Dependency Pairs Termination in Dependent Type Theory Modulo Rewriting

Frédéric Blanqui

(joint work with Guillaume Genestier and Olivier Hermant)

Deduc⊢eam

Inria

école normale supérieure paris–saclay

lsv

Outline

Introduction to dependent type theory modulo rewriting

Our contribution

Simply-typed λ-calculus ($\lambda \to$)

introduced by Church in 1940 to give a new formulation of the "simple theory of types" based on λ-calculus

now basis of functional programming and program typing

types $A, B =$

P	(type constant)
$A \to B$	(type of functions from A to B)

terms $t, u =$

x	(variable)
tu	(application of function t to argument u)
$\lambda x : A, t$	(function mapping x of type A to t)

Typing rule of $\lambda \to$

Γ: sequence of type declarations for free variables

$$\text{(var)}\ \frac{}{\Gamma, x : A, \Gamma' \vdash x : A}$$

$$\text{(lam)}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A, t : A \to B}$$

$$\text{(app)}\ \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

Curry-Howard correspondence

later it was remarked that λ-terms/types can be used to represent proofs/formulas:

$\lambda \rightarrow$	logic
type constant	proposition
arrow type	implication
variable	assumption
application	$\Rightarrow$-elim
abstraction	$\Rightarrow$-intro

example: $\lambda x : A, x$ is a proof of $A \Rightarrow A$

simple types correspond to propositional logic

which kind of types correspond to first-order logic ?

Dependent types ($\lambda\Pi$-calculus)

dependent types have been introduced by De Bruijn in the 60s

terms and types are mutually defined:

types $A, B =$

$\mid Pt_1 \dots t_n$	(type constant application)
$\mid \forall x : A, B(x)$	(type of functions from $x : A$ to $B(x)$)

terms $t, u =$

$\mid x$	(variable)
$\mid tu$	(application of function t to argument u)
$\mid \lambda x : A, t$	(function mapping x of type A to t)

examples: $3 \leq 4$, $Vec\,2$, etc.

Typing rule of $\lambda\Pi$

$$\text{(var)}\ \frac{}{\Gamma, x : A, \Gamma' \vdash x : A}$$

$$\text{(lam)}\ \frac{\Gamma + [x : A] \vdash t : B(x)}{\Gamma \vdash \lambda x : A, t : \forall x : A, B(x)}$$

$$\text{(app)}\ \frac{\Gamma \vdash t : \forall x : A, B(x) \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B(u)}$$

problem: $\Gamma \vdash tu : B(u) \wedge u \simeq_\beta u' \not\Rightarrow \Gamma \vdash tu : B(u')$

Definitional equality in dependent type theory

$$\text{(conv)} \quad \frac{\Gamma \vdash t : A}{\Gamma \vdash t : B} \quad \text{if } A \simeq B$$

Usually $\simeq$ is:

- $\leftrightarrow^*_\beta$ (Pure Type Systems (PTS))
- $\leftrightarrow^*_{\beta\iota}$ where $\rightarrow_\iota$ are the rules for induction (Gödel system T, Martin-Löf type theory, Calculus of Inductive Constructions)

Inconvenience: when $+$ defined by induction on 1st argument

$$P(0 + x) \leftrightarrow^*_{\beta\iota} P(x) \not\leftrightarrow^*_{\beta\iota} P(x + 0)$$

$\Rightarrow$ dependent types difficult to use (cast/transport, coherence laws)

Allow user-defined rewriting rules ?

$$x + 0 \to x \ ?$$

$$(x + y) + z \to x + (y + z) \ ?$$

Type-level rewriting allows to encode any functional PTS in $\lambda\Pi$!

- correctness (Cousineau & Dowek, 2007)
- completeness (Assaf, 2015)

$\Rightarrow \lambda\Pi/\mathcal{R}$ can be used as a logical framework/translation hub

$\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)

$$\simeq = \leftrightarrow^*_{\beta\mathcal{R}}$$

Implementation in Dedukti:

- version 1.0 (Boespflug, 2011)
- version 2.0 (Saillard, 2015)
- version 3.0 (Lepigre & B., 2018)

