Overapproximating the Cost of Loops

Master's Thesis description, November 2012

Daniel Schweizer, daschwei@student.ethz.ch

Supervisors: Dr. Pietro Ferrara, Prof. Dr. Peter Müller

INTRODUCTION

TouchDevelop [1] is a novel programming language developed by Microsoft to write scripts on mobile devices (in particular the Windows Phone [2]). This language is particularly simple, since it is aimed at developing applications on mobile devices with limited screen size and input devices. TouchDevelop scripts are created by the users on their smartphones and executed within the TouchDevelop run time environment on the phone. These scripts can be shared with other users by uploading them to the TouchDevelop cloud infrastructure. Because of the limited hardware resources of mobile devices, a cost analysis of a TouchDevelop script could provide some useful information to optimize its execution. In particular, we want to attach cost information to the script and use this information at runtime to decide whether the application should be executed locally on the mobile device, or on the cloud.

THE TOUCHDEVELOP LANGUAGE

The TouchDevelop language is a typed, structured programming language, built around the idea of only using a smartphone's touchscreen as input device when writing code [3]. The language mixes imperative, object-oriented, and functional features. It uses object-orientation only in a limited sense: the API offers a number of predefined classes (representing basic data types and data structures, different kinds of media, and interfaces for the physical components of the device), but the definition of new (own) classes is not possible. As a structured programming language, TouchDevelop offers sequential composition, if-then-else statements and three forms of loops:

1. `while` *condition* `do`
2. `for each` *x* `in` *collection* `where` *condition* `do`
   which iterates over all elements of a collection.
3. `for` $0 \leq i < expr$ `do`
   which uses an index variable *i* incremented from zero to *expr* - 1. The index variable is read-only, i.e., its value is increased by 1 after each loop iteration and cannot be changed in any other way.

SAMPLE

During the last three years, the Chair of Programming Methodology developed the static analyzer Sample (*Static Analyzer of Multiple Programming LanguagEs*) [4, 5] based on the abstract interpretation theory [6, 7]. This analyzer has been already applied to a wide range of properties (for instance, types, string values, integers, heap structures) and to different programming languages (Scala and Java bytecode). Therefore, it is flexible and generic enough to be applied to the cost analysis of TouchDevelop programs. The representation of a program in Sample relies on control flow graphs (CFG), and at this level the information about the specific type of loop (namely, `while`, `for`, and `foreach`) represented by the CFG is already abstracted away.

GOAL

The main goal of this master thesis is to develop a cost analysis that correctly overapproximates the cost of a loop. We define the cost of a loop as the number of times the loop is iterated. Since the representation of a program adopted by Sample is based on control flow graphs, we cannot distinguish between `while`, `for`, and `foreach` loops, and we have to treat them in a uniform way. We split the development of this master thesis in two steps:

1. The first step will be to precisely overapproximate the cost of loops obtained from `for` and `foreach` statements. We expect that the treatment of these loops will be easier than the ones obtained from `while` loops, since they iterate in a more predictable way. In particular, we will study the patterns of the CFG structure and of the statements contained in the loop body to identify the parts of the CFG obtained by the compilation of `for` and `foreach` statements. In this way, we hope to be able to precisely compute the costs of these parts of code.
2. The second step of the thesis will be the extension of this approach to loops obtained from arbitrary `while` statements. We expect that we will be able to precisely abstract the cost of only some of these loops, while in the other cases the analysis will not yield a precise result. This cannot be avoided since we cannot precisely compute the cost of every loop statically.

The whole approach will be implemented in Sample. This will then allow us to study the results of the proposed analysis in practice.
Note that this project is focused on loops. Therefore, the treatment of other statements is outside the scope of this master thesis, and it will be studied separately. In the context of this thesis, we will assume to know the cost of the body of a loop, focusing the work only on the statements that affect the number of iterations of the loop.

RELATED WORK

Developing a precise overapproximation of the cost of a program is a complex problem that has been (partially) explored in the last few years.
[8] presents the first approach to the automatic cost analysis of object-oriented bytecode programs. This method takes a bytecode program and a cost model specifying the resource of interest as input, and returns a set of recursive equations, which capture the execution cost of the program. In a first step, this approach generates an intermediate *rule-based representation (RBR)* from the original bytecode. Then, they use static analysis to infer linear *size relations* among program variables at different program points. As a size abstraction for integer variables they use the value of the variable, whereas the size abstraction of a data structure $x \in dom(lv)$ (where *lv* denotes some variable mapping) with respect to some heap is defined as the length of the maximal path reachable from the reference *lv(x)* by dereferencing, i.e., following other references as fields. The path-length of `null` is defined to be 0, that of a cyclic data structure is defined to be ∞. The next step consists of finding an appropriate cost model, which defines how cost is assigned to each execution step (for example, a cost model counts the number of instructions or the amount of memory consumption). Then, this method generates a *cost relation system* (*CRS*) from the RBR, the size relations and the cost model.
Finally, this method uses a solver for recursive systems in order to try to find an exact solution or an upper bound in non-recursive form for the CRS. This last step is described in more detail in [9].

When approximating the cost of a program, it is crucial to find an appropriate abstraction of the heap. A *maximal path-length* abstraction as used in [8] might yield good results in some cases, but in other cases it might be imprecise. For example, consider a cyclic data structure, which per definition has a size (i.e. maximal path-length) of ∞. However, it might be the case that in every execution of a given program, this data structure is traversed in a controlled way such that the actual maximal length of any path ever taken is some finite number *n*. In such a case, using the *maximal path-length* abstraction would yield a rather imprecise overapproximation of the cost of the program. So it will be a central point of this master thesis to find a better heap abstraction.

REFERENCES

[1] TouchDevelop. URL: http://research.microsoft.com/en-us/projects/touchdevelop/.

[2] Windows Phone. URL: http://www.microsoft.com/windowsphone/.

[3] N. Horspool, J. Bishop, A. Samuel, N. Tillmann, M. Moskal, J. de Halleux and M. Fähndrich. TouchDevelop – Programming on a Phone. Version 1.1 for TouchDevelop 2.8, May 2012.

[4] P. Ferrara. Static Type Analysis of Pattern Matching by Abstract Interpretation. In *Proceedings of FMOODS '10, pp. 186–200*, 2010.

[5] P. Ferrara, R. Fuchs and U. Juhasz. TVAL+: TVLA and Value Analyses Together. In *Proceedings of SEFM '12, pp. 63-77*, 2012.

[6] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of POPL '77*. ACM Press, 1977.

[7] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of POPL '79*. ACM Press, 1979.

[8] E. Albert, P. Arenas, S. Genaim, G. Puebla and D. Zanardini. Cost Analysis of Object-Oriented Bytecode Programs. In *Journal of Theoretical Computer Science, 413 (1), pp.142-159*, 2012.

[9] E. Albert, P. Arenas, S. Genaim, G. Puebla. Closed-Form Upper Bounds in Static Cost Analysis. In *Journal of Automated Reasoning, 46 (2), pp. 161–203*, 2011.