Type Theory with First-Order Data Types and Size-Change Termination

TYPES 2004

David Wahlstedt

davidw@cs.chalmers.se

The **halting** problem is undecidable . . .

Known characterizations of termination:

Primitive recursion

$$\begin{aligned} \mathrm{f}(\vec{x}, \mathbf{S}(n)) &= \mathrm{g}(\mathrm{f}(\vec{x}, n), \vec{x}, n) \\ \mathrm{f}(\vec{x}, \mathbf{0}) &= \mathrm{h}(\vec{x}) \end{aligned}$$

More general schema needed

$$\begin{aligned} \min(\mathbf{S}(m), \mathbf{S}(n)) &= \mathbf{S}(\min(m, n)) \\ \min(\mathbf{0}, \mathbf{S}(n)) &= \mathbf{0} \\ \min(m, \mathbf{0}) &= \mathbf{0} \end{aligned}$$

A general approach to "Structural Termination":

Size-Change Termination (SCT)
(C.S. Lee, N.D. Jones and A.M. Ben-Amram '01)

- More than primitive recursion
- Subsumes lexicographical termination
- Simple algorithm with non-trivial correctness proof (Ramsey's Theorem involved)

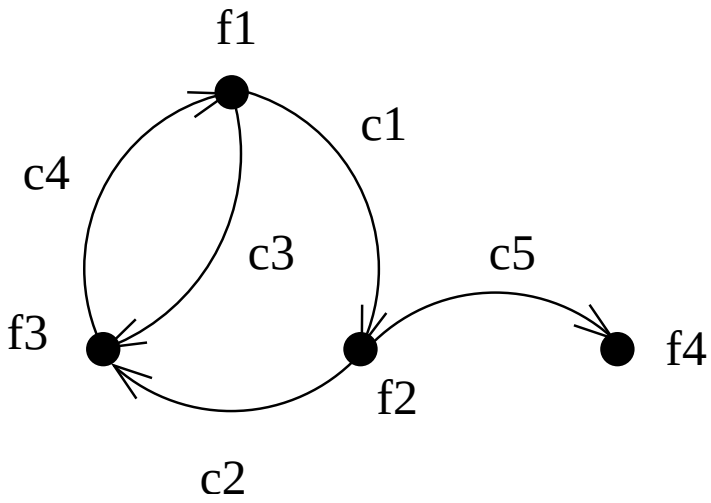
Size-change termination criterion:

All the infinite call sequences

contain an infinitely decreasing thread.

Program **Call Graph**

$$\begin{array}{lcl} \mathrm{f}_1(p_{11},\ \ldots,\ p_{1n}) & = & a_1 \\ \vdots & \vdots & \vdots \\ \mathrm{f}_n(p_{m1},\ \ldots,\ p_{mn}) & = & a_m \end{array} \quad \Rightarrow$$

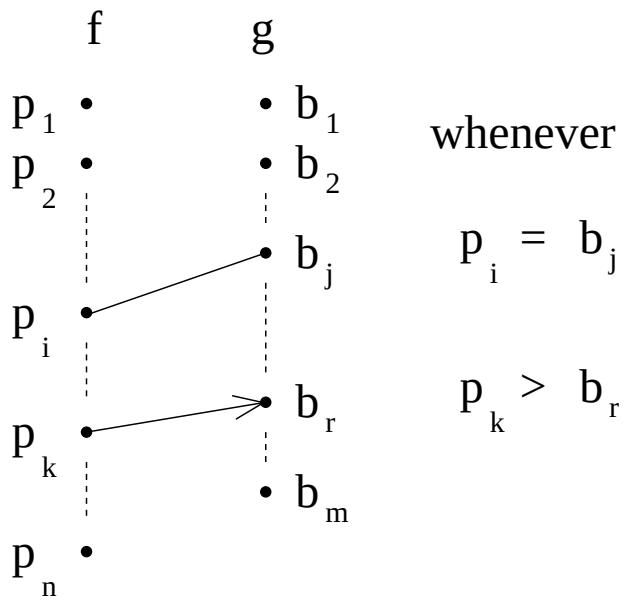


For each **call**, c, to f_j in a_i there is an arc from f_i to f_j.

