Dependency Pairs in Dependent Type Theory with Rewriting

Guillaume Genestier
Joint work with Frédéric Blanqui and Olivier Hermant

Thursday, June 27, 2019
1 Context: Dedukti
2 Termination Criterion
3 SizeChangeTool
Dedukti is a type-checker for the $\lambda\Pi$-calculus modulo rewriting.

Example of dependent type

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

$F_n = \text{Nat} \rightarrow \text{Nat} \rightarrow \cdots \rightarrow \text{Nat}$ with $n$ arrows.

Example of rewriting rules

```plaintext
def sum : (n: Nat) -> F n
[] sum 0 --> 0
[n] sum (s n) --> \lambda x, x
```

Example: $\text{sum } 5 1 2 3 4 5 \rightarrow^* 1+2+3+4+5 \rightarrow^* 15$
Typing Rules

Abstraction:

\[ \Gamma \vdash A : \text{Type} \quad \Gamma, x : A \vdash B : s \quad \Gamma, x : A \vdash t : B \]

\[ \Gamma \vdash \lambda(x : A). t : \Pi(x : A). B \]

Application:

\[ \Gamma \vdash t : \Pi(x : A). B \quad \Gamma \vdash u : A \]

\[ \Gamma \vdash t u : B[x \mapsto u] \]

Conversion:

\[ \Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \leftrightarrow_{\beta \eta} B \]

\[ \Gamma \vdash t : B \]
Dedukti is well-suited for interoperability

[Thiré18]

\[\text{Dedukti} \quad \text{Coq} \quad \text{HOL-Light} \]

\[\text{Dk[CIC]} \quad \text{Dk[HOL]} \]

Dependency Pairs in Dependent Type Theory
Guillaume Genestier
Non-restrictive Rewriting

- **overlapping:**  \( x + 0 \rightarrow x, \ 0 + x \rightarrow x \)

- **non-linearity:**  \( x - x \rightarrow 0 \)

- **defined symbols:**  \( (x + y) + z \rightarrow x + (y + z) \)

- **higher-order:**  \( lam(\lambda x. app F x) \rightarrow F \)

- 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 $\beta$-reduction;

- Type-level rewriting forbids the use of erasing tricks reducing termination to simply-typed $\lambda$-calculus;

- Type-level rewriting allows to encode any functional Pure Type System (e.g. System F or the Calculus of Constructions).
Contents

1 Context: Dedukti

2 Termination Criterion
   • Logical Relations
   • Dependency Pairs
   • Well-Structuring
   • Main Theorem

3 SizeChangeTool
If $\Gamma \vdash t : T$, then $t \in SN(\rightarrow_\beta R)$. 
Find a criterion such that:
If $\Gamma \vdash t : T$, then $t \in SN(\rightarrow_{\beta R})$. 
Logical Relations

Goal

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

Reducibility Conditions

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

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

[Tait67][Girard88]
We define \([\cdot]\) as the fixpoint of a monotonic function.

**Lemma (Adequacy)**

*If for all* \(f \in [\Theta_f]\) *and* \(\Gamma \vdash t : T\), *then* \(t \in [T]\).*

**Goal**

*Define* \([T]\) *such that:*

- \(\Gamma \vdash t : T\) implies \(t \in [T]\),
- \(t \in [T]\) implies \(t \in SN(\rightarrow_{\beta R})\).
**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 \to Nat \to Nat.

set infix "+" := plus.

[q] 0 + q \rightarrow q.

[p,q] (S p) + q \rightarrow S (p + q). (1)

[p,q] p + (S q) \rightarrow S (p + q). (2)

def append: (p: Nat) \rightarrow List p \rightarrow (q: Nat) \rightarrow List q \rightarrow List (p + q).

[q,m] append _ nil q m \rightarrow m.

