

Termination analysis of first order programs

Oleksandr Shturmov

January 25, 2012

Overview

References

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

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

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

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 the more benign consequence that we are unable to always prove termination.”

[Cook et al., 2011]

“The **size-change termination principle** for a first-order functional language with well-founded data is: a program terminates on all inputs if *every infinite call sequence* (following program control flow) would cause an infinite descent in some data values.”

[Lee et al., 2001]



An untyped, call-by-value functional first-order language.

Δ , data

```
<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'] ) ?
```

| 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} |

| | |
|--|-------------------------------|
| $\langle \text{program} \rangle ::= \langle \text{clause} \rangle^+ \langle \text{expression} \rangle$ | |
| $\langle \text{expression} \rangle ::= \langle \text{element} \rangle (\text{'.'} \langle \text{expression} \rangle) ?$ | x |
| $\langle \text{element} \rangle ::= \text{'0'} \mid \text{'('} \langle \text{element} \rangle \text{'('} \mid \langle \text{name} \rangle \mid \langle \text{application} \rangle$ | e |
| $\langle \text{application} \rangle ::= \langle \text{name} \rangle \langle \text{expression} \rangle^*$ | $\langle v, X \rangle$ |
| $\langle \text{clause} \rangle ::= \langle \text{name} \rangle \langle \text{pattern} \rangle^* \text{' := ' } \langle \text{expression} \rangle \text{' ; '}$ | $c = \langle v, P, x \rangle$ |
| $\langle \text{pattern} \rangle ::= \langle \text{pattern-element} \rangle (\text{'.'} \langle \text{pattern} \rangle) ?$ | p |
| $\langle \text{pattern-element} \rangle ::= \text{'0'} \mid \text{'_'} \mid \text{'('} \langle \text{pattern} \rangle \text{'('} \mid \langle \text{name} \rangle$ | p |
| $\langle \text{name} \rangle ::= [\text{'a'-'z'}] ([\text{'-' 'a'-'z'}]^* [\text{'a'-'z'}]) ?$ | 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

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

Observed mistakes

List indexing

Lists are sometimes 0-indexed rather than 1-indexed.

$\forall \{i \mid 0 \leq i < |P|\}$ 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, 88–98.
- [Lee et al., 2001] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. 2001. *The size-change principle for program termination*, In Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '01), ACM, New York, NY, USA, 81–92.