Proving termination in the $\lambda\Pi$-calculus modulo theory

How to use the Size-Change Principle

Guillaume Genestier

Tuesday, November 14th 2017

Dedukti

Dedukti is a multipurpose type-checker based on the $\lambda\Pi$-calculus modulo theory.

Example of rewrite rule

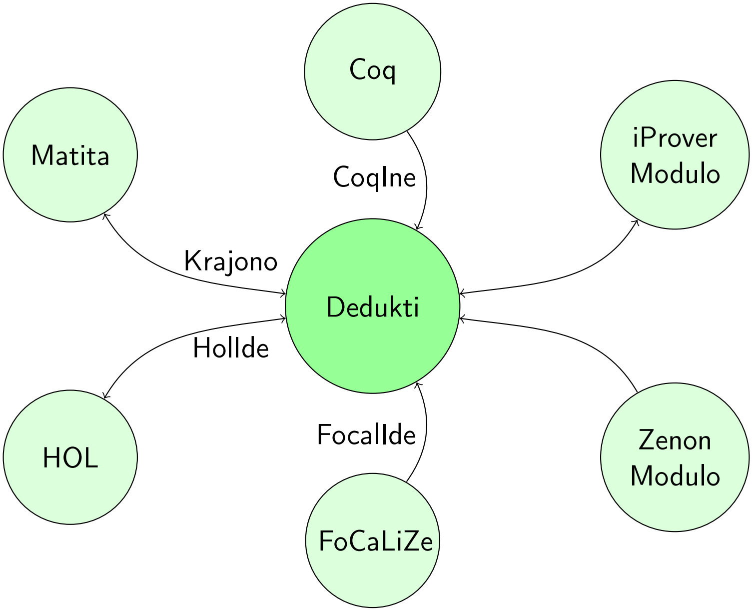
```
Nat : Type.
0 : Nat.
S : Nat -> Nat.

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



Contents

We consider a set of rewrite rules.

```
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).

def mult : Nat -> Nat -> Nat.
[]    mult 0     _ --> 0

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

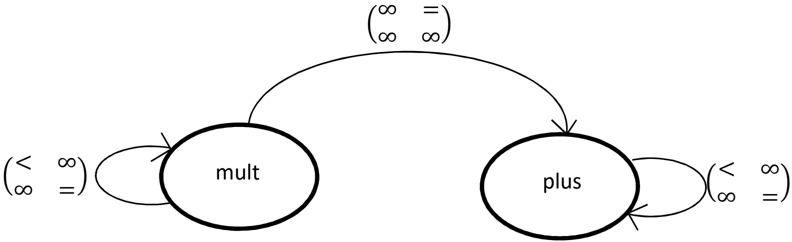
Size-Change Principle [Lee, Jones, Ben-Amram, 2001] [Lepigre, Raffalli, 2017]

We consider a set of rewrite rules.

`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).`
	$\begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix}$

`def mult : Nat -> Nat -> Nat.`
`[]    mult 0     _ --> 0`
`[m,n] mult (S m) n --> plus n (mult m n).`
	$\begin{pmatrix} \infty & \infty \\ = & \infty \end{pmatrix}$
	$\begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix}$



$\begin{pmatrix} \infty & = \\ \infty & \infty \end{pmatrix}$
$\begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix}$
mult
plus
$\begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix}$