[x,p,l,q,m] append _ (cons x p l) q m \rightarrow
cons x (p + q) (append p l q m). (3)

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

(1,2) + append (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 \((\Delta, f \bar{I} \rightarrow r)\), with \( \Theta_f = \Pi(x : T).U \), we have \( \Delta \vdash_{\leq_f} r : U[x \mapsto \bar{I}] \).

**Definition (Plain Function Passing)**

\( f \bar{I} \rightarrow r \) 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 \( t \in \Theta T \).

Lemma
Every \( f \in \Theta_f \), if:
- \( \mathcal{R} \) is well-structured,
- \( \mathcal{R} \) is PFP,
- \( (> \rightarrow^*_{\text{arg}}) \) is well-founded.
Main Result

Theorem

\( \rightarrow_{\beta_R} \) terminates on every typable term in \( \lambda \Pi / \mathcal{R} \) if:

- \( \rightarrow_{\beta_R} \) is locally confluent and type preserving,
- \( \mathcal{R} \) is well-structured and Plain Function Passing,
- \( (>\rightarrow_{\text{arg}}^*) \) is well-founded.
1 Context: Dedukti

2 Termination Criterion

3 SizeChangeTool
   - Size-Change Termination
   - Implementation
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:

\[ \text{(1) plus } (S \ p) \ q > \text{ plus } p \ q \]

\[ \begin{array}{c|cc}
S \ p & p & q \\
\hline
q & < & \infty \\
\infty & = & \infty \\
\end{array} \]

\[ \text{(3) append } - \ (\text{cons } x \ p \ l) \ q \ m > \text{ plus } p \ q \]

\[ \begin{array}{c|cc}
\text{cons } x \ p \ l & p & q \\
\hline
- & \infty & \infty \\
q & > & \infty \\
m & \infty & = \\
\end{array} \]

\[ (1,2) \quad + \quad \text{append} \quad (3) \]
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:

\[ (1) \text{ plus } (S\ p)\ q > \text{ plus } p\ q \]

\[
\begin{array}{c|c}
\text{plus} & p & q \\
\hline
S\ p & < & \infty \\
q & = & \\
\end{array}
\]

\[ (3) \text{ append } - (\text{cons} \ x\ p\ l)\ q\ m > \text{ plus } p\ q \]

\[
\begin{array}{c|c}
\text{append} & p & q \\
\hline
\text{cons} \ x\ p\ l & < & \infty \\
q & = & \\
m & = & \infty \\
\end{array}
\]

(1, 2) + append (3)
Introduced in [Lee, Jones, Ben Amram, 02] and used for MLTT in [Wahlstedt07].

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

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) $\text{plus} \ (S \ p) \ q &gt; \text{plus} \ p \ q$</td>
<td>$\begin{align*} S \ p \ q \end{align*}$ $(\begin{array}{c} &lt; \ \infty \end{array})$</td>
</tr>
<tr>
<td>(3) $\text{append} \ _ \ (\text{cons} \ x \ p \ l) \ q \ m &gt; \text{plus} \ p \ q$</td>
<td>$\begin{align*} \text{cons} \ x \ p \ l \ q \ m \end{align*}$ $(\begin{array}{c} \infty \ \infty \ \infty \end{array})$</td>
</tr>
</tbody>
</table>

(1,2) $+$ (3) $\text{append}$ (3)
Theorem

\[ \rightarrow_{\beta R} \text{ terminates on every typable term in } \lambda \Pi / R \text{ if:} \]

- \[ \rightarrow_{\beta R} \text{ is locally confluent and type preserving,} \]
- \[ R \text{ is well-structured and Plain Function Passing,} \]
- \[ R \text{ is Size-Change Terminating.} \]
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 ($\text{lim} : (\text{Nat} \to \text{Ord}) \to \text{Ord}$).

Dependency Pairs

- Adapt more “processors”,
- Recover completeness.

Tool improvement

- Modularity results:
  - with simple types (like [Harper, Honsell, Plotkin 93]),
  - with first-order (like [Jouannad, Okada 97] and [Fuhs, Kop 11]),
- 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