Proving termination using the Size-Change Principle

Guillaume Genestier

Olivier Hermant (MINES ParisTech) and Frédéric Blanqui (Inria)

Friday, September 15th 2017
Dedukti is a multipurpose type-checker based on the $\lambda\Pi$-calculus modulo theory.

**Example of rewrite rule**

Nat : Type.
0 : Nat.
S : Nat -> Nat.

def plus : Nat -> Nat -> Nat.
[n] plus 0 n --> n
[m,n] plus (S m) n --> S (plus m n)
[m,n] plus m (S n) --> S (plus m n).

**Example of dependent type**

List : Nat -> Type.
nil : List 0.
Cons : (n:Nat) -> Nat -> List n -> List (S n)
Dedukti is well-suited for interoperability
We consider a set of rewrite rules.

<table>
<thead>
<tr>
<th>Nat : Type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 : Nat.</td>
</tr>
<tr>
<td>S : Nat -&gt; Nat.</td>
</tr>
</tbody>
</table>

**def** plus : Nat -> Nat -> Nat.

[n] plus 0 n --> n

[m,n] plus (S m) n --> S (plus m n).

**def** mult : Nat -> Nat -> Nat.

[] mult 0 _ --> 0

[m,n] mult (S m) n --> plus n (mult m n).
We consider a set of rewrite rules.

<table>
<thead>
<tr>
<th>Nat : Type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 : Nat.</td>
</tr>
<tr>
<td>S : Nat -&gt; Nat.</td>
</tr>
<tr>
<td>def plus : Nat -&gt; Nat -&gt; Nat.</td>
</tr>
<tr>
<td>[n] plus 0 n --&gt; n</td>
</tr>
<tr>
<td>[m,n] plus (S m) n --&gt; S (plus m n).</td>
</tr>
<tr>
<td>def mult : Nat -&gt; Nat -&gt; Nat.</td>
</tr>
<tr>
<td>[] mult 0 _ --&gt; 0</td>
</tr>
<tr>
<td>[m,n] mult (S m) n --&gt; plus n (mult m n).</td>
</tr>
</tbody>
</table>
We consider a set of rewrite rules.

<table>
<thead>
<tr>
<th>Nat : Type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 : Nat.</td>
</tr>
<tr>
<td>S : Nat -&gt; Nat.</td>
</tr>
</tbody>
</table>

\[
\text{def plus : Nat -> Nat -> Nat.} \\
[n] \text{plus 0 n --> n} \\
[m,n] \text{plus (S m) n --> S (plus m n).}
\]

\[
\text{def mult : Nat -> Nat -> Nat.} \\
[] \text{mult 0 _ --> 0} \\
[m,n] \text{mult (S m) n --> plus n (mult m n).}
\]
We consider a set of rewrite rules.

<table>
<thead>
<tr>
<th>Nat : Type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 : Nat.</td>
</tr>
<tr>
<td>S : Nat -&gt; Nat.</td>
</tr>
<tr>
<td>def plus : Nat -&gt; Nat -&gt; Nat.</td>
</tr>
<tr>
<td>[n] plus 0 n --&gt; n</td>
</tr>
<tr>
<td>[m,n] plus (S m) n --&gt; S (plus m n).</td>
</tr>
<tr>
<td>def mult : Nat -&gt; Nat -&gt; Nat.</td>
</tr>
<tr>
<td>[] mult 0 _ --&gt; 0</td>
</tr>
<tr>
<td>[m,n] mult (S m) n --&gt; plus n (mult m n).</td>
</tr>
</tbody>
</table>
Proving termination using the Size-Change Principle

Guillaume Genestier
# Mutually recursive definition

<table>
<thead>
<tr>
<th>def f : Nat -&gt; Nat -&gt; Nat.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>def g : Nat -&gt; Nat -&gt; Nat -&gt; Nat.</td>
<td></td>
</tr>
<tr>
<td>[x] f 0 x --&gt; x</td>
<td></td>
</tr>
</tbody>
</table>
| [i,x] f (S i) x --> g i x (S i)           | (\left( \begin{array}{c} < \ 
|                                            | \infty \ = \ 
|                                            | \infty \ = \ 
|                                            | \infty \ = \ \infty \end{array} \right) |
| [a,b,c] g a b c --> f a (plus b c)        | \left( \begin{array}{c} = \ \infty \ 
|                                            | \infty \ = \ \infty \ 
|                                            | \infty \ = \ \infty \ 
|                                            | \infty \ = \ \infty \end{array} \right) |
Proving termination using the Size-Change Principle

