# Dependency Pairs Termination in Dependent Type Theory Modulo Rewriting

To appear in the proceedings of FSCD'19

Frédéric Blanqui, Guillaume Genestier, Olivier Hermant

Deduc⊢eam




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 $\to_\iota$ 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 $\simeq$ ?

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

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

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

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: $x + 0 \to x$, $0 + x \to x$
- 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 $\beta_0$ and associativity-commutativity (AC)

# Example

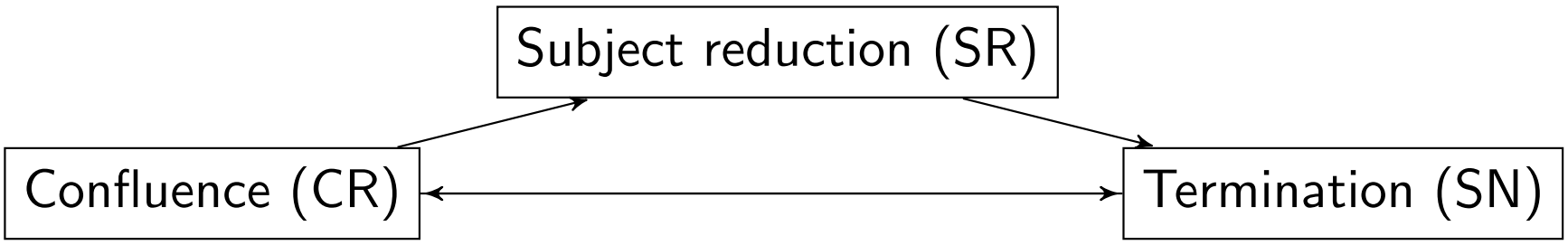
```
symbol Set: TYPE
symbol arrow: Set ⇒ Set ⇒ Set

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

symbol app: ∀a p, 𝕍 a p ⇒ ∀q, 𝕍 a q ⇒ 𝕍 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, 𝕍 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 $\simeq$ ?




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 $CC + HOR_{\star}$ (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

# 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
  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 $\lambda\Pi/\mathcal{R}$
- 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
- 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/`