Dependency Pairs Termination in Dependent Type Theory Modulo Rewriting

To appear in the proceedings of FSCD’19

Frédéric Blanqui, Guillaume Genestier, Olivier Hermant

Deducl-eam

INRIA  école normale supérieure paris-saclay  LSV  MINES ParisTech

Oslo, 19 June 2019
Outline

Introduction to dependent type theory modulo rewriting

Our contribution
Definitional equality in dependent type theory

\[
\frac{t : A}{t : B} \quad \text{if } A \simeq B
\]

Usually \(\simeq\) is:

- \(\simeq_\beta\) (STT, PTS)
- \(\simeq_\beta_\iota\) where \(\rightarrow_\iota\) are the rules for induction (T, MLTT, CIC)

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

\[
P(0 + x) \simeq P(x) \not\simeq P(x + 0)
\]

\(\Rightarrow\) dependent types difficult to use (cast/transport, coherence laws)
Allow user-defined rules in $\simeq$?

$x + 0 \rightarrow x$?

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

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

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

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

Implementation in Dedukti:

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

- rules can be both at the object level and at the type level
- LHS can overlap: \( x + 0 \rightarrow x, \ 0 + x \rightarrow x \)
- LHS can be non-linear: \( x - x \rightarrow 0 \)
- LHS can contain defined symbols: \((x + y) + z \rightarrow x + (y + z)\)
  
  no notion of constructor, just symbol declarations and rules
  
  \( \Rightarrow \) you can have computable quotient types \((s(px) \rightarrow x)\),
  inductive-recursive types, inductive-inductive types, \ldots

- LHS can contain abstractions: \( \text{lam} (\lambda x, \text{app} F x) \rightarrow F \)

- matching is modulo \( \beta_0 \) and associativity-commutativity (AC)
symbol Set: TYPE
symbol arrow: Set ⇒ Set ⇒ Set

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

symbol app: ∀a p, ∀ a p ⇒ ∀q, ∀ a q ⇒ ∀ a (p+q)
  rule app a _ (nil _) q m −→ m
  rule app a _ (cons _ x p l) q m
    −→ cons a x (p+q) (app a p l q m)

symbol filter: ∀a f p l, ∀ a (len_filter a f p l)
  rule filter a f _ (nil _) −→ nil a
  rule filter a f _ (cons _ x p l)
    −→ filter_aux (f x) a f x p l
  rule filter a f _ (app _ p l q m)
    −→ app a _ (filter a f p l) _ (filter a f q m)
Problem: decidability of $\sim$?

Subject reduction (SR)

Confluence (CR) [←] Termination (SN)

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)
**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 $CC + HOR_\star\rightarrow$ (Walukiewicz)
- 2001: SR+SN for $CC + HOR_\star + HOR_\square$ (B.)
  - $\leadsto$ prototype of Coq v7 modulo FOR
- 2007: SN for $MLTT + HOR_\star + HOR_\square$ (Wahlstedt)
- 2015: SR for $\lambda\Pi + HOR_\star + HOR_\square$ (Saillard)

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

\[
\begin{align*}
0 + y & \rightarrow y \\
(s \times x) + y & \rightarrow s (x + y) \\
0 \times y & \rightarrow 0 \\
(s \times x) \times y & \rightarrow y + x \times y
\end{align*}
\]

the dependency pairs are:

\[
\begin{align*}
(s \times x) + y & > x + y \\
(s \times x) \times y & > y + x \times y \\
(s \times x) \times y & > x \times y
\end{align*}
\]
Theorem

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

1. the call relation $\tilde{\succ} := \rightarrow_{\text{arg}}^* \circ \succ_s$ is SN

where

$\succ_s := \text{closure by substitution of } \succ$

$\rightarrow_{\text{arg}} := \text{reduction in arguments}$

advantage: no need for a strictly decreasing arg. in every call
Our contribution: extends Arts & Giesl to $\lambda\Pi/R$

Theorem

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

1. the call relation $\tilde{\succ} := \rightarrow^{*}_{\text{arg}} \circ \succ_{s}$ is SN
2. $\rightarrow_{\beta} \cup \rightarrow_{\mathcal{R}}$ 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
How to prove the termination of $\succeq$?

- 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 that 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 $\lambda \Pi / R$
- SN of $\rightarrow^\beta \cup \rightarrow^R$ reduces to SN of call relation $\tilde{>} := \rightarrow^*_{\text{arg}} \circ >_s$
- termination of $\tilde{>}$ can be proved by techniques from FP or FOR
- implementation in Dedukti/TermComp by Guillaume Genestier
- extension to Agda this summer?
Want to know more about rewriting?

Come to the International School on Rewriting (ISR)!

Paris, 1-6 July

https://isr2019.mines-paristech.fr/