For each recursive call c,
relate formal parameters and actual parameters:

Recursive call c $\Rightarrow$ **Size-change graph** G_c

$$\mathrm{f}(p_1, \ \ldots \ , p_n) =$$
$$\ldots$$
$$\mathrm{g}(b_1, \ \ldots \ , b_m)$$
$$\ldots$$

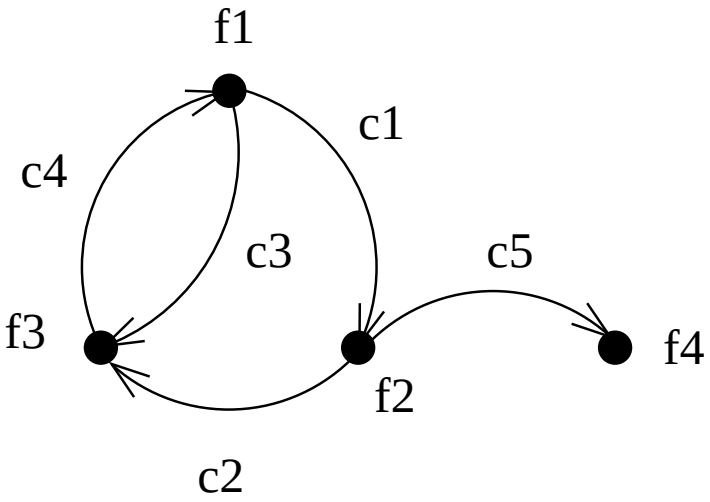


Take $>$ to be the **structural** size-change relation

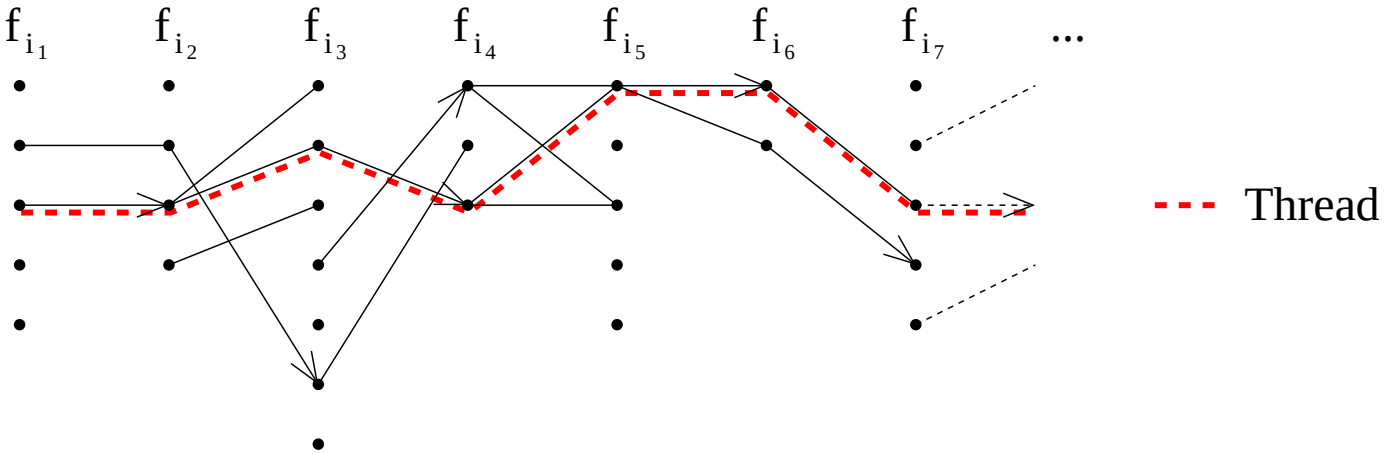
$$\frac{}{\mathbf{c}(t_1,\ \ldots\ ,t_n) > t_k} \qquad \frac{t_k > t}{\mathbf{c}(t_1,\ \ldots\ ,t_n) > t}$$

where $\mathbf{c}$ is a constructor.

Call graph

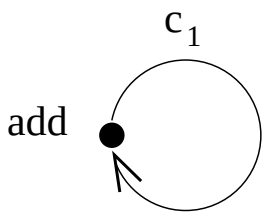


Path

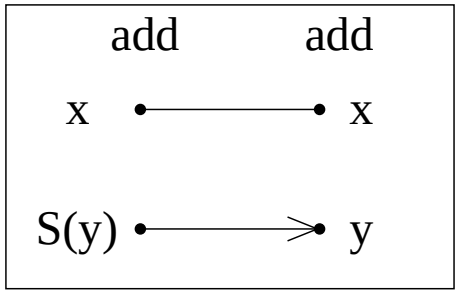


Addition

$$\begin{aligned} \mathrm{add}(x, \mathbf{0}) &= \mathbf{0} \\ \mathrm{add}(x, \mathbf{S}(y)) &= \mathbf{S}(^1\mathrm{add}(x, y)) \end{aligned}$$



