Science of Computer Programming 111 (2015) 483–504

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A practical comparator of cost functions and its applications[☆]

Elvira Albert[a,*], Puri Arenas[a], Samir Genaim[a], Germán Puebla[b]

[a] DSIC, Complutense University of Madrid (UCM), Spain
[b] DLSIIS, Technical University of Madrid (UPM), Spain

ARTICLE INFO

Article history:
Received 16 September 2013
Received in revised form 5 December 2014
Accepted 8 December 2014
Available online 16 December 2014

Keywords:
Resource analysis
Cost analysis
Function comparison
Upper/lower bounds

ABSTRACT

Automatic cost analysis has significantly advanced in the last few years. Nowadays, a number of cost analyzers exist which automatically produce upper- and/or lower-bounds on the amount of resources required to execute a program. Cost analysis has a number of important applications such as resource-usage verification and program synthesis and optimization. For such applications to be successful, it is not sufficient to have automatic cost analysis. It is also required to have automated means for handling the analysis results, which are in the form of *Cost Functions* (*CFs* for short) i.e., non-recursive expressions composed of a relatively small number of types of basic expressions. In particular, we need automated means for *comparing CFs* in order to prove that a *CF* is smaller than or equal to another one for all input values of interest. General function comparison is a hard mathematical problem. Rather than attacking the general problem, in this work we focus on comparing *CFs* by exploiting their syntactic properties and we present, to the best of our knowledge, the first practical *CF* comparator which opens the door to fully automated applications of cost analysis. We have implemented the comparator and made its source code available *online*, so that any cost analyzer can use it.

1. Introduction

Cost analysis [28,12], a.k.a. resource analysis, aims at statically predicting the resource consumption of programs in terms of their input data sizes. Given a program, cost analysis produces a *Cost Function* (*CF* for short) which may correspond to an upper-bound or a lower-bound, depending on the kind of analysis performed. For instance, upper bounds are required to ensure that a program can run within the resources available; lower bounds are useful for scheduling distributed computations. Starting from the seminal cost analysis framework by Wegbreit [28], cost analyzers are often generic on the notion of *cost model*, e.g., they can be used to measure different resources, such as the number of instructions executed, the amount of memory allocated, the number of calls to a certain method, etc. Thus, *CFs* can be used to represent the usage of any of such resources.

In all applications of resource analysis, such as resource-usage verification, program synthesis and optimization, etc., it is necessary to compare *CFs*. This allows choosing an implementation with smaller cost or guaranteeing that the given resource-usage bounds are preserved. Essentially, given a program m, a set of linear constraints φ which impose size re-

[☆] This work was funded partially by the EU project FP7-ICT-610582 ENVISAGE: Engineering Virtualized Services (http://www.envisage-project.eu), by the Spanish projects TIN2008-05624 and TIN2012-38137, and by the CM project S2013/ICE-3006.

[*] Corresponding author.
E-mail addresses: elvira@sip.ucm.es (E. Albert), puri@sip.ucm.es (P. Arenas), samir.genaim@fdi.ucm.es (S. Genaim), german@fi.upm.es (G. Puebla).

http://dx.doi.org/10.1016/j.scico.2014.12.001
0167-6423/© 2014 Elsevier B.V. All rights reserved.

strictions on the input values to m (e.g., that an argument is larger than a certain value or that the size of an array is non-zero), and a *CF* f_m^φ, we aim at comparing it with another *CF* bound $\mathtt{b}$. Depending on the application, such functions can be automatically inferred by a resource analyzer (e.g., if we want to compare the efficiency of two implementations) or one of them can be user-defined (e.g., in resource usage verification one tries to verify, i.e., prove or disprove, *assertions* written by the user about the efficiency of the program).

From a mathematical perspective, the problem of cost function comparison is analogous to the problem of proving that the difference of both functions is a positive function, e.g., $\mathtt{b} - f_m^\varphi \geq 0$ in the context φ. This is in general undecidable[1] and also non-trivial, as *CFs* involve non-linear subexpressions (e.g., exponential, polynomial and logarithmic subexpressions).

