Dependency Pairs in Dependent Type Theory with Rewriting

Guillaume Genestier
Joint work with Frédéric Blanqui and Olivier Hermant

Thursday, June 27, 2019

Contents

Dedukti is a type-checker for the $\lambda\Pi$-calculus modulo rewriting.

Example of dependent type

```
def F : Nat -> TYPE
[]  F 0     --> Nat
[n] F (s n) --> Nat -> F n
```

F n = Nat $\to$ Nat $\to \cdots \to$ Nat with n arrows.

Example of rewriting rules

```
def sum : (n: Nat) -> F n
[]  sum 0         --> 0
[]  sum (s 0)     --> λx, x
[n] sum (s (s n)) --> λx y, sum (s n) (plus x y)
```

Example : sum 5 1 2 3 4 5 $\longrightarrow^*$ 1+2+3+4+5 $\longrightarrow^*$ 15

Typing Rules

Abstraction:

$$\frac{\Gamma \vdash A : \mathrm{Type} \qquad \Gamma, x : A \vdash B : s \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : A).t : \Pi(x : A).B}$$

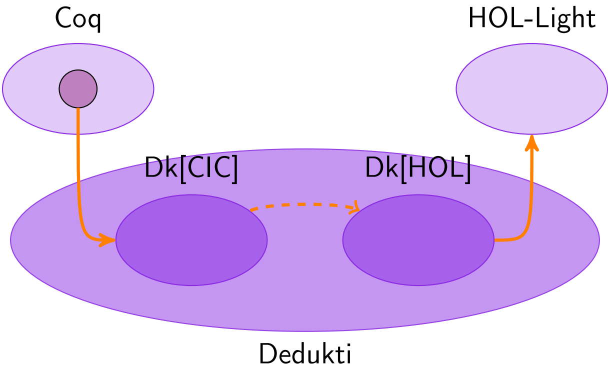
Application:

$$\frac{\Gamma \vdash t : \Pi(x : A).B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B\,[x \mapsto u]}$$

Conversion:

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s \qquad A \leftrightarrow_{\beta\mathcal{R}} B}{\Gamma \vdash t : B}$$

Dedukti is well-suited for interoperability [Thiré18]

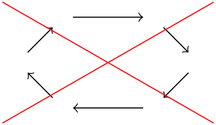


Non-restrictive Rewriting

- overlapping: $x + 0 \to x, \; 0 + x \to x$
- non-linearity: $x - x \to 0$
- defined symbols: $(x + y) + z \to x + (y + z)$
- higher-order: $lam\,(\lambda x.app\,F\,x) \to F$
- there can be rules both at the object and type levels

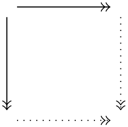
Expected Properties of Rewriting

- Termination: There is no infinite sequence of reduction starting from a well-typed term;



- Typing preservation (*Subject reduction*): If a term is well-typed, its reducts have the same type;

- Confluence: Two reducts of a term have a common reduct.



Termination: Difficulties

- The set of terms $\lambda\Pi/\mathcal{R}$ depends on rewriting rules $\mathcal{R}$;
- Higher-order rules cannot be dealt with independently of β-reduction;
- Type-level rewriting forbids the use of erasing tricks reducing termination to simply-typed λ-calculus;
- Type-level rewriting allows to encode any functional Pure Type System (e.g. System F or the Calculus of Constructions).

Contents

Main Goal

If $\Gamma \vdash t : T$, then $t \in \mathrm{SN}(\rightarrow_{\beta\mathcal{R}})$.

Main Goal

Find a criterion such that:

If $\Gamma \vdash t : T$, then $t \in \mathrm{SN}(\rightarrow_{\beta\mathcal{R}})$.

Goal

Define $\llbracket T \rrbracket$ such :

- $\Gamma \vdash t : T$ implies $t \in \llbracket T \rrbracket$,
- $t \in \llbracket T \rrbracket$ implies $t \in \mathrm{SN}(\rightarrow_{\beta\mathcal{R}})$.

Reducibility Conditions

- $\llbracket T \rrbracket \subseteq \mathrm{SN}$,
- If $t \in \llbracket T \rrbracket$ and $t \rightarrow_{\beta\mathcal{R}} u$, then $u \in \llbracket T \rrbracket$,
- If t is neutral and $\{ u | \, t \rightarrow_{\beta\mathcal{R}} u \} \subseteq \llbracket T \rrbracket$, then $t \in \llbracket T \rrbracket$.

- For β-reduction, we set $\llbracket \Pi(x : A).B \rrbracket = \{ t | \, \forall a \in \llbracket A \rrbracket , t\, a \in \llbracket B\,[x \mapsto a] \rrbracket \}$
- For conversion rule, if $T \leftrightarrow_{\beta\mathcal{R}} U$, then $\llbracket T \rrbracket = \llbracket U \rrbracket$.

Our Interpretation

We define $[\![.]\!]$ as the fixpoint of a monotonic function.

Lemma (Adequacy)

If for all f, $f \in [\![\Theta_f]\!]$ and $\Gamma \vdash t : T$, then $t \in [\![T]\!]$.

