma \vdash t : T$ and $t \rightarrow_{\beta\Gamma} t'$ then $\Gamma \vdash t' : T$.

Well-typed rule (strong version)

$$\text{(Rewrite)} \quad \frac{\Gamma\ \mathbf{wf} \qquad \Gamma\Delta \vdash l : T \qquad \Gamma\Delta \vdash r : T}{\Gamma([\Delta]l \hookrightarrow r)\ \mathbf{wf}}$$

Theorem

if $\rightarrow_{\beta\Gamma}$ is confluent and every rule is well-typed then subject reduction holds.

Problems

- well-typedness and linearity (next slide) are often in conflicting conditions.
- Subject Reduction might be preserved even with a non well-typed rule.

Linearity

Definition

A rewrite rule $[\Delta]l \hookrightarrow r$ is left-linear if the variables in Δ appear at most once in l.

Why should you prefer linear rewrite rules ?

- Non-linear rules are less efficient: conversion tests needed.
- Confluence and Termination of the combination of a non-linear rewrite system with β-reduction are more difficult to prove.

Example: Well-typedness vs Left-linearity

```
Nat: Type.
Z: Nat.
S: Nat -> Nat.
Plus: Nat -> Nat -> Nat.
[n:Nat] Plus Z n --> n
[n:Nat;m:Nat] Plus (S n) m --> S (Plus n m).

A: Type.
Listn: Nat -> Type.
Nil: Listn Z.
Cons: A -> n:Nat -> Listn n -> Listn (S n).

Append: n:Nat -> Listn n -> m:Nat -> Listn m -> Listn (Plus n m).
[m:Nat;l:Listn m] Append Z Nil m l --> l
[n:Nat;l1;Listn n;m:Nat;l2:Listn m] ( * ) --> Cons (S (plus n m)) a (Append n l1 m l2).
```

Two choices for (*)

- **Append** (**S** n) (**Cons** n a l1) m l2 (well-typed but non linear)
- **Append** n2 (**Cons** n a l1) m l2 (linear but not well-typed)

But these two rules match exactly the same **typed** terms !

If σ(**Append** n2 (**Cons** n a l1) m l2) is well-typed then $\sigma n2 \equiv_{\beta\Gamma} S\ \sigma n$

Past Solution: Dot Patterns

Dot Patterns

Append {**S** n} (**Cons** n a l1) m l2 $\hookrightarrow$ ...

The term between { } is used for typing only.

Problem:

Subject Reduction is not preserved anymore.

Example

```
T: Nat -> Type.
a: Nat.
b: T a.
F: x:Nat -> T x.
[] F {a} --> b.
```

Then **F** (**S** a) $\hookrightarrow b$ but **F** (**S** a) has type **T** (**S** a) $\neq$ (**T** a).

Weakening the well-typedness property

Well-typed rule (strong version)

$$\text{(Strong Rewrite)}\quad \frac{\Gamma\ \mathbf{wf} \qquad \Gamma\Delta \vdash l : T \qquad \Gamma\Delta \vdash r : T}{\Gamma([\Delta] l \hookrightarrow r)\ \mathbf{wf}}$$

The following condition is sufficient (and necessary) to preserve subject reduction:

Well-typed rule (weak version)

$$\text{(Weak Rewrite)}\quad \frac{\Gamma\ \mathbf{wf} \qquad \forall \sigma \in \mathcal{S}(\Delta), (\Gamma \vdash \sigma l : T \Rightarrow \Gamma \vdash \sigma r : T)}{\Gamma([\Delta] l \hookrightarrow r)\ \mathbf{wf}}$$

with $\mathcal{S}(\Delta) := \{\sigma | dom(\sigma) = \Delta\}$

This is undecidable !

Typing a rule as a unification problem

well-typedness as a set of equations

Typing a term t consists in inspecting its structure and checking that some equations modulo $\beta\Gamma$ hold.
Thus we can associate to a term t a system of equations $E(t)$ and a term $T(t)$ such that t is typable iff $E(t)$ holds and in this case $T(t)$ is its type.

Well-typed rule (equivalent definition)

$$\frac{\Gamma\ \mathbf{wf} \qquad S(\Delta, E(l)) \subset S(\Delta, E(r) \cup \{T(l) \equiv_{\beta\Gamma} T(r)\})}{\Gamma([\Delta] l \hookrightarrow r)\ \mathbf{wf}}$$

where $S(X, E)$ is the set of solutions of the (higher order) unification problem E with variables in X.

