Research Statement

Abu Naser Masud
`http://costa.ls.fi.upm.es/~anm/`

My principal research area is on analysis, specification, and verification of software systems. Software development has reached an era where the demand for obtaining reliable, correct and efficient software has increased significantly. Motivated by this ever increasing demand of formally verifying correct software code, my primary research goals have been directed towards analyzing and understanding software systems. More particularly, I am interested both in analyzing source code statically and analyzing an abstract model of the system which captures some properties of the original program to be verified.

Automatic program analysis and verification are challenging due to complex program semantics, difficulty in obtaining appropriate formal model of the given program, limited scalability of the existing mathematical theories, precision-applicability-scalability trade-off and so on. With this motivation and challenges in mind, I started working on static program analysis for inferring complexity properties of the given program. Automatic inference of computational complexity properties is of the utmost importance in understanding the efficiency (e.g. the input program has polynomial execution behavior) or reliability (e.g. the program will not consume more than 1000K memory) of program code.

In the following, I present some of my contributions in more detail and outline briefly about my future research plans.

Automatic Complexity Analysis

The aim of automatic complexity analysis is to estimate the bound of resource consumption (aka cost) of executing a given program as a function of its input data sizes. Existing complexity analysis frameworks (e.g. ACE [9], CiaoPP [8], Costa [3] etc.) infer complexity bounds that are either not precise or the analysis approach is not widely applicable. In my approach [4], I have developed techniques based on symbolic computation and static analysis that infer precise bounds and are widely applicable. Due to the difficulty in underapproximating execution cost, the framework for the estimation of lower bound resource consumption either does not exist (e.g. Costa) or exists with a very limited applicability (e.g. CiaoPP). The main advantage of my approach is that it can be applied in a dual way to infer very precise and nontrivial lower bounds. The approach is completely automatic and the efficiency does not depend on the structural complexity of the inferred closed-form symbolic formulas.

Termination Analysis

There are theoretical interests in understanding the degree of solvability of inferring resource bounds for some class of programs. In my approach, cost analysis requires solving the termination problem of simple integer loops for bounding the loop iterations or recursion depths. Hence, study of the theoretical limits for termination analysis, which is important for the termination analysis itself, can be inherited to the limits of cost analysis. In the termination analysis [5] of simple integer loops, I have shown that it is undecidable when the body of the loop is expressed by a set of linear inequalities where an arbitrary irrational is allowed as a coefficient; when the loop is a sequence of instructions, that compute either linear expressions or the step function; or when the loop body is a piecewise linear deterministic update with two pieces. For integer constraint loops with rational coefficients only, I have shown that a Petri net can be simulated with such a loop which implies some interesting lower bounds.

Future Research Plan

Abstract Program Model. The classical approach of static complexity analysis first infers an abstract program model from the input program. This abstract model captures the cost of the input program while leaving all the details of programming language specific features. Abstract program models vary according to the degree of expressiveness. Less expressive models are easy to analyze with existing tools and techniques but possibly do not capture important program properties. My future research plan includes understanding the theoretical limits on the expressiveness of various formal program models with respect to obtaining different qualitative or quantitative program properties. For example, given an abstract program model, I am interested to know weather it is possible to infer amortized complexity of the corresponding program; or weather the given model allows inferring costs of the corresponding program which allows nested function calls or have nonlinear input-output size relations etc.

Cost Analysis for Concurrent Programs. Cost analysis of thread-based concurrent programs are notoriously difficult due to a large number of thread interleaving which makes the static inference of shared variables value at particular program point much harder. Cost analysis requires inferring invariant relations among program variables or data structures. Due to the difficulty in approximating shared variables value, often the input-output size relations are not precise enough to capture relevant costs of the given thread. Existing verification or cost analysis approaches restrict the concurrency model by restricting the number of context-switches [1], number of running threads [6], concurrency primitives [2] and so on. My future plan is to investigate about cost analysis framework for different concurrency models.

