Semantics of Termination

Patricia Peratto
psperatto@adinet.com.uy
Uruguay

June 2007

Abstract

This work presents proof rules for termination and non-termination for a subset of a **C-like** language that consists of the statements : **assignation**, **composition**, **if**, **skip** and **while** and rules for termination of **function calls**.

1 Introduction

There are many books that treat the natural semantics and the structural operational semantics of **C-like** languages [1, 2, 3].

They usually define when a statement **S** on a state **s** *terminates* or *loops*, but does not give a way for proving the *termination* or *non-termination* of a program.

We define a formal system that allows to make these proofs.

An exception is [1] that gives rules for termination of statements except for **while**.

Work on termination of programs has been developed for logic programming, functional programming and concurrent programmming but not for imperative programming.

Examples of the work developed are [4, 5, 10] for logic programming, [9, 6, 7] for functional programming and [8] for concurrent programming. More pointers to the literature can be find in these works.

The outline of our paper is as follows: section 2 presents the abstract syntax of the language (this includes arithmetic and boolean expressions and statements). We don't present the semantics of expressions nor of statements because is widely presented in the literature. In section 3 we present the formal systems of our concern and examples of proofs in particular programas. In section 4 we extend the system to manage termination of functions. From the semantics of termination for functions one can infer a natural semantics for functions that was not presented before. In section 5 we present further work.

2 Abstract syntax of the language

We use a syntactic notation based on BNF. Parenthesis can be used (not indicated in our BNF) to solve ambiguities and uniquely determine the corresponding parse tree.

We have the following Syntactic Categories and meta-variables ranging over them

n will range over numerals, **Num**
a will range over arithmetic expressions, **AExp**
x will range over arithmetic variables, **AVar**
b will range over boolean expressions, **BExp**
S will range over statements, **Stm**

The meta-variables can be subscripted for example $a, a_1, a_2, \ldots$ all stand for arithmetic expressions.

Definition 2.1. *Abstract syntax for Arithmetic Expressions*

$a ::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2$

Definition 2.2. *Abstract syntax for Boolean Expressions*

$b ::= \textbf{true} \mid \textbf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \vee b_2 \mid b_1 \wedge b_2$

true and **false** stand for constant truth values.

Definition 2.3. *Abstract syntax for Statements*

$S ::= x := a \mid \textbf{skip} \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \mid \textbf{while } b \textbf{ do } S$

3 Logics for termination and non-termination of programs

We present logics for *termination* and *non-termination* of programas together with examples of their application.

In what follows A stands for the semantics function for arithmetic expressions and B for the semantic function for boolean expressions.

3.1 Logic for termination

Let S stand for a program. We define the relationship S in a state s terminates on state s'. Table 1 presents the semantics of termination of statements.

Theorem 3.1. *Equivalence I*
An statement **S** *on a state* **s** terminates *in an state* s' *according to the natural semantics as defined in [2] if and only if* $(S)s$ *terminates on* s'. **proof** *straightforward.*

Theorem 3.2. *Equivalence II*
An statement **S** *on a state* **s** terminates *in an state* s' *according to the structural operational semantics as defined in [2] if and only if* $(S)s$ *terminates on* s'. **proof** *straightforward.*

Example 3.3. *The factorial program*
We prove the termination of the factorial program when is evaluated in the value 5.

$(x := a)s$ *terminates on* $s[x := A(a)S]$

$(skip)s$ *terminates on* s

$$\frac{(S_1)s \text{ terminates on } s' \qquad B(b)s = true}{(if\ b\ then\ S_1\ else\ S_2)s \text{ terminates on } s'}$$

$$\frac{(S_2)s \text{ terminates on } s' \qquad B(b)s = false}{(if\ b\ then\ S_1\ else\ S_2)s \text{ terminates on } s'}$$

$$\frac{(S_1)s \text{ terminates on } s' \qquad (S_2)s' \text{ terminates on } s''}{(S_1; S_2)s \text{ terminates on } s''}$$

$$\frac{\exists j, 0 \leq j\ B(b)s_j = false \qquad \forall k, 0 \leq k < j\ B(b)s_k = true \qquad (S)s_i \text{ terminates on } s_{i+1}, 0 \leq i < j}{(while\ b\ do\ S)s \text{ terminates on } s_j}$$

