Bachelor Project Synopsis

Sound and complete termination analysis of higher-order programs

Oleksandr Shturmov
University of Copenhagen
DIKU

September 30^{th}, 2011

1 Problem statement

What are some of the presently known methods of performing termination analysis? Can they be applied to higher-order programs? Are they fully automatic, i.e. do not require human intervention? Are they sound? Are they complete? What are the limitations of these methods and are they efficient?

The preliminary focus is on the size-change termination principle, which states that a program terminates if every infinite control flow sequence infinitely descends a well-founded data value, that is, a data value that bottoms out[1].

2 Motivation

While the halting problem is unsolvable in general, it remains an active area of research to what extent automated termination analysis can be performed in computing environments constrained in various ways.

Interesting results have been achieved by constraining one-self to first-order programs, well-founded data types and axiomatic program construction. Some have extended these ideas to higher-order programs. Some methods are sound, but most are incomplete, as e.g. the size-change termination principle[2]. Termination in and of itself is interesting for various reasons:

- Proving total correctness of programs demands a proof of the program termination aside from the proof of the program adhering to a given specification. Total correctness is thus more interesting than partial correctness(where the termination proof is left out), since a program that does not terminate may conceptually adhere to any specification.[3].
- Some program manipulation systems, e.g. wrt. operating systems, would often rather avoid making use of programs that do not terminate[4].
- Automated complexity analysis can't be performed in a sound and complete manner we can't decide the halting property of the program[5].

It is interesting to see if the restrictions imposed by some of the presently known methods for automatic termination analysis are permissive enough to allow for some of these more practical problems to be solved.

3 Learning Objectives

- Document the general limitations imposed by the halting problem.

- Present the state of automated termination analysis in some constrained computing environments.
- Present relevant scientific material of both classical and modern nature.
- Devise a simple language and describe some of the key concepts from the articles above in terms of it.
- Determine the general usefulness of such methods, i.e. what are their limitations and are they efficient.

4 Tasks & schedule

The general outline of the project is to first get better acquainted with the limitations imposed by the halting problem in general, followed by an analysis of one or more ideas which have been successfully applied to perform automated termination analysis in variously constrained computing environments, with the preliminary focus on portraying these ideas in a higher-order programming context.

1. **Read and analyze classical literature on the general undecidability of the halting problem and the consequences thereof.**

   Period: week 40/2011

2. **While week number** < 50 **repeat steps 3 to 5.**

   Steps 3 to 5 present an expected work-load in terms of weeks. This work-load can vary depending on the complexity of the articles chosen in step 3. As already mentioned, the preliminary focus will be on the size-change termination principle, and this will hence be the first area of research, if time allows for it, other methods will be explored as well.

   Period: week 41-50/2011

3. **Find a suitable amount of relevant scientific material and perform a preliminary analysis thereof.**

   This task embodies finding a set of relevant articles, of both classical and derived nature. To find a *suitable* amount of scientific material it is both important to estimate the relevance of the considered articles as well as their complexity given the time constraints. Such estimations cannot be derived from reading the titles alone, nor are they seldom clear from the articles' respective abstracts. It is therefore expected that the preliminary analysis goes in lock-step with finding the material.

   Expected work-load: 2 weeks

4. **Pick out the key concepts from the articles above and devise a simple language to which these concepts can be applied without loss of generality.**

   This step involves reading through the previously chosen set of articles and picking out the key ideas. Some articles may prove to be of a lesser importance, or of greater complexity than originally estimated, and a brief detour to task 3 may be necessary.

   The purpose of the language was already described in the learning objectives. In addition to that, it should be as simple and as small as possible. There is seemingly no interest in devising any sort of interpreter for the language, however this should be trivial if the language indeed is as simple and small as intended. If it should prove too complicated to devise a single language for all the key concepts, a set of simple dialects is preferred over one more complicated language.

   Expected work-load: 2 weeks

5. **Describe each key concept in terms of the language.**

   This is expected to be the most time-consuming and challenging part of the project as it involves rereading the articles and analysing them in enough detail to be able to reconvey the ideas in own terms. The intent here is furthermore not to recap the articles iteratively and in full detail, but rather to recap the key ideas by combining both classical and derived articles to describe the chosen key concepts.

   Expected work-load: 2-4 weeks

6. **Proof reading, overall project revision and conclusion.**

   At this stage it should be possible to reason about the methods covered in the previous sections and conclude of what use those methods are in their respective computing environments, i.e. are they sound, are they complete, are they efficient, etc.

   Period: week 52/2011 - 01/2012

5 Documentation

It is expected that project documentation in the form of a report is handed in by noon, January 10^{th} 2012.

Documentation is expected to be written in lockstep with the various tasks outlined above, and therefore demands no task in and of itself. For example, given the theoretic nature of the project a substantial amount of "reporting" is expected for the step where the key concepts are to be described in terms of the devised language.

References

[1] Chin Soon Lee, Neil D. Jones, Amir M. Ben-Amram, *The Size-Change Principle for Program Termination*. Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL '01, ISBN 1-58113-336-7, pages 81-92, London, United Kingdom, ACM, id 360210.

[2] Neil D. Jones and Nina Bohr, *Termination Analysis of Untyped λ-Calculus*, RTA 2004, LNCS 3091, pages 1-23, Springer-Verlag, Berlin Heidelberg, 2004.

[3] Zohar Manna and Amir Pnueli, *Axiomatic Approach to Total Correctness of Programs*, Acta Informatica Volume 3, Number 3, pages 243-263, 1974.

[4] Byron Cook, Andreas Podelski and Andrey Rybalchenko, *Termination Proofs for Systems Code*. Prooceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation, PLDI '06, ISBN 1-59593-320-4, pages 415-426, Ottawa, Ontario, Canada, ACM, id 1134029.

[5] Mads Rosendahl, *Automatic Complexity Analysis*. Proceedings of the fourth international conference on Functional programming languages, FPCA' 89, ISBN 0-89791-328-0, Imperial College, London, United Kingdom, pages 144-156, AMC, id 99381.