1.1. *Summary of contributions*

As our first main contribution, we present a practical approach to the comparison of cost functions. We take advantage of the form that cost functions originating from the analysis of programs have and of the fact that they evaluate to non-negative values. Essentially, our technique consists of the following steps, which constitute our main technical contributions:

1. Normalizing cost functions to a form which makes them amenable to be syntactically compared. This step includes handling operators like max and min (used to express the maximum and minimum of a set of expressions), and transforming arithmetic expressions into sums of products of basic cost expressions.
2. Defining a series of comparison rules for basic cost expressions and their (approximated) differences, which then allow us to compare two products.
3. Providing sufficient conditions for comparing two sums of products by relying on the product comparison, and enhancing it with a *composite* comparison schema which establishes when a product is larger than a sum of products.

The second main contribution is an implementation of the cost comparator that we have made available online at costa.ls.fi.upm.es/comparator to the resource analysis community and which is free software under the *General Public License* (GPL). We define there the syntax of the cost functions used in the implementation, and provide specifications of its interface functions, so that our comparator can be easily integrated in any resource analyzer.

A preliminary version of this work was presented at FOPARA'09 [3]. This article improves Ref. [3] in several aspects: (1) it notably improves the formalization of the comparison process, (2) it formally proves the correctness of the approach, (3) it extends the method to also handle lower bounds, (4) we present applications (including new ones) of the comparator, and (5) finally we provide a new implementation of our approach.

1.2. *Organization of the article*

The rest of the paper is organized as follows. The next section introduces some background in cost analysis and cost functions and presents the syntax of the cost expressions (*CEs* for short) which we handle in this paper. Section 3 presents the problem of comparing two *CFs* in a context provided by means of constraints. In order to come up with practical ways to solve the problem, we propose means for handling the nat-, max- and min-operators and transform the comparison problem to that of comparing a series of expressions which no longer contain such operators. In Section 4, we introduce a novel approach to proving that a *CF* is smaller than another one. Our approach is based on a series of syntactic schemes which provide sufficient conditions to syntactically detect that a given expression is smaller than or equal to another one. The comparison is presented as a fixed point transformation in which we remove from *CEs* those subexpressions for which the comparison has already been proven until the left hand side expression becomes zero. Section 5 discusses several applications of our *CFs* comparator, namely its direct use to check the efficiency improvement of program optimizations, and for program verification and certification. Interestingly, in cases in which an upper bound cannot be found (for instance because the analyzer does not find an upper bound on the number of loop iterations), our comparator can be used to check that the resource consumption is below a given threshold. An overview of other approaches and related work is presented in Section 6 and some conclusions are presented in Section 7.

2. Background on cost analysis

This section introduces some background material on cost analysis and presents the syntax of the *CEs* studied in the paper. We start by introducing some notation. The sets of natural, integer and non-zero natural values are denoted by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{N}^+$, respectively. We write x, y, and z, to denote variables which range over $\mathbb{Z}$. The notation $\bar{t}$ stands for a sequence $t_1, \ldots, t_n$, for some $n > 0$. A *linear expression* over a sequence of variables $\bar{x}$ is of the form $v_0 + v_1x_1 + \ldots + v_nx_n$, where $v_i \in \mathbb{Z}$, $0 \leq i \leq n$ and $x_i \in \bar{x}$ for all $1 \leq i \leq n$. Similarly, a *linear constraint* (over $\mathbb{Z}$) has the form $l_1 \leq l_2$, where l_1 and l_2 are

[1] Since variables range over the integers, undecidability follows from the undecidability of that of Hilbert's 10th problem: given a multivariate polynomial $p(\bar{x})$, decide whether $p(\bar{x}) = 0$ has an integer solution. This problem can be reduced to checking whether $p(\bar{x})^2$ is positive, which is an instance of comparing cost functions.