Table 1: Semantics of Termination

$y := 1$
$x := 5$
while $x > 0$ *do*
$\quad y := y * x$
$\quad x := x - 1$

the initial state s *is the empty function.*
$(y := 1)s$ *terminates in* $s[y := 1]$.
$(x := 5)s[y := 1]$ *terminates in* $s_0 = s[y := 1][x := 5]$
$((y := y * x); (x := x - 1))s_i$ *terminates on* s_{i+1}
From $s_0(x) = 5$ *and* $s_{i+1}(x) = s_i(x) - 1$ *follows* $s_5(x) = 0$
and then $B(x > 0)s_5 = false$.

3.2 Logic for non-termination

Let S stand for a program. We define the relationship S in s state s does not terminate. Table 2 presents the semantics of non-termination of statements.

Theorem 3.4. *Equivalence I*
An statement **S** *on a state* **s** loops *according to the natural semantics as defined in [2] if and only* $(S)s$ *non-terminates.* **proof** *straightforward.*

Theorem 3.5. *Equivalence II*
An statement **S** *on a state* **s** loops *according to the structural operational semantics as defined in [2] if and only if* $(S)s$ *non-terminates.* **proof** *straightforward.*

Example 3.6. *A trivial looping program*

while true do skip

$$\frac{(S_1)s\ non-terminates \qquad B(b)s = true}{(if\ b\ then\ S_1\ else\ S_2)s\ non-terminates}$$

$$\frac{(S_2)s\ non-terminates \qquad B(b)s = false}{(if\ b\ then\ S_1\ else\ S_2)s\ non-terminates}$$

$$\frac{(S_1)s\ non-terminates}{(S_1;S_2)s\ non-terminates}$$

$$\frac{(S_1)s\ terminates\ on\ s' \qquad (S_2)s'\ non-terminates}{(S_1;S_2)s\ non-terminates}$$

$$\frac{\forall k, 0 \leq k\ (S)s_k\ terminates\ on\ s_{k+1} \forall j, 0 \leq j\ B(b)s_j = true}{(while\ b\ do\ S)s\ non-terminates}$$

$$\frac{\exists i, 0 \leq i\ (S)s_i\ non-terminates \quad \forall k, 0 \leq k \leq i\ B(b)s_k = true \quad \forall k, 0 \leq k < i\ (S)s_k\ terminates\ on\ s_{k+1}}{(while\ b\ do\ S)s\ non-terminates}$$

Table 2: Semantics of Non-termination

$\forall j, 0 \leq j\ (skip)s = s\ and\ B(b)s_0 = true$
then $\forall j, 0 \leq j\ B(b)s_j = true$ *and the program does not terminate.*

Example 3.7. *A wrong factorial program*
We prove the non-termination of a wrong factorial program whose loop-condition diverges.

$y := 1$
$x := 5$
while $x \leq 5$ *do*
    $y := y * x$
    $x := x - 1$

Let s_0 *be* $s[y := 1][x := 5]$.
We prove by induction that $B(x \leq 5)s_i = true$ *for all* $0 \leq i$.

base) $B(x \leq 5)s_0 = true$
inductive step) assume $B(x \leq 5)s_i = true$ *or what is equivalent* $s_i(x) \leq 5$.
Since $s_{i+1}(x) = s_i(x) - 1$ *then* $s_{i+1}(x) < s_i(x)$
then $s_{i+1}(x) \leq 5$ *and* $B(x \leq 5)s_{i+1} = true$

Example 3.8. *A third example*
We prove the non-termination of a program that includes the wrong factorial program in an inner loop.