Example

Example

E(**Append** n2 (**Cons** n a l1) m l2) = { $n2 \equiv_{\beta\Gamma} S\ n$ }

E(**Cons** (**S** n) a (**Append** n l1 m l2)) = {}

- let us write **@** for **Append**, **C** for **Cons**, $\mathbb{N}$ for **Nat**, $\mathcal{L}$ for **Listn** and **+** (infix) for **Plus**
- and let the local context Δ be:

$$
\begin{array}{ccc}
\mathrm{n2} : \mathbb{N} & \mathrm{n} : \mathbb{N} & \mathrm{l1} : \mathcal{L}\ \mathrm{n} \\
\mathrm{m} : \mathbb{N} & \mathrm{l2} : \mathcal{L}\mathrm{m} &
\end{array}
$$

Keeping contexts implicit, we have the following typing tree:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mathbf{@}\ \mathrm{n2} : \mathcal{L}(\mathrm{n2}) \to (\mathrm{m}:\mathbb{N}) \to \mathcal{L}(\mathrm{m}) \to \mathcal{L}(\mathrm{n2+m})} \qquad \dfrac{\dfrac{\vdots}{\vdash \mathbf{C}\ \mathrm{n\ a\ l1} : \mathcal{L}(\mathbf{S}\ \mathrm{n})} \qquad \mathbf{n2} \equiv_{\beta\Gamma} \mathbf{S\ n}}{\vdash \mathbf{C}\ \mathrm{n\ a\ l1} : \mathcal{L}(\mathrm{n2})}}{\vdash \mathbf{@}\ \mathrm{n2}\ (\mathbf{C}\ \mathrm{n\ a\ l1}) : (\mathrm{m}:\mathbb{N}) \to \mathcal{L}(\mathrm{m}) \to \mathcal{L}(\mathrm{n2+m})}}{\vdots}}{\vdash \mathbf{@}\ \mathrm{n2}\ (\mathbf{C}\ \mathrm{n\ a\ l1})\ \mathrm{m\ l2} : \mathcal{L}(\mathrm{n2+m})}
$$

Implemented solution

A simple implementation

1. Find an approximation σ of a most general unifier for $E(l)$.
   (ie σ must be a prefix of any solution of $E(l)$).
2. Check that σ is a solution of $E(r) \cup \{ T(l) \equiv_{\beta\Gamma} T(r) \}$.

This solution has been able to deal with every previous use of dot patterns.

Dedukti

A Type-Checker for the $\lambda\Pi$-calculus modulo

Dedukti

Dedukti is

- a type-checker for the $\lambda\Pi$-calculus modulo.
- a proof-checker for your logic (ie a logical framework).
- comparable with the kernel of an ITP.

Dedukti does not

- check that your rewrite rules are true/admissible (think of them as axioms).
- check that your rewrite rules are terminating and confluent.

Remark: Dedukti has been completely re-implemented in OCaml (about 1000 lines of code). (No more Lua...)

Dedukti and Friends

Dedukti is used as a back-end by these tools:

- **Coqine** (Assaf, Burel): an encoding of the Coq's language (the Calculus of Inductive Constructions) into $\lambda\Pi$-calculus modulo.
- **Focalide** (Cauderlier): an extension of Focalize to generate proofs in $\lambda\Pi$-calculus modulo.
- **Holide** (Assaf, Burel): an encoding of HOL into $\lambda\Pi$-calculus modulo.
- **iProver to Dedukti** (Burel): an extension of iProver to generate proofs in $\lambda\Pi$-calculus modulo.
- **Zenonide** (Gilbert): an extension of Zenon to generate proofs in $\lambda\Pi$-calculus modulo.

Dedukti
Case Study: the OpenTheory Library

Holide

OpenTheory (Hurd)

OpenTheory is a proof format designed to share theorems between proof checker of the HOL family. It comes with a standard theory library.

Holide (Assaf, Burel)

Holide is a tool that can encode the OpenTheory's format into the $\lambda\Pi$-calculus or the $\lambda\Pi$-calculus modulo.

Derivation rules of HOL

$$\frac{}{\vdash t = t}\ \text{Refl } t \qquad \frac{\Gamma \vdash t = u}{\Gamma \vdash \lambda x^A.t = \lambda x^A.u}\ \text{AbsThm } x$$

$$\frac{\Gamma \vdash f = g \qquad \Gamma \vdash t = u}{\Gamma \cup \Delta \vdash f\ \ t = g\ \ u}\ \text{AppThm} \qquad \frac{}{\vdash (\lambda x^A.t)\ x = t}\ \text{Beta } x\ t$$

$$\frac{}{\{\phi\} \vdash \phi}\ \text{Assume} \qquad \frac{\Gamma \vdash \phi = \psi \qquad \Delta \vdash \phi}{\Gamma \cup \Delta \vdash \psi}\ \text{EqMp}$$

$$\frac{\Gamma \vdash \phi \qquad \Delta \vdash \psi}{(\Gamma - \{\psi\}) \cup (\Delta - \{\phi\}) \vdash \phi = \psi}\ \text{DeductAntiSym} \qquad \frac{\Gamma \vdash \phi}{[\sigma]\Gamma \vdash [\sigma]\phi}\ \text{Subst } \sigma$$