Size−change graph:



because

x = x and S(y) > y

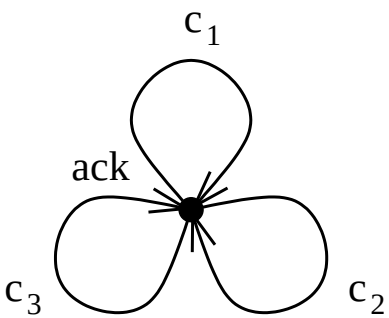
Ackermann function (Rózsa Péter's version)

$$\begin{aligned}
\mathrm{ack}(\mathbf{0}, y) &= \mathbf{S}(y) \\
\mathrm{ack}(\mathbf{S}(x), \mathbf{0}) &= {}^{1}\mathrm{ack}(x, \mathbf{S}(\mathbf{0})) \\
\mathrm{ack}(\mathbf{S}(x), \mathbf{S}(y)) &= {}^{2}\mathrm{ack}(x, {}^{3}\mathrm{ack}(\mathbf{S}(x), y))
\end{aligned}$$

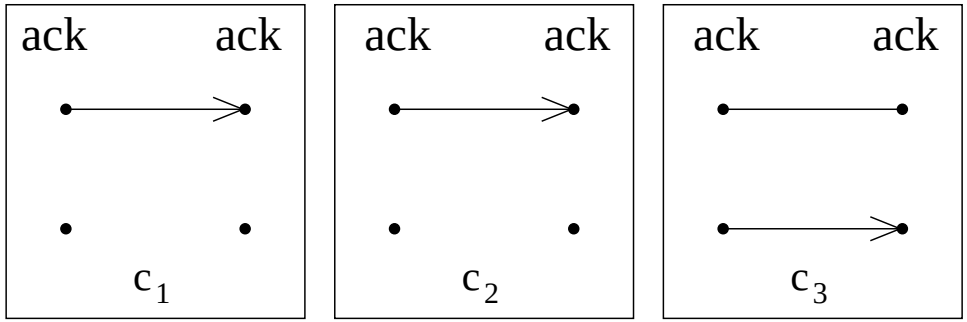
We compare parameters for each call:

$$\begin{aligned}
c_1 : \quad & \mathbf{S}(x) > x \\
c_2 : \quad & \mathbf{S}(x) > x \\
c_3 : \quad & \mathbf{S}(x) = \mathbf{S}(x), \quad \mathbf{S}(y) > y.
\end{aligned}$$

Call graph:



Size−change graphs:



Quicksort — non-structural

$$\begin{aligned}
&\text{qsort}[] = [] \\
&\text{qsort}(x : xs) = \\
&\qquad {}^{1}\text{append}({}^{2}\text{qsort}({}^{3}\text{filter_small}(x, xs)), \\
&\qquad\qquad x : {}^{4}\text{qsort}({}^{5}\text{filter_big}(x, xs)))
\end{aligned}$$

No structural decrease in calls c_2 and c_4.

Another non-structurally terminating program:

Tree flattening — rearrange instead of descreasing.

$$\text{data Tree } a = \mathbf{Leaf}$$
$$\mid \mathbf{Node}\ a\ (\text{Tree } a)\ (\text{Tree } a)$$

$$\text{flatten}(\mathbf{Leaf}) = []$$
$$\text{flatten}(\mathbf{Node}\ a\ \mathbf{Leaf}\ t) = a : {}^{1}\text{flatten}\ t$$
$$\text{flatten}(\mathbf{Node}\ a\ (\mathbf{Node}\ b\ t_1\ t_2)\ t_3) =$$
$${}^{2}\text{flatten}(\mathbf{Node}\ b\ t_1\ (\mathbf{Node}\ a\ t_2\ t_3))$$

The call c_2 has no decreasing parameter.

Let us use size-change termination in a proof system !

Martin-Löf's logical framework '86
(in Nordström, Petersson, Smith '90)

$$\begin{array}{llll} \mathit{Terms} & t, u, v & ::= & x \mid t\ u \mid \lambda x.t \\ \mathit{Types} & T, U, V & ::= & \text{Set} \mid \text{El}\ t \mid \text{Fun}\ T\ (\lambda x.U) \end{array}$$

The notation $(x : T) \to U$ is shorthand for Fun T $(\lambda x.U)$.

An open system—we extend the language:

$$\begin{array}{lllll} \textit{Terms} & t, u, v & ::= & x \mid t\ u \mid \lambda x.t & \\ & & & \mid\ \mathbf{c}(t_1,\ \ldots\ , t_n) \mid \mathrm{f}(t_1,\ \ldots\ , t_n) \mid \mathrm{D} \end{array}$$

Function constants f, constructors $\mathbf{c}$, and set codes D.

Restricted sublanguage

$$\begin{array}{lllll} \text{FO-Terms} & a & ::= & x \mid \mathbf{c}(t_1,\ \ldots\ , t_n) \mid \mathrm{f}(t_1,\ \ldots\ , t_n) \mid \mathrm{D} \\ \text{Patterns} & p & ::= & x \mid \mathbf{c}(p_1,\ \ldots\ , p_n) \end{array}$$

$$\textit{Rewrite rules} \qquad \mathrm{f}(p_1,\ \ldots\ , p_n) = a$$

A formalism for reasoning about inductively defined objects.

