foetus - Termination Checker for Simple Functional Programs

Andreas Abel

June 15, 1998

foetus is an abbreviation of
"Funktionale – Obgleich Eingeschränkt – Termination Untersuchende Sprache"

Example 1: List flattening.

```
f([])    = []
f(x::xs) = g(x,xs)

g([],   ys) = f(ys)
g(x::xs,ys) = x::g(xs,ys)
```

Example 2.

```
f(0,  y)   = 0
f(x,  0)   = 0
f(x+1,y+1) = g(x,y+1) + f(x+3,y)

g(0,  y)   = 0
g(x,  0)   = 0
g(x+1,y+1) = f(x+1,y+1) + g(x,y+2)
```

Orders and Termination

Definition. An irreflexive order $\prec$ on X is *well founded* if there are no infinite decending chains $x_1 \succ x_2 \succ \ldots$ $(x_1, x_2, \ldots \,\epsilon\, X)$.

Proposition. A function $f : \tau \to \sigma$ *terminates* if there is a well founded order $\prec$ on τ such that $y \prec x$ in each recursive call $f(x) \to f(y)$.

Definition. Given ordered types $(\tau_1, <_1), \ldots, (\tau_n, <_n)$ we define the *lexical order* $<_\pi$ on $\tau := \tau_1 \times \ldots \times \tau_n$ in respect of the permutation $\pi \,\epsilon\, S_n$ as

$$\begin{array}{rcl} x <_\pi y & :\Leftrightarrow & x_{\pi(1)} <_{\pi(1)} y_{\pi(1)} \quad \text{or} \\ x_{\pi(1)} =_{\pi(1)} y_{\pi(1)} & \text{and} & (x_{\pi(2)} <_{\pi(2)} y_{\pi(2)} \quad \text{or} \\ x_{\pi(2)} =_{\pi(2)} y_{\pi(2)} & \text{and} & (\ldots \\ \ldots & \text{and} & (x_{\pi(n)} <_{\pi(n)} y_{\pi(n)}) \ldots) \end{array}$$

$(x, y \,\epsilon\, \tau)$.

Examples for Structural Orders.

Binary trees.

```
Leaf   : 1 -> Tree
Branch : Tree * Tree -> Tree

l,r: Tree
t = Branch(l,r)  =>  l < t, r < t
```

Natural numbers.

```
0 : 1 -> Nat
S : Nat -> Nat

x : Nat  =>  x < S(x)
```

foetus **Term Syntax**

In the following x, x_1, x_2, ... denote variables, C, C_1, C_2, ... constants and u, t, t_1, t_2, ... foetus terms.

$$
\begin{array}{lcll}
t & :: & x & \textit{variable} \\
 & | & [x]t & \textit{lambda} \\
 & | & tu & \textit{application} \\
 & | & C(t) & \textit{constructor} \\
 & | & \text{case } t \text{ of } \{\ C_1 x_1 \Rightarrow t_1 | \ldots | C_n x_n \Rightarrow t_n\ \} & \textit{pattern matching} \\
 & | & (C_1 = t_1, \ldots, C_n = t_n) & \textit{tuple} \\
 & | & t.C & \textit{projection} \\
 & | & \text{let } x_1 = t_1, \ldots, x_n = t_n \text{ in } t & \textit{let} \\
 & | & (\mathrm{t}) & \textit{(extra parentheses)}
\end{array}
$$

Example 1 - foetus code

```
f = [l]case l of
    { Nil z => Nil()
    | Cons p => g p.HD p.TL },
g = [l][ls]case l of
    { Nil z => f ls
    | Cons p => Cons(HD=p.HD, TL=(g p.TL ls)) };
```

Example 2 - foetus code

```
f = [x][y]case x of
    { O z  => O()
    | S x' => case y of
        { O z  => O()
        | S y' => add (g x' y) (f (S(S(x))) y') } },
g = [x][y]case x of
    { O z  => O()
    | S x' => case y of
        { O z  => O()
        | S y' => add (f x y) (g x' (S(y))) } };
```

foetus Type System

The judgement

$$\Gamma \vdash t : \sigma$$

means "in context Γ term t is of type σ".

Lambda abstraction.

$$\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash [x]t : \sigma \to \tau} \to -\mathrm{i}$$

Application.

$$\frac{\Gamma \vdash t : \sigma \to \tau \quad \Gamma \vdash u : \sigma}{\Gamma \vdash tu : \sigma} \to -\mathrm{e}$$

Constructor.

$$\frac{\Gamma \vdash t : \sigma_i}{\Gamma \vdash C_i(t) : \{C_1 : \sigma_1 | \ldots | C_n : \sigma_n\}} \{\} - \mathrm{i}$$

Pattern matching.

$$\frac{\Gamma \vdash t : \{C_1 : \sigma_1 | \ldots | C_n : \sigma_n\} \qquad \Gamma, x_i : \sigma_i \vdash u_i : \sigma \text{ for all } 1 \leq i \leq n}{\Gamma \vdash \text{case } t \text{ of } \{C_1(x_1) \Rightarrow u_1 | \ldots | C_n(x_n) \Rightarrow u_n\} : \sigma} \{\} - \mathrm{e}$$

Tupels.