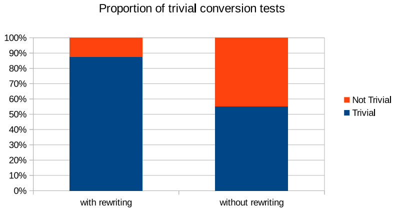
Encoding HOL in the λΠ-calculus

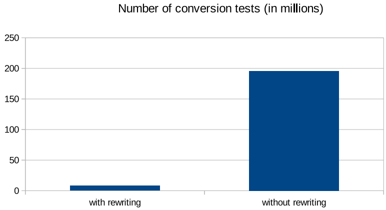
```
#NAME hol_lp

(; HOL Types ;)

type : Type.

bool : type.
ind  : type.
arr  : type -> type -> type.

(; HOL Terms ;)

term : type -> Type.

lam : a : type -> b : type -> (term a -> term b) -> term (arr a b).
app : a : type -> b : type -> term (arr a b) -> term a -> term b.
eq : a : type -> term (arr a (arr a bool)).
select : a : type -> term (arr (arr a bool) a).

EQ : a : type -> term a -> term a -> term bool :=
  a : type => x: term a => y : term a => app a bool (app a (arr a bool) (eq a) x) y.

[...]

(; HOL Proofs ;)

proof : term bool -> Type.

REFL : a : type -> t : term a ->
  proof (EQ a t t).
ABS_THM : a : type -> b: type -> f : (term a -> term b) -> g : (term a -> term b) ->
  (x : term a -> proof (EQ b (f x) (g x))) ->
  proof (EQ (arr a b) (lam a b f) (lam a b g)).

[...]
```

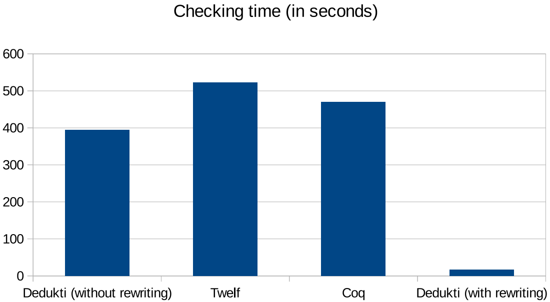
ENCODING HOL IN THE λΠ-CALCULUS MODULO

```
#NAME hol_lpm

(; HOL Types ;)

type : Type.

bool : type.
ind  : type.
arr  : type -> type -> type.

(; HOL Terms ;)

term : type -> Type.
[a : type, b : type] term (arr a b) --> term a -> term b.

eq : a : type -> term (arr a (arr a bool)).
select : a : type -> term (arr (arr a bool) a).




[...]

(; HOL Proofs ;)

proof : term bool -> Type.

REFL : a : type -> t : term a ->
  proof (eq a t t).
ABS_THM : a : type -> b: type -> f : (term a -> term b) -> g : (term a -> term b) ->
  (x : term a -> proof (eq b (f x) (g x))) ->
  proof (eq (arr a b) f g).

[...]
```

Results







Benchmarks obtained on the core package of the OpenTheory library (88 files, 1.4G).

The tests were run on a Linux laptop with a processor Intel Core i7-3520M CPU @ 2.90GHz x 4 and 16GB of Ram.

Future Work
Confluence and Termination

Confluence and Termination (1)

Dedukti's type checking algorithm assumes:

- The Confluence of $\rightarrow_{\beta\Gamma}$.
- The Strong Normalization of $\rightarrow_{\beta\Gamma}$.

Can Dedukti help checking these properties?

Confluence

Criteria for the confluence of $\rightarrow_{\beta\Gamma}$

- $\rightarrow_{\Gamma}$ is weakly orthogonal.
- $\rightarrow_{\Gamma}$ is weakly confluent and $\rightarrow_{\beta\Gamma}$ is terminating.

Future Work

(Weak) orthogonality/confluence detection, critical pair detection, export functionality to (higher-order?) confluence prover.

Termination

Termination of $\rightarrow_{\beta\Gamma}$

- for Object Level/Type Level Rewriting System?
- for First Order/Higher Order Rewriting System?

Criteria

- Modular properties of algebraic pure type systems (Barthe and Geuvers, 1996).
- Definition by rewriting in the Calculus of Constructions (Blanqui, 2005).

Future Work (Termination)

Partial implementation of these criteria.

TOWARD EXPLICIT REWRITE RULES IN THE $\lambda\Pi$-CALCULUS MODULO

Ronan Saillard (Olivier Hermant)

Deducteam INRIA

MINES ParisTech

December 14th, 2013