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 \(\lambda\Pi\)-calculus modulo

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

Example of a Rewrite Rules

\[
[f:\Pi x^A.B] \text{Map } f \text{ Nil } \rightsquigarrow \text{Nil} \\
[f:\Pi x^A.B, a:A, l:\text{ListA}] \text{Map } f \text{ (Cons a l)} \rightsquigarrow \text{Cons } (f \text{ a}) (\text{Map } f \text{ l})
\]
## Typing Rules for $\lambda\Pi$-calculus modulo

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(Empty)</strong></td>
<td>$\emptyset \vdash A : s$</td>
</tr>
<tr>
<td><strong>(Dec)</strong></td>
<td>$\Gamma \vdash A : s \quad x \not\in \Gamma \quad \Gamma(x : A) \vdash B : s$</td>
</tr>
<tr>
<td><strong>(Rewrite)</strong></td>
<td>$\Gamma \vdash \Gamma([\Delta]l \mapsto r) : s$</td>
</tr>
<tr>
<td><strong>(Type)</strong></td>
<td>$\Gamma \vdash \text{Type} : \text{Kind}$</td>
</tr>
<tr>
<td><strong>(Var)</strong></td>
<td>$\Gamma \vdash x : A \quad (x : A) \in \Gamma$</td>
</tr>
<tr>
<td><strong>(App)</strong></td>
<td>$\Gamma \vdash t : \Pi x. A.B \quad \Gamma \vdash u : A \quad \Gamma \vdash tu : B[x/u]$</td>
</tr>
<tr>
<td><strong>(Conv)</strong></td>
<td>$\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \equiv_{\beta} B \quad \Gamma \vdash t : B$</td>
</tr>
<tr>
<td><strong>(Abs)</strong></td>
<td>$\Gamma \vdash A : \text{Type} \quad \Gamma(x : A) \vdash t : B \quad B \not\in \text{Kind} \quad \Gamma \vdash \lambda x^A.t : \Pi x.A.B$</td>
</tr>
<tr>
<td><strong>(Prod)</strong></td>
<td>$\Gamma \vdash A : \text{Type} \quad \Gamma(x : A) \vdash B : s \quad \Gamma \vdash \Pi x^A.B : s$</td>
</tr>
</tbody>
</table>
The $\lambda \Pi$-calculus modulo
Typing the Rules
Subject Reduction

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

Well-typed rule (strong version)

(Rewrite) \[
\frac{\Gamma \text{ wf} \quad \Gamma\Delta \vdash l : T \quad \Gamma\Delta \vdash r : T}{\Gamma([\Delta]/l \Leftarrow r) \text{ wf}}
\]

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 \([\Delta]l \rightarrow r\) is **left-linear** if the variables in \(\Delta\) 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 \(\beta\)-reduction are more **difficult** to prove.
Example: Well-typedness vs Left-linearity

Nat: Type.
Z: Nat.
S: Nat → Nat.
Plus: Nat → Nat → Nat.
\[ \text{Plus Z n } \rightarrow n \]
\[ \text{Plus (S n) m } \rightarrow S (\text{Plus n m}) \].

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).
\[ \text{Append Z Nil m l } \rightarrow l \]
\[ \text{Append n l1 m l2} \rightarrow \text{Cons (S (Plus n m)) a (Append n l1 m l2)} \].

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 \( \sigma(\text{Append n2 (Cons n a l1) m l2}) \) is well-typed then \( \sigma n2 \equiv_{\beta \Gamma} S \sigma n \)
**Past Solution: Dot Patterns**

**Dot Patterns**

Append \( \{ S \; n \} \; (\text{Cons} \; n \; a \; l1) \; m \; l2 \mapsto \ldots \)

The term between \( \{ \; \} \) is used for *typing only*.

**Problem:**

Subject Reduction is *not preserved* anymore.