Guillaume Genestier
Transitive closure of the call graph

Proving termination using the Size-Change Principle

Guillaume Genestier
An implementation of this algorithm has been done for Dedukti.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List : Type.</td>
<td></td>
</tr>
<tr>
<td>Nil : List.</td>
<td></td>
</tr>
<tr>
<td>Cons : Nat -&gt; List -&gt; List.</td>
<td></td>
</tr>
<tr>
<td>def map : (Nat -&gt; Nat) -&gt; List -&gt; List.</td>
<td></td>
</tr>
<tr>
<td>[] map _ Nil --&gt; Nil.</td>
<td></td>
</tr>
<tr>
<td>[f, x, l] map f (Cons x l) --&gt; Cons (f x) (map f l).</td>
<td></td>
</tr>
<tr>
<td>Lamt : Type.</td>
<td></td>
</tr>
<tr>
<td>Lam : ( Lamt -&gt; Lamt ) -&gt; Lamt.</td>
<td></td>
</tr>
<tr>
<td>[f,t] App (Lam f) t --&gt; f t.</td>
<td></td>
</tr>
</tbody>
</table>
Define a reducibility predicate for weak normalisation.

Show that if every function in the signature is reducible then every typable term is reducible.

Show that if the call relation is well-founded and every type in the signature is reducible, then every function in the signature is reducible.

The size-change principle is used to show that the call relation is well-founded.

The reducibility of every type occurring in the signature is decidable.
Definition (Terms)

We use:
- \(x, y, z\) to denote variables,
- \(f, g\) to denote defined constants,
- \(c\) to denote element constructors,
- \(d\) to denote set constructors.

\[ t, \tau, u, v, l, r ::= x \mid \lambda(x : u).t \mid tu \mid \Pi(x : t) u \mid \text{Kind} \mid \text{Type} \mid c \mid d \mid f \]

Definition (Contexts)

\[ \Gamma, \Delta ::= [] \mid \Gamma, x : t \]
Normalisation predicates

\[\begin{align*}
\text{NF}(u) & \equiv \neg (\exists v. u \leadsto v) \\
\text{SN}(u) & \equiv \neg (\exists (v_i)_{i \in \mathbb{N}}. v_0 = u \land \forall i. v_i \leadsto v_{i+1}) \\
\text{WN}(u) & \equiv \exists v. u \leadsto^* v \land \text{NF}(v) \\
\downarrow v & \equiv u \leadsto^* v \land \text{NF}(v)
\end{align*}\]
### Sub-categories of terms

#### Definition (β-normal term)

We define this syntactical sub-category of terms as:

\[
s ::= x \; s_1 \ldots s_n \mid h \; s_1 \ldots s_{\text{ar}(h)} \mid \lambda(x : T).s
\]

where \( h \) is one symbol in the signature, set constructor, element constructor or defined function.

#### Definition (Constructor patterns)

\[
p ::= x \mid c \; p_1 \ldots p_n
\]

#### Definition (Strongly neutral terms)

\[
b ::= x \; t_1 \ldots t_n \; \text{where} \; \text{NF}(t_i)
\]

\[
| f \; t_1 \ldots t_n \; \text{where} \; \text{NF}(f \; t_1 \ldots t_n) \; \text{and} \; n \geq \text{ar}(f)
\]
Each *rewrite rule* is of the form $f \ p_1 \ldots \ p_k \to s$ where:

- the $p_i$ are constructor patterns,
- $k \leq \text{ar}(f)$,
- $p_k$ is not a variable,
- $s$ is $\beta$-normal,
- $s$ starts with $(\text{ar}(f) - k)$ $\lambda$-abstractions,
- the rule is *left-linear*, meaning that a free variable cannot appear twice in $f \ p_1 \ldots \ p_k$.

Furthermore, the set of rewrite rules is *non-unifiable*, meaning that for any two rules $l_1 \to r_1$ and $l_2 \to r_2$, there are no substitutions $\sigma_1$ and $\sigma_2$ such that $\sigma_1(l_1) = \sigma_2(l_2)$.