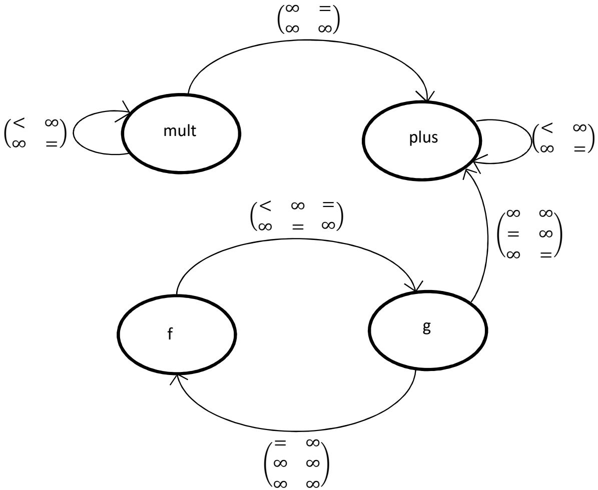
Mutually recursive definition

We define a function f such that

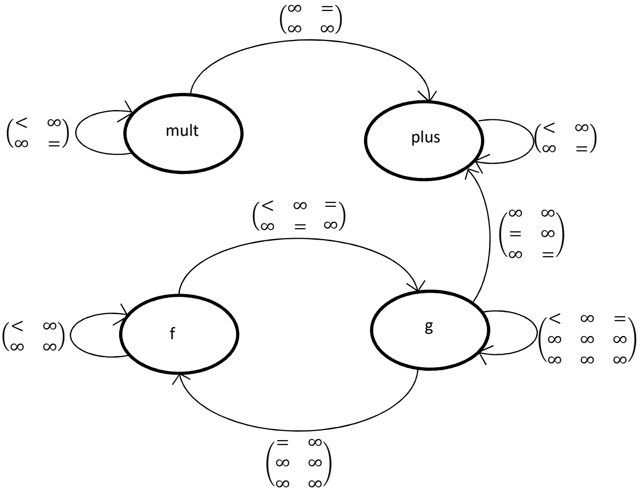
$$f : (a, b) \mapsto \left(\sum_{i=0}^{a} i \right) + b$$

`def f : Nat -> Nat -> Nat.` `def g : Nat -> Nat -> Nat -> Nat.`	
`[x]   f 0     x --> x` `[i,x] f (S i) x --> g i x (S i)`	$\begin{pmatrix} < & \infty & = \\ \infty & = & \infty \end{pmatrix}$
`[a,b,c] g a b c --> f a (plus b c)`	$\begin{pmatrix} = & \infty \\ \infty & \infty \\ \infty & \infty \end{pmatrix}$ $\begin{pmatrix} \infty & \infty \\ = & \infty \\ \infty & = \end{pmatrix}$

Call graph



Transitive closure of the call graph



An implementation of this algorithm has been done for *Dedukti*.

`List : Type.`	
`Nil : List.` `Cons : Nat -> List -> List.`	
`def map : (Nat -> Nat) -> List -> List.`	

`[]         map _ Nil          --> Nil.`
`[f, x, l] map f (Cons x l) -->`
`Cons (f x) (map f l).`
	$\begin{pmatrix} = & \infty \\ \infty & < \end{pmatrix}$
	$\begin{pmatrix} \infty \\ < \end{pmatrix}$

`Lamt : Type.`	
`def app : Lamt -> Lamt -> Lamt.` `Lam : ( Lamt -> Lamt ) -> Lamt.`	
`[f,t] app (Lam f) t --> f t.`	$\begin{pmatrix} \infty \\ = \end{pmatrix}$

Contents

Syntax

Definition (Terms)

We use :

- *x,y,z to denote variables,*
- *f, g, F to denote defined constants,*
- *c to denote element constructors,*
- *d to denote set constructors.*

$$t, \tau, u, v, l, r ::= x \mid \lambda(x : u).t \mid t\, u \mid c \mid f$$

$$T, U ::= \lambda(x : U).T \mid \Pi(x : U).T \mid U\, v \mid d \mid F$$

$$K ::= \text{Type} \mid \Pi(x : U).K$$

Definition (Contexts)

$$\Gamma, \Delta ::= [] \mid \Gamma, x : T$$

Sub-categories of terms

Definition (β-normal term)

We define this syntactical sub-category of terms as :

$$s ::= x\, s_1 \ldots s_n \mid h\, s_1 \ldots s_{\mathrm{ar}(h)} \mid \lambda(x : T).s$$

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)

$$\begin{aligned} b ::= &\; x\, t_1 \ldots t_n \textit{ where } \mathrm{NF}(t_i) \\ &\mid f\, t_1 \ldots t_n \textit{ where } \mathrm{NF}(f\, t_1 \ldots t_n) \textit{ and } n \geqslant \mathrm{ar}(f) \end{aligned}$$

Rewrite rules

- Each *rewrite rule* is of the form $f\ p_1 \dots p_k \to s$ where :
  - the p_i are constructor patterns,
  - s is β-normal,
  - the rule is *left-linear*, meaning that a free variable cannot appear twice in $f\ p_1 \dots p_k$.
- 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 σ_1 and σ_2 such that $\sigma_1(l_1) = \sigma_2(l_2)$.
- All the rules defining a symbol f have the same number of arguments, called the arity of f.