$x := 5$
while $x > 0$ *do*
    $y := 1$
    while $x \leq 5$ *do*
        $y := y * x$

$$x := x - 1$$

the initial state s is the empty function.
$(x := 5)s$ terminates in $s_1 = s[x := 5]$.
$B(x > 0)s_1 = true$
If we call S the inner loop then $(S)s_1$ non-terminates.

4 Rules for functions

Any procedure can be written as a function that returns an arbitrary value, so we restrict ourselves to the treatment of functions.

We define a category of functions declarations called D, a new syntactic category of statements S', a new syntactic category of expressions a' and some aditional syntactic categories as $listofvars$ for the parameters in the declaration of a function and $listofvalues$ for the instantiation of the variables in a function call.

Definition 4.1. *Abstract syntax for functions*

D ::= **function** $name\ (listofvars)\ S'$

S' ::= $x\ :=\ a \mid$ **skip** $\mid S'_1; S'_2 \mid$ **if** b **then** S'_1 **else** $S'_2 \mid$ **while** b **do** $S' \mid return(a')$

a' ::= $name(listofvalues) \mid a'_1 + a'_2 \mid a'_1 * a'_2 \mid a'_1 a'_2$

$listofvars$ ::= **var** $x; listofvars \mid x; listofvars \mid \epsilon$

$listofvalues\ :=\ a'; listofvalues \mid \epsilon$

ϵ stands for the empty list and name for a string of letters and numbers.
The meta-variables in the primed categories are primed and can be subscripted for example $a', a'_1, a'_2, \ldots$.

The arguments of the functions definition as well as the value returned are integers.

Definition 4.2. *Environments for functions declaration*

We define Env in **Names** $\rightarrow S' \times listofvars$ *and we store in it the declarations of functions. We define the semantic function Decl to process declarations of functions:*

$Decl($**function** $name\ (listofvars)\{S'\})Env =$
$\quad Env[name := (S', listofvars)]$

Definition 4.3. *Semantics of return*

We extend our definition of termination. *In case a statement terminates we return also a value, that is the value of the arithmetic expression in the case of the statement return. For the definition of termination of the other statements (given before) we put by convention the value* 0. *We pass also as parameter the environment.*

$$\frac{A'(a')s\ e\ terminates\ on\ n}{(return\ a')s\ e\ terminates\ on\ (s, n)}$$

4.1 Semantics of termination of function calls

The evaluation of arithmetic expressions can finish or not depending on the termination of function calls. We define the function A' to give the semantics of the extended category of arithmetic expressions.

$A' : a' \times State \times Env \to Z \times State$

$$\frac{(S')changestate(l, listofvalues, s)\ terminates\ on\ (s', n) \quad e(name) = (S', l)}{A'(name(listofvalues))s\ e\ = (n, modifybyref(l, s, s'))}$$

$$\frac{A'(a'_1)s\ e\ = (n_1, s_1) \quad A'(a'_2)s_1\ e\ = (n_2, s_2)}{A'(a'_1 + a'_2)s\ e\ = (n_1 + n_2, s_2)}$$

$$\frac{A'(a'_1)s\ e\ = (n_1, s_1) \quad A'(a'_2)s_1\ e\ terminates\ on\ (n_2, s_2)}{A'(a'_1 * a'_2)s\ e\ = (n_1 * n_2, s_2)}$$

$$\frac{A'(a'_1)s\ e\ = (n_1, s_1) \quad A'(a'_2)s_1\ e\ terminates\ on\ (n_2, s_2)}{A'(a'_1 - a'_2)s\ e\ = (n_1 - n_2, s_2)}$$

$changestate(\epsilon, \epsilon, s) = s$
$changestate((\mathbf{var}\ x : l_1), (a' : l2), s) = changestate(l_1, l_2, s[x := A'(a')s])$
$changestate((x : l_1), (a : l2), s) = changestate(l_1, l_2, s[x := A'(a')s])$

$modifybyref(\epsilon, s, s') = s$
$modifybyref((\mathbf{var}x : l), s, s') = modifybyref(l, s[x := s'(x)], s')$
$modifybyref((x : l), s, s') = modifybyref(l, s, s')$

The function $modifybyref$ above changes the values of variables passed by reference (declared with **var**).