`https://github.com/Deducteam/lambdapi`

Currently in Dedukti

- rules can be both at the object level and at the type level
- LHS can overlap: $x + 0 \to x$, $0 + x \to x$
- LHS can be non-linear: $x - x \to 0$
- LHS can contain defined symbols: $(x + y) + z \to x + (y + z)$

  no notion of constructor, just symbol declarations and rules

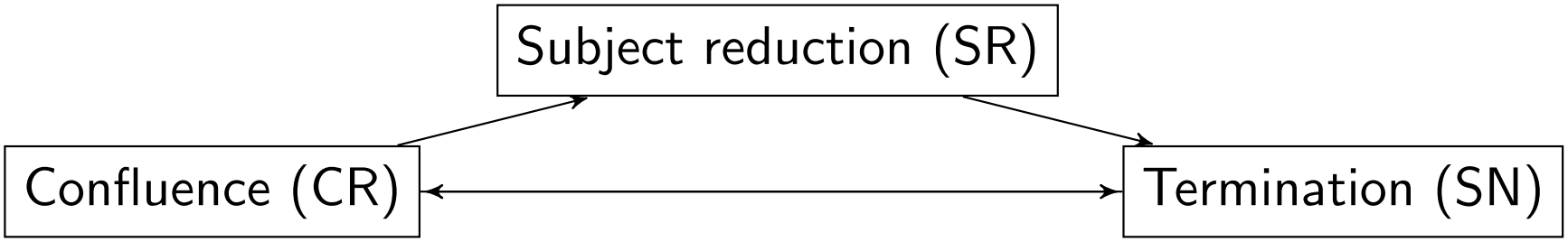
  $\Rightarrow$ you can have computable quotient types ($s(px) \to x$), inductive-recursive types, inductive-inductive types, ...
- LHS can contain abstractions: $lam\,(\lambda x, app\,F\,x) \to F$
- matching is modulo β_0 and associativity-commutativity (AC)

Example

```
symbol Set: TYPE
symbol arrow: Set → Set → Set

symbol El: Set → TYPE
rule El (arrow a b) ↪ El a → El b

symbol app: Πa p, V a p → Πq, V a q → V a (p+q)
rule app a _ (nil _) q m ↪ m
with app a _ (cons _ x p l) q m
    ↪ cons a x (p+q) (app a p l q m)

symbol filter: Πa f p l, V a (len_filter a f p l)
rule filter a f _ (nil _) ↪ nil a
with filter a f _ (cons _ x p l)
    ↪ filter_aux (f x) a f x p l
with filter a f _ (app _ p l q m)
    ↪ app a _ (filter a f p l) _ (filter a f q m)
```

Problem: decidability of $\leftrightarrow^*_{\beta\mathcal{R}}$?

NB. the set of well-typed terms grows with $\mathcal{R}$



checking tools currently used in Dedukti:

- SR: internal (a new algorithm using KB completion is under dev)
- CR: external (CSI^ho, see Confluence Competition CoCo)
- SN: external (SizeChangeTool, G. Genestier, see TermComp)

$\Rightarrow$ SOL soon ?

Rewriting in type theory: some previous results

- 1988: CR for $\lambda^{\rightarrow} + FOR_{\star}^{\rightarrow}$ (Breazu)
- 1989: SN for $\lambda^{\rightarrow} + FOR_{\star}^{\rightarrow}$ (Breazu & Gallier, Okada)
- 1991: SN for $\lambda^{\rightarrow} + HOR_{\star}^{\rightarrow}$ (Jouannaud & Okada)

- 1995: SN for $PTS + HOR_{\star}^{\rightarrow}$ (Barthe)
- 1997: SR+CR+SN for $CC + HOR_{\star}^{\rightarrow}$
  (Barbanera, Fernández & Geuvers)
- 2000: SN for $CC + HOR_{\star}$ (Walukiewicz)
- 2001: SR+SN for $CC + HOR_{\star} + HOR_{\Box}$ (B.)
  $\rightsquigarrow$ prototype of Coq v7 modulo FOR
- 2007: SN for $MLTT + HOR_{\star} + HOR_{\Box}$ (Wahlstedt)
- 2015: SR for $\lambda\Pi + HOR_{\star} + HOR_{\Box}$ (Saillard)

F=First, H=Higher, O=Order, $\star$=object-level, $\Box$=type-level

Outline

Introduction to dependent type theory modulo rewriting

Our contribution

The notion of dependency pair

introduced by Arts & Giesl in 1996
generalizes the notion of call graph to rewriting