Property (Confluence)

Such a rewrite system is orthogonal, hence it is confluent.

Typability, reducibility and weak normalisation

Proposition

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

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

Theorem

If $\forall f.\, \mathrm{RED}_{(\mathrm{Type})}(\Sigma(f)) \wedge \mathrm{RED}_{(\Sigma(f))}(f)$ *then*

$$\Gamma \vdash t : T \Rightarrow \left[\forall \sigma.\, \mathrm{RED}_{\Gamma(x)}(\sigma(x)) \Rightarrow \mathrm{RED}_{(\sigma(T))}(\sigma(t))\right]$$

The reducibility predicate for kinds

- $\mathrm{RED}_{(\mathrm{Kind})}(K)$ holds if one of the following conditions occurs:
  - $K = \mathrm{Type}$
  - $\exists A, B. \begin{cases} K = \Pi(x : A)\ B \\ \mathrm{RED}_{(\mathrm{Type})}(A) \\ \forall a.\, \mathrm{RED}_{(A)}(a) \Rightarrow \mathrm{RED}_{(\mathrm{Kind})}\left(B\left[{}^{a}/{}_{x}\right]\right) \end{cases}$

The reducibility predicate for types

- $\mathrm{RED}_{(\mathrm{Type})}\,(U)$ holds if one of the following conditions occurs:
  - $\exists d, u_1, \dots, u_m. \begin{cases} U \Downarrow d\, u_1 \dots u_m \\ \Sigma(d) = \Pi(x_1 : T_1) \dots (x_k : T_k)\ \mathrm{Type} \\ \forall i.\, \mathrm{RED}_{(\mathrm{Type})}\,(T_i) \wedge \mathrm{RED}_{(T_i)}\,(u_i) \end{cases}$
  - $\exists A, B. \begin{cases} U \Downarrow \Pi(x : A)\ B \\ \mathrm{RED}_{(\mathrm{Type})}\,(A) \\ \forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow \mathrm{RED}_{(\mathrm{Type})}\,\left(B\left[{}^{a}/_{x}\right]\right) \end{cases}$
  - $\exists b. U \Downarrow b$
- $\mathrm{RED}_{(\Pi(x:A)\,B)}\,(U)$ holds if
  $\forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow \mathrm{RED}_{\left(B\left[{}^{a}/_{x}\right]\right)}\,(U\, a)$

The reducibility predicate for objects

- If $U \Downarrow d\, u_1 \ldots u_m$, then $\mathrm{RED}_{(U)}\,(v)$ holds if one of the following conditions occurs:
  - $\exists c, v_1, \ldots, v_n.\ \begin{cases} v \Downarrow c\, v_1 \ldots v_n \\ \Sigma(c) = \Pi(x_1 : U_1) \ldots (x_m : U_m)\,(d\, \tau_1 \ldots \tau_k) \\ \forall i.\, \mathrm{RED}_{(\mathrm{Type})}\left(U_i\left[{}^{v_1}/_{x_1}, \ldots, {}^{v_{i-1}}/_{x_{i-1}}\right]\right) \\ \forall i.\, \mathrm{RED}_{\left(U_i\left[{}^{v_1}/_{x_1}, \ldots, {}^{v_{i-1}}/_{x_{i-1}}\right]\right)}\,(v_i) \end{cases}$
  - $\exists b. v \Downarrow b$
- If $U \Downarrow \Pi(x : A)\ B$, then $\mathrm{RED}_{(U)}\,(v)$ holds if $\forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow \mathrm{RED}_{(B[{}^{a}/_{x}])}\,(v\, a)$
- If $U \Downarrow b$, then $\mathrm{RED}_{(U)}\,(v)$ holds if $\exists b'. v \Downarrow b'$

Such a predicate exists

Definition (Precedence on set constructors)