**Property (Confluence)**

*Such a rewrite system is orthogonal, hence it is confluent.*
The reducibility predicate

\[
\text{RED}_{(\text{Kind})}(t) \text{ holds if one of the following conditions occurs:}
\]

- \( t = \text{Type} \) and then \( \text{RED}_{(\text{Type})}(u) \) holds if one of the following conditions occurs:

\[
\exists d, u_1, \ldots, u_m. \begin{cases}
  u \downarrow d u_1 \ldots u_m \\
  \mathcal{D}(d) = \prod(x_1 : T_1) \ldots (x_k : T_k) \text{ Type} \quad \text{and} \\
  \forall i. \text{RED}_{(\text{Type})}(T_i) \land \text{RED}_{(T_i)}(u_i)
\end{cases}
\]

then \( \text{RED}(u)(v) \) holds if one of the following conditions occurs:

\[
\begin{cases}
  v \downarrow c v_1 \ldots v_n \\
  \mathcal{C}(c) = \prod(x_1 : U_1) \ldots (x_m : U_m) (d \tau_1 \ldots \tau_k) \\
  \forall i. \text{RED}_{(\text{Type})}(U_i[v_1/x_1, \ldots, v_i-1/x_{i-1}]) \\
  \forall i. \text{RED}(u_i[v_1/x_1, \ldots, v_i-1/x_{i-1}]) (v_i)
\end{cases}
\]

- \( \exists b. v \downarrow b \)
The reducibility predicate

\[ u \Downarrow \Pi(x : A) B \]

- \( \exists A, B. \begin{cases} \RED_{\text{Type}}(A) \\ \forall a. \RED(A)(a) \Rightarrow \RED_{\text{Type}}(B[a/x]) \end{cases} \]

\( \RED(u)(\nu) \) holds if \( \forall a. \RED(A)(a) \Rightarrow \RED(B[a/x])(\nu a) \)

- \( \exists b. u \Downarrow b \) and then \( \RED(u)(\nu) \) holds if \( \exists b'. \nu \Downarrow b' \)

\[ t = \Pi(x : A) B \]

- \( \exists A, B. \begin{cases} \RED_{\text{Type}}(A) \\ \forall a. \RED(A)(a) \Rightarrow \RED_{\text{Kind}}(B[a/x]) \end{cases} \]

\( \RED(\Pi(x:A)B)(u) \) holds if \( \forall a. \RED(A)(a) \Rightarrow \RED(B[a/x])(ua) \)
Typability, reducibility and weak normalisation

**Proposition**

For all $T$, $t$, if $\text{RED}_{(\text{Type})}(T)$ then

1. $\text{RED}_{(T)}(t) \Rightarrow \text{WN}(t)$
2. $t \Downarrow b \Rightarrow \text{RED}_{(T)}(t)$

**Theorem**

If $\forall f. \text{RED}_{(\text{Type})}(\mathcal{F}(f)) \land \text{RED}_{(\mathcal{F}(f))}(f)$ then

$$\Gamma \vdash t : T \Rightarrow [\forall \sigma. \text{RED}_{\Gamma(x)}(\sigma(x)) \Rightarrow \text{RED}_{(\sigma(T))}(\sigma(t))]$$
Definition (Formal call)

We define \((f, (p_1, \ldots, p_m)) \succ (g, (u_1, \ldots, u_n))\) by:
- there is a \(k\) such that \(f \, p_1 \ldots p_k \rightarrow s\) is in \(\mathbb{R}\),
- \(\text{ar}(f) = m\), \(\text{ar}(g) = n\),
- \(g \, u_1 \ldots u_n\) is a subterm of \(s \, p_{k+1} \ldots p_m\).

Definition (Instantiated call)

\((f, (t_1, \ldots, t_m)) \tilde{\succ} (g, (v_1, \ldots, v_n))\) holds if there exists a substitution \(\sigma\) such that:
- \(\forall i. \exists p_i. t_i \leadsto^* \sigma(p_i)\),
- \(\forall i. \text{WN}(t_i)\),
- \(\forall j. v_j = \sigma(u_j)\)
- \((f, (p_1, \ldots, p_m)) \succ (g, (u_1, \ldots, u_n))\).
Theorem

If $\tilde{\succ}$ is well-founded and $\forall f. \text{RED}_{(\text{Type})}(\mathcal{F}(f))$ then

$\forall f. \text{RED}_{(\mathcal{F}(f))}(f)$
Define a reducibility predicate for weak normalisation.

Show that if every function in the signature is reducible then every typable term is reducible.

Show that if the call relation is well-founded and every type in the signature is reducible, then every function in the signature is reducible.

The size-change principle is used to show that the call relation is well-founded.

The reducibility of every type occurring in the signature is decidable.
Future work

- Strong normalisation (from Frédéric Blanqui’s work)
- Study decidability of type reducibility
- Enrichment of the Size-Change Principle
- Implementation of the Wahlstedt criterion