Goal

Define $[\![T]\!]$ such that:

- $\Gamma \vdash t : T$ implies $t \in [\![T]\!]$, $\sim$
- $t \in [\![T]\!]$ implies $t \in \mathrm{SN}(\to_{\beta\mathcal{R}})$. ✓

Dependency Pairs

Definition (Dependency Pairs)

A rule $f\,\bar{l} \rightarrow r$ gives rise to the *dependency pairs* $f\,\bar{l} > g\,\bar{m}$ where:

- g is (partially) defined by rewriting,
- $g\,\bar{m}$ is a maximally applied subterm of r.

Theorem (Arts and Giesl, 2000)

First order:
$\rightarrow_{\mathcal{R}}$ *terminates iff there is no* $f\,\bar{t} > g\,\bar{u} \rightarrow^*_{arg} g\,\bar{u}' > \ldots$

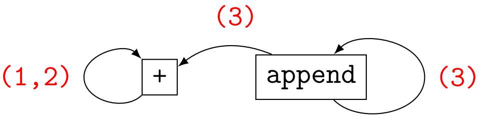
Higher-Order

Static and dynamic definition: [Blanqui06][Kusakari, Sakai 07][Kop, van Raamsdonk 12][Kop, Fuhs 19]

```
def plus : Nat -> Nat -> Nat.
set infix "+" := plus.
[q]   0     + q      --> q.
[p,q] (S p) + q      --> S (p + q). (1)
[p,q] p     + (S q)  --> S (p + q). (2)

def append: (p: Nat) -> List p ->
            (q: Nat) -> List q -> List (p + q).
[q,m]       append _ nil          q m --> m.
[x,p,l,q,m] append _ (cons x p l) q m -->
             cons x (p + q) (append p l q m). (3)
```

```
(1)          (S p) + q     > p + q
(2)          p     + (S q) > p + q
(3) append _ (cons x p l) q m > append p l q m
(3) append _ (cons x p l) q m > p + q
```

```
def plus : Nat -> Nat -> Nat.
set infix "+" := plus.
[q]    0     + q      --> q.
[p,q] (S p) + q      --> S (p + q). (1)
[p,q] p     + (S q) --> S (p + q). (2)

def append: (p: Nat) -> List p ->
            (q: Nat) -> List q -> List (p + q).
[q,m]       append _ nil            q m --> m.
[x,p,l,q,m] append _ (cons x p l) q m -->
             cons x (p + q) (append p l q m). (3)
```



Well-Structuring

$\succeq$ quasi-order on the signature compatible with rewriting and typing.

Definition (Well-Structured System)

$\mathcal{R}$ is *well-structured* if for all rule $(\Delta, f\,\bar{l} \to r)$, with $\Theta_f = \Pi(\bar{x} : \bar{T}).U$, we have $\Delta \vdash_{\preceq f} r : U[\bar{x} \mapsto \bar{l}]$.

Definition (Plain Function Passing)

$f\,\bar{l} \to r$ is *PFP* if every functional type variable occurring in r is equal to one of the l_i.

Main Result

Reminder

If for all f, $f \in [\![\Theta_f]\!]$ and $\Gamma \vdash t : T$, then $t \in [\![T]\!]$.

Lemma

Every $f \in [\![\Theta_f]\!]$, if:

- *$\mathcal{R}$ is well-structured,*
- *$\mathcal{R}$ is PFP,*
- *$(>\rightarrow^*_{arg})$ is well-founded.*

Main Result

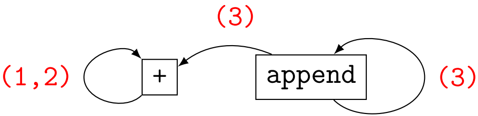
Theorem

$\rightarrow_{\beta\mathcal{R}}$ *terminates on every typable term in* $\lambda\Pi/\mathcal{R}$ *if:*

- $\rightarrow_{\beta\mathcal{R}}$ *is locally confluent and type preserving,*
- $\mathcal{R}$ *is well-structured and Plain Function Passing,*
- $(> \rightarrow^{*}_{arg})$ *is well-founded.*

Contents

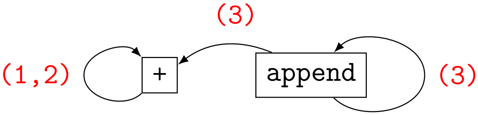
```
def plus : Nat -> Nat -> Nat.
set infix "+" := plus.
[q]   0      + q      --> q.
[p,q] (S p) + q      --> S (p + q). (1)
[p,q] p      + (S q) --> S (p + q). (2)

def append: (p: Nat) -> List p ->
            (q: Nat) -> List q -> List (p + q).
[q,m]         append _ nil           q m --> m.
[x,p,l,q,m] append _ (cons x p l) q m -->
               cons x (p + q) (append p l q m). (3)
```



Size-Change Termination : Example

Introduced in [Lee, Jones, Ben Amram, 02] and used for MLTT in [Wahlstedt07].
Keeping track of the evolution of the sizes of the arguments:

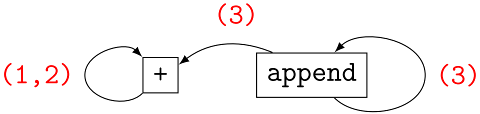
(1) `plus (S p) q` $>$ `plus p q`	$\begin{array}{c c} & \begin{array}{cc} p & q \end{array} \\ \begin{array}{r} S\ p \\ q \end{array} & \begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix} \end{array}$
(3) `append _ (cons x p l) q m` $>$ `plus p q`	$\begin{array}{c c} & \begin{array}{cc} p & q \end{array} \\ \begin{array}{r} \text{cons x p } \overline{\text{l}} \\ \text{cons x p l} \\ q \\ m \end{array} & \begin{pmatrix} \infty & \infty \\ < & \infty \\ \infty & = \\ \infty & \infty \end{pmatrix} \end{array}$



Size-Change Termination : Example

Introduced in [Lee, Jones, Ben Amram, 02] and used for MLTT in [Wahlstedt07].
Keeping track of the evolution of the sizes of the arguments:

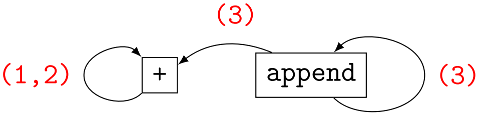
(1) `plus (S p) q > plus p q`	$\begin{array}{c} \\ \mathsf{S\ p} \\ \mathsf{q} \end{array} \begin{array}{c} \mathsf{p}\quad\ \mathsf{q} \\ \begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix} \end{array}$
(3) `append _ (cons x p l) q m > plus p q`	$\begin{array}{r} \\ \mathsf{cons\ x\ p}\ \overline{\mathsf{l}} \\ \mathsf{q} \\ \mathsf{m} \end{array} \begin{array}{c} \mathsf{p}\quad\ \mathsf{q} \\ \begin{pmatrix} \infty & \infty \\ < & \infty \\ \infty & = \\ \infty & \infty \end{pmatrix} \end{array}$



Introduced in [Lee, Jones, Ben Amram, 02] and used for MLTT in [Wahlstedt07].

Keeping track of the evolution of the sizes of the arguments:

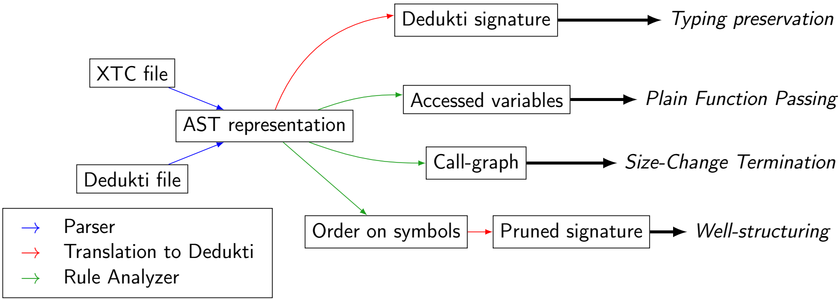
(1) `plus (S p) q > plus p q`	$\begin{array}{r} \\ \mathsf{S\ p} \\ \mathsf{q} \end{array} \begin{array}{c} \mathsf{p} \quad \mathsf{q} \\ \begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix} \end{array}$
(3) `append _ (cons x p l) q m > plus p q`	$\begin{array}{r} \\ \mathsf{cons\ x\ p\ \bar{l}} \\ \mathsf{q} \\ \mathsf{m} \end{array} \begin{array}{c} \mathsf{p} \quad \mathsf{q} \\ \begin{pmatrix} \infty & \infty \\ < & \infty \\ \infty & = \\ \infty & \infty \end{pmatrix} \end{array}$



Theorem

$\to_{\beta\mathcal{R}}$ terminates on every typable term in $\lambda\Pi/\mathcal{R}$ if:

- *$\to_{\beta\mathcal{R}}$ is locally confluent and type preserving,*
- *$\mathcal{R}$ is well-structured and Plain Function Passing,*
- *$\mathcal{R}$ is Size-Change Terminating.*



Comparisons with Other Tools

Simply-typed

- Annual competition, few participants (Wanda, SOL?),
- Prove less examples, much faster.

Orthogonal Rules

- Integrated in proof assistants,
- Very similar to Agda's,
- Easily deals with argument permutation, unlike Coq's.

Future Work

Plain Function Passingness

Weaken this hypothesis to "positivity", requires to use structural ordering rather than subterm.
Example : Brouwer ordinals (`lim : (Nat -> Ord) -> Ord.`)

Dependency Pairs

- Adapt more "processors",
- Recover completeness.

Tool improvement

- Modularity results:
  - with simple types (like [Harper, Honsell, Plotkin 93]),
  - with first-order (like [Jouannad, Okada97] and [Fuhs, Kop11]),
- Non-termination,
- Other input format (Agda).

Dependency Pairs in Dependent Type Theory with Rewriting

Guillaume Genestier
Joint work with Frédéric Blanqui and Olivier Hermant

Thursday, June 27, 2019