Definition 4.4. *Statements with function calls and return*
We have to define again termination for assignation because the expression can be a function call.

$$\frac{A'(a')s\ e\ = (n, s')}{(x := a')s\ e\ = (0, s'[x := n])}$$

In what refers to composition, since a statement return finishes the execution of the function, we redefine its semantics.

$(return\ a; S_2)s\ e\ terminates\ on\ (s, A'(a)s\ e)$

$$\frac{(S'_1)s\ e\ terminates\ on\ (s_1, n_1) \quad (S'_2)s_1\ e\ terminates\ on\ (s_2, n_2)}{(S'_1; S'_2)s\ e\ terminates\ on\ (s_2, n_2)}$$

Example 4.5. *The Fibonacci Function*

function $Fib(n)$
$\{if\ n = 0\ then\ return\ 1$
$\ else\ if\ n = 1\ then\ return\ 1$
$\quad else\ return(Fib(n-1) + Fib(n-2))\}$

The call $Fib(0)$ *terminates if terminates the body of the function in an state* s *in which* n *is bound to* 0. *The body of the function terminates because terminates* $(return\ 1)$ *in* $(s, 1)$. *So* $Fib(0)$ *terminates in* 1. *Following the same reasoning,* $Fib(1)$ *terminates in* 1.

Suppose we want to evaluate $Fib(2)$. *The body of the function terminates if terminates* $return(Fib(1)+Fib(0))$. *The evaluation of the expression (Fib(1)+Fib(0)) terminates on* 2 *by the corresponding rule of the function* A'.

5 Further Work

We have presented rules for termination and non-termination for a subset of the **C***-language. Remains to study non-termination of* recursive functions.

We find also interesting to study termination and non-termination of the statements **break**, **continue** *and* **goto**.

We can go further studying automatics termination analysis for imperative languages.

References

[1] Matthew Hennesy. The Semantics of Programming Languages, An Elementary Introduction using Structural Operational Semantics, *John Wiley & Sons, Chichester-New York-Brisbane-Toronto-Singapore, 1990.*

[2] H.R.Nielson and F.Nielson. Semantics with Applications, A Formal Introduction for Computer Science, *John Wiley & Sons, 1992.*

[3] Glynn Winskel. The Formal Semantics of Programming Languages. An Introduction., *The MIT Press, 1993.*

[4] Annalisa Bossi, Sandro Etalle, Sabina Rossi, Jan-Georg Smaus. Semantics and termination of simply-moded logic programs with dynamic scheduling. *Available via CoRR:http://arXiv.org/archive/cs/intro.html, 2001.*

[5] Fred Mesnard, Salvatore Ruggieri. On Proving Left Termination of Constraint Logic Programs. *ACM Transactions on Computational Logic, Vol 4, Issue2, APril 2003, pages 207-259.*

[6] Flemmming Nielson, Hanne Riis Nielson. Termination analysis based on Operational Semantics. *Technical Report, Aarhus University, Denmark 1995.*

[7] Jürgen Brauburger, Jürgen Giesl. Termination Analysis by Inductive Evaluation. *Proceedings of the 15th International Conference on Automated Deductions (CADE-15), Lindau, Germany, LNAI 1421, 1998.*

[8] J.W.Coleman, C.B.Jones. A structural proof of the soundness of rely/guarantee rules. *University of Newcastle upon Tyne. Technical report series, No CS-TR-987, October 2006.*

[9] Christoph Walther and Stephan Schweitzer. Automated Termination Analysis for Incompletely Defined Programs. *F. Baader and A. Voronkov (Eds.):LPAR 2004, LNAI 3452, pp. 332-346, 2005.*

[10] Dino Pedreschi, Salvatore Ruggieri. Classes of Terminating Logic Programs. *Theory and Practice of Logic Programming, 2(3):369-418, 2002.*