**Example**

\[
\begin{align*}
T & : \text{Nat} \to \text{Type}. \\
a & : \text{Nat}. \\
b & : T \; a. \\
F & : x : \text{Nat} \to T \; x. \\
[ ] & F \{ a \} \longrightarrow b.
\end{align*}
\]

Then \( F \; (S \; a) \mapsto b \) but \( F \; (S \; a) \) has type \( T \; (S \; a) \neq (T \; a) \).
Weakening the well-typedness property

Well-typed rule (strong version)

(Strong Rewrite) \[
\begin{array}{c}
\Gamma \text{ wf} \\
\Gamma \Delta \vdash l : T \\
\Gamma \Delta \vdash r : T \\
\end{array}
\Rightarrow 
\\Gamma([\Delta]l \xrightarrow{} r) \text{ wf}
\]

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

Well-typed rule (weak version)

(Weak Rewrite) \[
\begin{array}{c}
\Gamma \text{ wf} \\
\forall \sigma \in S(\Delta), (\Gamma \vdash \sigma l : T \Rightarrow \Gamma \vdash \sigma r : T) \\
\end{array}
\Rightarrow 
\\Gamma([\Delta]l \xrightarrow{} r) \text{ wf}
\]

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

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)

$$\Gamma \text{ wf} \quad S(\Delta, E(l)) \subset S(\Delta, E(r) \cup \{T(l) \equiv_{\beta \Gamma} T(r)\})$$

$$\Gamma([\Delta]/l \leftrightarrow r) \text{ wf}$$

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

Example

\[ E(\ Append\ n2\ (Cons\ n\ a\ l1)\ m\ l2)\ ) = \{\ n2 \equiv_\Gamma\ S\ n\ \} \]
\[ E(\ Cons\ (S\ n)\ a\ (Append\ n\ l1\ m\ l2))\ ) = \{\} \]

- let us write \( @ \) for \( Append \), \( C \) for \( Cons \), \( \mathbb{N} \) for \( Nat \), \( L \) for \( Listn \) and \( + \) (infix) for \( Plus \)

- and let the local context \( \Delta \) be:
  \[
  \begin{align*}
  n2 & : \mathbb{N} & n & : \mathbb{N} & l1 & : L\ n \\
  m & : \mathbb{N} & l2 & : L\ m
  \end{align*}
  \]

Keeping contexts implicit, we have the following typing tree:

\[
\vdash @\ n2 : L(n2) \rightarrow (m:\mathbb{N}) \rightarrow L(m) \rightarrow L(n2+m) \\
\vdash C\ n\ a\ l1 : L(S\ n) \quad n2 \equiv_\Gamma\ S\ n \\
\vdash @\ n2\ (C\ n\ a\ l1) : (m:\mathbb{N}) \rightarrow L(m) \rightarrow L(n2+m) \\
\vdash @\ n2\ (C\ n\ a\ l1)\ m\ l2 : L(n2+m)
\]
A simple implementation

1. Find an approximation $\sigma$ of a most general unifier for $E(l)$. (ie $\sigma$ must be a prefix of any solution of $E(l)$).
2. Check that $\sigma$ 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 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
**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

\[ \Gamma \vdash t = t \quad \text{Refl } t \]
\[ \Gamma \vdash f = g \quad \Gamma \vdash t = u \quad \frac{\Gamma \cup \Delta \vdash f \ t = g \ u}{\text{AppThm}} \]
\[ \frac{\{ \phi \} \vdash \phi}{\text{Assume}} \]
\[ \Gamma \vdash \phi \quad \Delta \vdash \psi \quad \frac{(\Gamma - \{ \psi \}) \cup (\Delta - \{ \phi \}) \vdash \phi = \psi}{\text{DeductAntiSym}} \]
\[ \frac{\Gamma \vdash t = u}{\text{AbsThm } x} \quad \frac{\Gamma \vdash \lambda x^A.t = \lambda x^A.u}{\text{AbsThm } x} \]
\[ \frac{\Gamma \vdash t = u}{\text{AppThm}} \quad \frac{(\lambda x^A.t) \ x = t}{\text{Beta } x \ t} \]
\[ \frac{\Gamma \vdash \phi = \psi \quad \Delta \vdash \phi}{\text{EqMp}} \]
\[ \frac{\Gamma \vdash \phi \quad \Delta \vdash \phi}{\text{DeductAntiSym}} \]
\[ \frac{\Gamma \vdash \phi}{\text{Subst } \sigma} \]
\[ [\sigma] \Gamma \vdash [\sigma] \phi \]
Encoding HOL in the $\Lambda\Pi$-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 \(\lambda\Pi\)-calculus modulo

```plaintext
#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).

[...]
```
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