In order to obtain cost analysis of thread based concurrent programs, we need to obtain the notion of costs that are relevant for concurrent programs. The next step would be to infer the input-output size relation which is the most tricky part of the analysis. Inferring this input-output size relation requires inferring the invariants (both global invariants on shared variables and local invariants on thread local variables) at different program points. There are some works [1, 7] on inferring invariants for thread-based concurrent programs but are restricted in the sense that either they work on programs where the number of context-switches are limited or the inferred invariants are not able to capture sophisticated relationships among variables suitable for cost analysis. Sometimes, even if the relational invariants are possible to infer, they are sufficient only to prove program assertions in most cases. For cost analysis, we need to have sophisticated invariants that capture both relational invariants (e.g. $x \geq x' + y$) and non-relational invariants (e.g. $x \geq c$) at the same time, and may need to infer the progress on the values of the shared variables until a particular program point. Also, sharing or alias analysis will be required together with the program invariants which in turn will help obtaining loop bounds for threads in the concurrent setting.

Termination Analysis. As I have mentioned before, termination analysis is important from the point of view of program verification as well as understanding theoretical and practical limits of cost analysis. In my previous research [5], I have left some open questions regarding the termination property of some simple integer loops. I would like to investigate the decidability property of simple loops where the loop conditions are affine linear inequality constraints, loop updates are affine linear updates and all program variables are over the integers. It has been conjectured that termination of this loop is decidable. I am interested in investigating its decidability on termination by analyzing the property of complex numbers in the context of linear algebra. This is because if we represent the update of the program variables using matrix notation, the property of this matrix after some iterations depend on eigenvalues and eigenvectors (elements of which are possibly in the complex domain) of that matrix. Then in the integer domain, complex eigenvalues represent repetitions on the values of program variables inside the loop and hence a possible non-termination. I am also interested in investigating the decidability property of the previous loops where all loop updates are affine linear inequality constraints, as this is important in the context of program analysis where all updates are abstracted by linear inequalities and ask for the termination of those loops.

So far, I have presented some future research plans on cost and termination analysis. In general, I would like to explore techniques on inferring quantitative program properties which can be either accumulative or non-accumulative. Finally, I would also like to extend my research area in the algorithmic verification of programs deployed in different computational environments.

References

[1] E. Albert, P. Arenas, S. Genaim, M. Gómez-Zamalloa, and G. Puebla. Cost analysis of concurrent oo programs. In H. Yang, editor, *APLAS*, volume 7078 of *Lecture Notes in Computer Science*, pages 238–254. Springer, 2011.

[2] E. Albert, P. Arenas, S. Genaim, M. Gómez-Zamalloa, and G. Puebla. COSTABS: A Cost and Termination Analyzer for ABS. In *Proceedings of the 2012 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2012, Philadelphia, Pennsylvania, USA, January 23-24, 2012*, pages 151–154. ACM Press, January 2012.

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

[4] E. Albert, S. Genaim, and A. N. Masud. On the inference of resource usage upper and lower bounds. *ACM Transactions on Computational Logic*, 2013. To appear.

[5] A. M. Ben-Amram, S. Genaim, and A. N. Masud. On the termination of integer loops. *ACM Trans. Program. Lang. Syst.*, 34(4):17, 2012.

[6] B. Cook, A. Podelski, and A. Rybalchenko. Proving thread termination. In J. Ferrante and K. S. McKinley, editors, *PLDI*, pages 320–330. ACM, 2007.

[7] J. Field and M. Hicks, editors. *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*. ACM, 2012.

[8] J. A. Navas, E. Mera, P. López-García, and M. V. Hermenegildo. User-definable resource bounds analysis for logic programs. In V. Dahl and I. Niemelä, editors, *ICLP*, volume 4670 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2007.

[9] M. Rosendahl. Automatic complexity analysis. In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, FPCA '89, pages 144–156, New York, NY, USA, 1989. ACM.