$$\frac{\Gamma \vdash t_i : \sigma_i \text{ for all } 1 \leq i \leq n}{\Gamma \vdash (C_1 = t_1, \ldots, C_n = t_n) : (C_1 : \sigma_1, \ldots, C_n : \sigma_n)} () - \mathrm{i}$$

Projection.

$$\frac{\Gamma \vdash t : (C_1 : \sigma_1, \ldots, C_n : \sigma_n)}{\Gamma \vdash t.C_i : \sigma_i}\ () - \mathrm{e}$$

Let.

$$\frac{\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash t_i : \sigma_i \text{ for all } 1 \leq i \leq n \qquad \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash u : \tau}{\Gamma \vdash \text{let } x_1 = t_1; \ldots; x_n = t_n \text{ in } u : \tau}\ \text{let}$$

Analyzer

- evaluation under λ
- no stepping into defined expressions

Dependencies.

$$x = C(y) \quad \Rightarrow \quad y < x \tag{1}$$

$$y \overset{=}{<} x \quad \Rightarrow \quad y.L \overset{=}{<} x \tag{2}$$

$$y \overset{=}{<} x \quad \Rightarrow \quad (y\,a) \overset{=}{<} x \tag{3}$$

Example 3.

```
addord = [x][y]case x of
        { O o => y
        | S x' => S(addord x' y)
        | Lim f => Lim([z]addord (f z) y) };
```

Call Matrix

$R = \{<, =, ?\}$.

$+$	$<$	$=$	$?$
$<$	$<$	$<$	$<$
$=$	$<$	$=$	$=$
$?$	$<$	$=$	$?$

$\cdot$	$<$	$=$	$?$
$<$	$<$	$<$	$?$
$=$	$<$	$=$	$?$
$?$	$?$	$?$	$?$

With $+$ and $\cdot$ R forms a communtative rig with 0-element $?$ and 1-element $=$.

Definition. A *call matrix* is a matrix over R with no more than one element different from $?$ per row.

$$\mathrm{CM}(n, m) := \{(a_{ij})\,\epsilon\,R^{n \times m} : \forall i \forall j \forall k \neq j (a_{ij} = ? \vee a_{ik} = ?)\}$$

Call graph

For each $i\,\epsilon\,\mathbb{N}$ we assume a set $F^{(i)} = \{f^{(i)}, g^{(i)}, h^{(i)}, ...\}$ of identifiers for functions of arity i, $\mathcal{F} = \biguplus_{i\,\epsilon\,\mathbb{N}} F^{(i)}$.

Definition. We form the set of *calls* as follows

$$\mathcal{C} = \{(f^{(n)}, g^{(m)}, A) : f^{(n)} \,\epsilon\, F^{(n)}, g^{(m)} \,\epsilon\, F^{(m)}, A \,\epsilon\, \mathrm{CM}(m,n)\}$$

On calls we define the partial operation *combination of calls*

$$\circ : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$$
$$\Big((g^{(m)}, h^{(l)}, B), (f^{(n)}, g^{(m)}, A)\Big) \mapsto (f^{(n)}, h^{(l)}, BA)$$

Definition. A *call graph* is a graph (V, E) with vertices $V = \mathcal{F}$ and edges $E \subset_{\text{finit}} \mathcal{C}$. A call graph is *complete* if

$$E \circ E \subseteq E$$

Proposition. (Completion algorithm) Be (V, E) a call graph, (V, E') its completion and $(E_n)_{n \,\epsilon\, \mathbb{N}}$ a sequence of sets of calls defined as follows:

$$\begin{aligned} E_0 &= E \\ E_{n+1} &= E_n \cup (E_n \circ E) \end{aligned}$$

Then there is a $n \,\epsilon\, \mathbb{N}$ so that

$$E' = E_n = E_{n+1} = E_{n+2} = \ldots$$

Lexical Order

Definition. Be (V, E) a complete call graph and $f^{(i)}$ a function of arity i. We call

$$E_{f^{(i)}} := \{\Delta(C) : (f^{(i)}, f^{(i)}, C) \,\epsilon\, E\} \subset R^i$$

the *recursion behaviour* of function $f^{(i)}$.

Definition & Algorithm. Be B the recursion behaviour of a given function. We call the permutation $\pi \,\epsilon\, S_n$ a *termination order* on B if $|B| = 0$ or

$$\begin{aligned} &\exists r \,\epsilon\, B : r_{\pi(0)} = < \\ &\wedge\ \nexists r \,\epsilon\, B : r_{\pi(0)} = \mathbf{?} \\ &\wedge\, \pi'_0 \,\epsilon\, S_{n-1} \text{ termination order on } B' := \{r'_{\pi(0)} : r_{\pi(0)} \neq <\} \subset R^{n-1} \end{aligned}$$

whereas $\pi'_i = [k_0 \ldots k_{i-1} k_{i+1} \ldots k_{n-1}] \,\epsilon\, S_{n-1}$ given $\pi = [k_0 \ldots k_{n-1}] \,\epsilon\, S_n$ and $r'_i = (k_0, \ldots, k_{i-1}, k_{i+1}, \ldots, k_{n-1}) \,\epsilon\, R$ given $r = (k_0, \ldots, k_{n-1}) \,\epsilon\, R^n$.

Extentions

- let assignments

```
case list of
        {Cons pair => let
                hd = pair.HD,
                tl = pair.TL in ...
```

- tuple handling
- parameter permutation
- coinductive types