We define the relation on $\mathbb{D}$: $\preccurlyeq_\Sigma$ as the reflexive-transitive cloture of $d' \preccurlyeq_\Sigma d$ if:

- $\Sigma(d) = \Pi\overline{(x_i : T_i)}.\mathrm{Type}$ *and d' appears in a T_i*
- *or if there is a c such that $\Sigma(c) = \Pi\overline{(x_i : U_i)}.(d\,\bar{s})$ and d' appears in a U_i.*

Definition (Strictly positive set constructor)

A set constructor d is said strictly positive if:

- $\Sigma(d) = \Pi\overline{(x_i : T_i)}.\mathrm{Type}$ *and no symbole Σ-equivalent to d occurs in any T_i*
- *For all c such that $\Sigma(c) = \Pi\overline{(x_i : U_i)}.(d\,\bar{s})$, if the U_i are of the form $\Pi\overline{(y_j : V_j)}.V$ then no symbol Σ-equivalent to d appears in a V_j.*

Such a predicate exists

Definition (elementary type interpretation)

Let $\mathcal{A} = \{d_i\}_{i\in\{1,\dots,n\}}$ *be a* $\approx_\Sigma$*-equivalence class.*

$$F_\mathcal{A} : \prod_{i=1}^{n} \mathcal{P}(\Lambda) \to \prod_{i=1}^{n} \mathcal{P}(\Lambda)$$

$$(X_i)_i \mapsto \left(\{\, u \mid \exists b. u \Downarrow b\} \cup \left\{ u \;\middle|\; \begin{array}{c} u \Downarrow c\, \bar{v} \\ \Sigma(c) = \Pi\overline{(x_j : U_j)}.(d\, \bar{s}) \\ \forall j. v_j \in R_{U_j} \end{array} \right\} \right)_{i\in\{1,\dots,n\}}$$

where $R_\alpha(\bar{X}) = \begin{cases} [\![d]\!] & \text{if} \quad \alpha = d\,\bar{s}, \text{ where } \forall u \in \mathcal{A}. d \prec_\Sigma u \\ X_i & \text{if} \quad \alpha = d_i\,\bar{s} \\ R_{t_1} \to R_{t_2} & \text{if} \quad \alpha = \Pi(x : t_1).t_2 \end{cases}$

where, for $d \in \mathcal{A}$, $[\![d]\!]$ is the least fixpoint of $F_\mathcal{A}$ and

$$A \to B = \{\, t \mid \forall u \in A. t\, u \in B\}\,.$$

Call relation

Definition (Formal call)

We define $(f, (p_1, \ldots, p_m)) \succ (g, (u_1, \ldots, u_n))$ *by :*

- $f\ p_1 \ldots p_m \to s$ *is a rewrite rule declared by the user,*
- $\mathrm{ar}(f) = m$, $\mathrm{ar}(g) = n$,
- $g\ u_1 \ldots u_n$ *is a subterm of* s.

Definition (Instantiated call)

$(f, (t_1, \ldots, t_m)) \widetilde{\succ} (g, (v_1, \ldots, v_n))$ *holds if there exists a substitution* σ *such that :*

- $\forall i. \exists p_i. t_i \rightsquigarrow^* \sigma(p_i)$,
- $\forall i. \mathrm{WN}(t_i)$,
- $\forall j. v_j = \sigma(u_j)$
- $(f, (p_1, \ldots, p_m)) \succ (g, (u_1, \ldots, u_n))$.

Final result, introducing the Size-Change Principle

Theorem

If $\widetilde{\succ}$ *is well-founded and* $\forall f.\, \mathrm{RED}_{(\mathrm{Type})}(\Sigma(f))$ *then*

$$\forall f.\, \mathrm{RED}_{(\Sigma(f))}\,(f)$$

Theorem (Size-Change induces well-foundedness [Wahlstedt, 2007])

If the rewrite rules satisfy the Size-Change Principle, with the formal call order $\succ$*, then the instantiated call order* $\widetilde{\succ}$ *is well-founded.*

Future work

- First of all, finish the implementation
- Relax the constraints on rewrite rules
- Strong normalisation
- Study decidability of type reducibility
- Enrichment of the Size-Change Principle