Judgement forms

$$\vdash U \text{ Type} \qquad \vdash t : U$$

Propositions–as–types

$\vdash t : U$ interpreted as t *is a proof of the proposition* U.

Type-valued functions

$\lambda x.\text{El}\ x$ returns different type dependent on the argument.

We can write propositional functions by pattern-matching. (Smith '89)

Contribution

We prove normalization for a combination of

- Martin-Löf's logical framework.
- First-order recursive pattern-matching rules, justified by size-change termination.
- First-order mutually recursive data-types.

- The new definitions are stored in a *Signature* $(\mathcal{D}, \mathcal{F}, \mathcal{R})$

  $\mathcal{D}$ Mutually recursive first-order data type definition.
  All data types are given simultaneously.

  $\mathcal{F}$ Function constants with typing
  $\mathrm{f} : (x_1 : \mathrm{El}\ a_1,\ \ldots,\ x_n : \mathrm{El}\ a_n) \to A$
  with A on the form $\mathrm{El}\ a \mid \mathrm{Set}$.

  $\mathcal{R}$ Collection of mutually recursive pattern-matching rules $\mathrm{f}(p_1,\ \ldots\ , p_n) = a$ for the constants in $\mathcal{F}$.

- Valid signature—correct types in $\mathcal{F}$
  and well-typed computation rules in $\mathcal{R}$.

Theorem

If

1. The pattern-matching rules in $\mathcal{R}$ satifies SCT,
2. the signature is valid,
3. and $() \vdash t : T$,

then t has a normal form.

A proof about sub-typing:

We define the propositional function

$$\text{subType} : (x_1 : \text{El Num},\ x_2 : \text{El Num}) \to \text{El Bool}$$

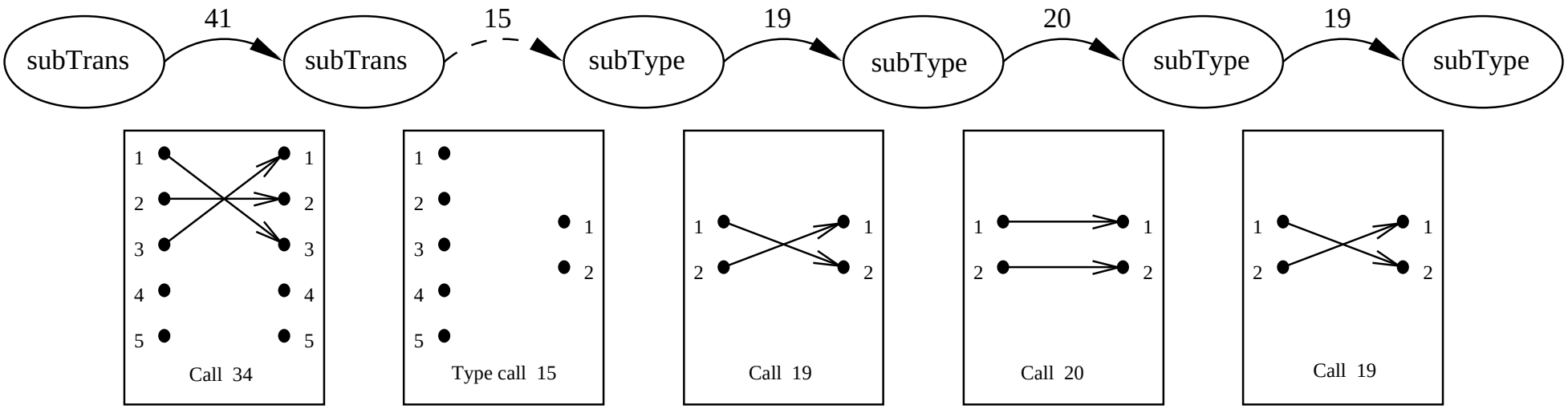
the recursive case is

$$\text{subType}(\mathbf{Arrow}(x_1,\ x_2),\ \mathbf{Arrow}(y_1,\ y_2)) = \text{and}(\text{subType}(y_1,\ x_1),\ \text{subType}(x_2,\ y_2))$$

We can prove that the relation is transitive using the constant,

$$\begin{aligned}\text{subTrans} : (&x : \text{El Num},\ y : \text{El Num},\ z : \text{El Num},\\ &h_1 : \text{El } (\text{T}(\text{subType}(x,\ y))),\\ &h_2 : \text{El } (\text{T}(\text{subType}(y,\ z)))\\ &) \rightarrow \text{El } (\text{T}(\text{subType}(x,\ z)))\end{aligned}$$

Path:



Thread: 2 > 1 > 1 > 2