Example: for

$$\begin{array}{rcl} 0 + y & \to & y \\ (s\ x) + y & \to & s\ (x + y) \\ 0 \times y & \to & 0 \\ (s\ x) \times y & \to & y + x \times y \end{array}$$

the dependency pairs are:

$$\begin{array}{rcl} (s\ x) + y & > & x + y \\ (s\ x) \times y & > & y + x \times y \\ (s\ x) \times y & > & x \times y \end{array}$$

Arts & Giesl theorem (1996)

Theorem

Given a set $\mathcal{R}$ of rewriting rules, with dependency pairs $>$, $\rightarrow_{\mathcal{R}}$ is SN on FOR terms if

1. *the call relation $\tilde{>} := \rightarrow^*_{arg} \circ >_s$ is SN*

where
$>_s$ *:= closure by substitution of* $>$
$\rightarrow_{arg}$ *:= reduction in arguments*

advantage: no need for a strictly decreasing arg. in every call

Our contribution: extends Arts & Giesl to $\lambda\Pi/\mathcal{R}$

Theorem

Given a set $\mathcal{R}$ of rewriting rules, with dependency pairs $>$, $\to_\beta \cup \to_\mathcal{R}$ is SN on terms typable in $\lambda\Pi/\mathcal{R}$ if

1. *the call relation $\tilde{>} := \to^*_{arg} \circ >_s$ is SN*
2. *$\to_\beta \cup \to_\mathcal{R}$ is locally confluent*
3. *LHS variables are accessible (matching preserves computability)*

Proof. By building a model in some reducibility candidates.

This improves previous results by D. Walhstedt on MLTT (2007)

N.B. We assume local confluence only

Model requirements

we are looking for $[\![\ ]\!] : \mathit{Term} \to \wp(\mathit{Term})$ such that:

- $[\![A]\!]$ is a reducibility candidate à la Girard:
  - $[\![A]\!] \subseteq \mathit{SN}$ (termination)
  - $\to([\![A]\!]) \subseteq [\![A]\!]$ (stability by reduction)
  - $t \in [\![A]\!]$ if t is neutral and $\to(t) \subseteq [\![A]\!]$ (neutral terms)
    t is neutral if, for all $\vec{u}$, $\to(t\vec{u}) \subseteq \to(t)\vec{u} \cup t \to(\vec{u})$
- $[\![\forall x : A, B(x)]\!] = \{t \mid \forall a \in [\![A]\!], ta \in [\![B(a)]\!]\}$ (for dealing with β)
- $t \to u \Rightarrow [\![t]\!] = [\![u]\!]$ (for dealing with $\simeq$)

Model construction

least fixpoint of the monotone function F on the directed-complete poset of partial functions from Term to $\wp(\mathit{Term})$ such that:

- $\mathrm{dom}(F(I)) = \{T \in SN \mid$
  $T \to^* \forall x : A, B(x) \Rightarrow A \in \mathrm{dom}(I) \wedge \forall a \in I(A), B(a) \in \mathrm{dom}(I)\}$
- if $T \in \mathrm{dom}(F(I))$ and $nf(T) = \forall x : A, B(x)$ then
  $F(I)(T) = \{t \mid \forall a \in I(A), ta \in I(B(a))\}$
- if $T \in \mathrm{dom}(F(I))$ and $nf(T)$ is not a product then
  $F(I)(T) = SN$

NB. $nf(T)$ exists when $T \in SN$ and $\to$ is locally confluent

How to prove the termination of $\tilde{>}$?

- Size Change Principle (Lee, Jones & Ben Amram, 2001)
  used by Wahlstedt and in SizeChangeTool by Genestier
- MANY techniques developed in FOR and simply-typed HOR
  can probably be extended to $\lambda\Pi/\mathcal{R}$

state-of-the-art FOR termination checkers are based on
dependency pairs analysis and output certificates
(see CeTA proved in and extracted from Isabelle/HOL)

Conclusion

- dependent type theory modulo user-defined rewrite rules $\lambda\Pi/\mathcal{R}$
- SN of $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$ reduces to SN of call relation $\tilde{>} := \rightarrow^*_{arg} \circ >_s$
- termination of $\tilde{>}$ can be proved by techniques from FP or FOR
- implementation in Dedukti/TermComp by Guillaume Genestier