Journal of Network and Computer Applications 59 (2016) 109–116

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Termination analysis with recursive calling graphs

Teng Long [a,b,*], Wenhui Zhang [b]

[a] School of Information Engineering, China University of Geosciences (Beijing), 29#, Xueyuan Road, Haidian District, Beijing, PR China
[b] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, 4#, South Fourth Street of Zhongguancun, Haidian District, Beijing, PR China

ARTICLE INFO

Available online 16 July 2015

Keywords:
Termination analysis
Green software programs
Size-change termination principle

ABSTRACT

As one of the significant aspects for green software systems, termination analysis is related to the optimization of the resource utilization. The approach for size-change termination principle was first proposed by Lee, Jones and Ben-Amram in 2001, which is an effective method for automatic termination analysis. According to its abstracted constructs (size-change graphs), the principle ignores the condition and return values for function call. In this paper, we devise a new construct including the ignoring features to extend the set of programs that are size-change terminating in real life. The main contribution of our paper is twofold: firstly, it supports the analysis of functions in which the returned values are relevant to termination. Secondly, it gains more accuracy for oscillating value change in termination analysis.

1. Introduction

With the development of information and communication technologies, it gives rise to the increasing popularity of electronic devices, raising serious environment concerns for the society. Trying to address these issues, there is a critical need to consider greenness in the whole life cycle of software systems.

A guarantee of right behaviors of software systems is significant, which can lighten greatly network loads and avoid wasting resources. Termination property is one of the properties that describe the right behaviors of the software system. Moreover, the termination analysis (Kuwahara et al., 2014; Heizmann et al., 2010; Codish et al., 2010, 2012; Farzan et al., 2015) is a much-studied topic. In real-life software systems, methods to derive termination properties from recursive and mutually recursive functions are useful in program analysis.

The size-change termination (SCT) principle which was presented in Lee et al. (2001) is simple but surprisingly rich enough to capture the progress of many real-life programs. Size-change graphs (SCG) are essential auxiliary constructs to support SCT. More importantly, the relations between input and output variables of a function call are restricted to the forms as $x > y'$ or $x \geq y'$.[1] It operates over the variables whose " size" is well-founded based on the function call's structure in the program. Once a combination of the relations has been found, termination follows if one variable at least is guaranteed to decrease. The termination analysis is unrelated to the order of the variables which is decided in SCT. Therefore, it is one of the approaches which are successfully applied in a large class of programs for termination analysis. There are some attempts (Ben-Amram and Genaim, 2014, 2013; Ben-Amram, 2009; Cook et al., 2013) to relax the well-founded restriction. However, the downside is that the SCG can only express the transitions involving decreases in well-founded partial orders. In fact, there are still many situations with oscillating value change, in which the traditional SCT approach is not applicable.

We present a technique for deriving program termination properties from size-change information, by constructing a recursive callings graph (RCG, defined in Section 3) and a set of extended size-change graphs (ESCGs, defined in Section 3) for the program. The former one describes the relation to function and function return based on locations in the program. The latter ones approximate the oscillating size change at each location in the program. The strong connected components in the RCG are an approximation of all sequences of function calls that could be idempotent.[2] Our paper presents an important technique which could help stramline the application of the green software programs. The main contribution of our paper with recursive calling graphs is twofold: Firstly, it supports the analysis that can handle recursive and mutually recursive functions in which return values are relevant to termination. Secondly, the approach extends the set of programs that are size-change terminating.

The paper is organized as follows: in Section 2, we introduce the syntax of the language used in this paper and the definitions that are related to size-change termination principle. We propose the definitions

* Corresponding author.
E-mail addresses: longteng@cugb.edu.cn (T. Long), zwh@ios.ac.cn (W. Zhang).

[1] The names for input and output variables could be different.

[2] Idempotent graphs describe the self-recursive function calls.

http://dx.doi.org/10.1016/j.jnca.2015.06.019
1084-8045/

of auxiliary constructs and do termination analysis in Section 3. Section 3 outlines how to record the returned values, and how to do a more precise analysis. An algorithm is proposed in Section 4. Section 5 illustrates the application of the approach on an example with self-recursive function callings. Related work is discussed in Section 6. We end with concluding remarks in Section 7.

2. Preliminaries

In this section, we will list the syntax of the language used in this paper and the definitions in size-change termination principle in Lee et al. (2001).

2.1. Language

The language used in this paper is a simple first-order call-by-value functional language, defined in Table 1. A variable can represent any expression. Similarly, a constant can represent any expression. An expression can be a conditional choice based on the equational theory (if..then..else). Functions are used to represent definitions about these variables. So does the expressions.

2.2. Size-change termination principle

Definition 1 (***Size-change graph (SCG)***). Suppose functions f and g are defined in program P. A size-change graph $G: f \to g$ for P is a set of labeled arcs $x \overset{>}{\to} y$ or $x \overset{\geq}{\to} y$ where $x \in Variables(f)$, $y \in Variables(g)$.

