Termination analysis of first order programs

Oleksandr Shturmov

January 23, 2012

Overview

$$H(M,x) = \begin{cases} \textit{true} & M \text{ halts on } x, \\ \textit{false} & M \text{ does not halt on } x. \end{cases}$$

$$H(M,x) = \begin{cases} \mathit{true} & M \text{ halts on } x, \\ \mathit{false} & M \text{ does not halt on } x. \end{cases}$$

$$F(M) = \begin{cases} \mathit{true} & H(M,M) \rightsquigarrow \mathit{false}, \\ \mathit{false} & H(M,M) \rightsquigarrow \mathit{true}. \end{cases}$$

Consider $F(F)$.

$$H(M,x) = \begin{cases} \textit{true} & M \text{ halts on } x, \\ \textit{false} & M \text{ does not halt on } x, \\ \textit{unknown} & M \text{ may or may not halt on } x. \end{cases}$$

$$H(M,x) = \begin{cases} \textit{true} & M \text{ halts on } x, \\ \textit{unknown} & M \text{ may or may not halt on } x. \end{cases}$$

"Unfortunately, many have drawn too strong of a conclusion about the prospects of automatic program termination proving and falsely believe we are always unable to prove termination, rather than more benign consequence that we are unable to always prove termination."

[Cook et al., 2011]

Δ, data

Δ, syntax


```
    <program> ::= <clause>+ <expression>
 <expression> ::= <element> ( '.' <expression> ) ?
    <element> ::= '0' | '(' <element> ')' | <name> | <application>
<application> ::= <name> <expression>*
     <clause> ::= <name> <pattern>* ':=' <expression>';'
    <pattern> ::= <pattern-element> ( '.' <pattern> ) ?
<pattern-element> ::= '0' | '_' | '(' <pattern> ')' | <name>
       <name> ::= ['a'-'z'] ( ['-' 'a'-'z']* ['a'-'z'] ) ?
```

Symbols

Description	**Instance**	**Finite list**	**Space**
Expression	x	X	$\mathbb{X}$
Element (of an expression)	e	E	$\mathbb{E}$
Function	f	F	$\mathbb{F}$
Clause	c	C	$\mathbb{C}$
Pattern	p	P	$\mathbb{P}$
Value (think “binary”)	b	B	$\mathbb{B}$
Name (think “variable”)	v	V	$\mathbb{V}$
Program (p was taken)	r	R	$\mathbb{R}$

Δ, syntax

(14)

```
    <program> ::= <clause>+ <expression>
 <expression> ::= <element> ( '.' <expression> ) ?                    x
    <element> ::= '0' | '(' <element> ')' | <name> | <application>    e
<application> ::= <name> <expression>*                                ⟨v, X⟩
     <clause> ::= <name> <pattern>* ':=' <expression>';'              c = ⟨v, P, x⟩
    <pattern> ::= <pattern-element> ( '.' <pattern> ) ?               p
<pattern-element> ::= '0' | '_' | '(' <pattern> ')' | <name>          p
       <name> ::= ['a'-'z'] ( ['-' 'a'-'z']* ['a'-'z'] ) ?            v
```

Functions in Δ

$$f = \langle v, C \rangle \quad \text{s.t.} \quad \forall \ \langle v_1, P_1, \_ \rangle, \langle v_2, P_2, \_ \rangle \in C \ (v_1 = v_2 = v) \wedge (|P_1| = |P_2|)$$

Pattern matching is ensured **exhaustive** at compile time.

$$\forall\, b \in \mathbb{B}\ \exists\, c \in C\ c \succ b$$

Programs in Δ

$$r = \langle F, x \rangle$$

Demo

Disjoint shapes

$$s_1 \cap s_2 = \emptyset \quad \text{iff} \quad B_1 \cap B_2 = \emptyset$$

where

$$s_1, s_2 \in \mathbb{S} \wedge B_1 = \{b \mid b \in \mathbb{B} \wedge b \succ s_1\} \wedge B_2 = \{b \mid b \in \mathbb{B} \wedge b \succ s_2\}$$

Observed mistakes

List indexing

Lists are sometimes 0-indexed rather than 1-indexed.
$\forall \{i \mid 0 \geq i < |P|\}$ should obviously be $\forall \{i \mid 0 < i \leq |P|\}$.

References

[Cook et al., 2011] B. Cook, A. Podelski & A. Rybalchenko, *Proving program termination*, Communications ACM Vol. 54(5), 2011, pp. 88–98.