Functions f and g (the caller and callee) are respectively called the source and the target of G. G is an abstraction of a call transition – from f to g. The arcs in the graph are as abstract transitions.

Definition 2 (***Closure***). The closure of a set of size-change graphs is the smallest set $cl(g)$ such that

- $g \subseteq cl(g)$
- If $G_1: f \to f'$ and $G_2: f' \to f''$ are in $cl(g)$, then $G_1; G_2 \in cl(g)$. $G_1; G_2: f \to f''$ with arc set E defined below: $E = \{x \overset{>}{\to} z \mid \exists y, w. x \overset{>}{\to} y \overset{w}{\to} z$ or $x \overset{w}{\to} y \overset{>}{\to} z, w \in \{>, \geq\}\} \cup \{x \overset{\geq}{\to} z \mid \exists y. x \overset{\geq}{\to} y \overset{\geq}{\to} z)\}$ where $x \overset{w}{\to} y$ and $y \overset{w'}{\to} z$ are respectively arcs of G_1 and G_2.

Size-change graph G is idempotent if $G; G = G$. A self-recursive function call can be expressed by an idempotent size-change graph.

Theorem 1 (*Size-change termination principle*). *Program P is SCT terminating iff every idempotent G in cl(g) has an arc* $z \overset{>}{\to} z$.

The SCT is described in Lee et al. (2001) as follows: *A program terminates on all inputs if every infinite call sequence (following program control flow) would cause an infinite descent in some data values.*

Table 1
The language syntax.

x	$\in$	Var	(Variables)
op	$\in$	Prim	(Primitive operators)
f	$\in$	FName	(Function names)
c	$\in$	Const	(Constants)
e	$\in$	Exp	(Expressions)
		e:: if e_0 then e_1 else e_2	
		$\mid x \mid c \mid e_1$ op e_2	
		$\mid f(e_1, \ldots, e_n)$	
d	$\in$	Def	(Definitions)
		d:: $f(x_1, \ldots, x_n) = e$	

A closure set of size-change graphs (sets of abstract transitions) is used to express the set of all the possible call sequences. All of the idempotent ones in closure set stand for the infinite call sequences. If a descent arc "$>$" can be found, it satisfies the "infinite descent" in the principle.

3. Termination analysis

The termination analysis for programs with nested self-recursive functions over integer domain is complex, because of the complicated "size change" situation. The possible cases are as follows:

1. The value of variables in the program declines all the time.
2. The value of variables in the program increases in each function call.
3. The value of variables in the program is oscillating, such as in the McCarthy's 91 function.

There are only two arc forms in size-change graphs, such as $x > y'$ or $x \geq y'$, therefore, it can deal with the first case. In our work, additional variables and a new auxiliary construct are used to address the latter two cases.

3.1. Additional variables

In this work, additional variables consist of counter variables and returned variables. The former one is used to record the number of times for some function call/return, while the latter one is used to record the returned value.

The returned values are always ignored in available approaches, so functions in which the returned values are relevant to termination cannot be analyzed correctly. Adding extra variables in the program is the first step to extend the range of programs for termination analysis.

A set of additional variables $\mathcal{V}_{add}$ consists of a set of returned variables and a set of counter variables. After augmenting the program with additional variables, the set of variables $\mathcal{V}$ in the program will augmented as $\mathcal{V} \cup \mathcal{V}_{add}$.

3.2. Definitions of auxiliary constructs

We construct the recursive calling graph (RCG) to describe the control flow for the augmented program which can express not only the function calls but the function returns. The transitions in a RCG are described as ESCGs which are extended by transforming the arc types from $x > y'$ or $x \geq y'$ to $x \overset{\delta}{\to} y'$ where $\delta \in \mathbb{Z}$ is the changed value in the transition.

The changes of values in ESCGs have to be satisfied in size reducing. In other words, the corresponding code cannot be revisited, and the recursive calling graph cannot result in any infinite cycle.

3.2.1. Extended size-change graph

Definition 3 (***Extended size-change graph (ESCG)***). Suppose functions f and g are defined in program P. Location a is the place where the value of variables function g changed in program P. An extended size-change graph $B_a: f \to g$ for P is a set of labeled arcs $x \overset{\delta}{\to} y$ where $x \in Variables(f)$, $y \in Variables(g), \delta \in \mathbb{Z}$ is the changed value for function calls.

Suppose functions f, g are defined in the program. An ESCG $B_a \in \mathcal{B}$, $B_a: f \to g$ for the program is a set of labeled arcs $x \overset{\delta}{\to} y$ where $\delta \in \mathbb{Z}, x \in Variables(f), y \in Variables(g)$. Functions f and g are respectively called the source and the target of $B(source(B) = f, target(B) = g)$.