Limits of Optimal Parallel Flow Analysis [Abstract]

Markus Müller-Olm

June 12, 2001

Static program slicing [13] is an established program reduction technique that has applications in program understanding, debugging, and testing [12]. It has also been proposed as a technique for ameliorating the state-explosion problem when formally verifying software or hardware [3, 2, 1, 6]. The goal of program slicing is to identify and remove parts of the program that cannot (potentially) influence given value(s) at given program point(s) of interest.

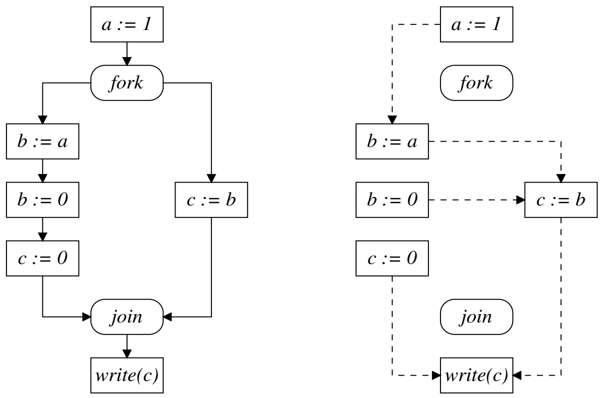
In recent years, the standard slicing technique in sequential program which is based on a backwards reachability analysis in the program dependency graph (PDG) has been extended to certain classes of parallel languages in order to allow slicing of languages like JAVA or VHDL. The resulting slicing algorithms are sound – they never remove statements that can influence the slicing criterion – but unlike the sequential algorithms they are not complete[1]. Consider, e.g., the program

$$a := 1; [(b := 1; b := 0; c := 0) \parallel c := b] \,; \mathbf{write}(c) \,,$$

for which in Fig. 1 (a) a control flow graph-like representation is shown and in (b) the data and interference dependences. We are interested in slicing w.r.t. variable c at the write instruction. Clearly, the instruction $a := 1$ is backwards reachable in the dependency graph from the write-instruction. But it is not hard to see that there is no execution of the program that realizes all dependences in this path and therefore an optimal slicer must remove $a := 1$.

Is just the extension of the sequential techniques too naive or are there fundamental limits? In my talk which was based on [8, 7], I presented new complexity and undecidability results showing the latter to be the case. Specifically, the interprocedural parallel slicing problem is undecidable and the intraprocedural problem is PSPACE-complete. Even if we restrict attention to parallel programs without loops and procedures the problem remains NP-complete. Similar results can be shown for other data-flow problems like detection of copy constants, and detection of truly-life variables in parallel programs. This comes as a surprise, as only slightly simpler analysis questions, namely, *strong*

[1]We call a slicer *complete* if it yields statement-minimal slices after abstraction of conditional branching to non-deterministic branching, an abstraction that is quite common in data-flow analysis in order to ban undecidability.



(a) CFG-like representation (b) Data and interference dependences

Figure 1: An example program.

copy constant propagation and (ordinary) liveness of variables can be optimally solved in polynomial time [5, 4, 10].

Previous complexity and undecidability results for dataflow problems for concurrent languages [11, 9] exploit in an essential way synchronization primitives of the considered languages. In contrast our results hold independently of any synchronization and exploit only interleaving of atomic assignment statements. Therefore, they apply to a much wider class of concurrent languages.

References

[1] E. M. Clarke, M. Fujita, S. P. Rajan, T. Reps, S. Shankar, and T. Teitelbaum. Program slicing for VHDL. In *Correct hardware design and verification methods (Proceedings Charme'99)*, Bad Herrenalb, Germany, September 1999.

[2] J. Hatcliff, J. Corbett, M. Dwyer, S. Sokolowski, and H. Zheng. A formal study of slicing for multi-threaded programs with JVM concurrency primitives. In A. Cortesi and G. Filé, editors, *Static Analysis – SAS'99*, LNCS 1694, pages 1–18. Springer-Verlag, 1999.

[3] M. Iwaihara, M. Nomura, S. Ichinose, and H. Yasuura. Program slicing on VHDL descriptions and its applications. In *Proc. 3rd Asian Pacific Conference Hardware Description Languages (APCHDL'96)*, pages 132–139, Bangalore, January 1996.

[4] J. Knoop. Parallel constant propagation. In D. Pritchard and J. Reeve, editors, *4th European Conference on Parallel Processing (Euro-Par)*, LNCS 1470, pages 445–455. Springer-Verlag, 1998.

[5] J. Knoop, B. Steffen, and J. Vollmer. Parallelism for free: Efficient and optimal bitvector analyses for parallel programs. *ACM Transactions on Programming Languages and Systems*, 18(3):268–299, 1996.

[6] L. I. Millett and T. Teitelbaum. Issues in slicing PROMELA and its applications to model checking, protocol understanding, and simulation. *International Journal on Software Tools for Technology Transfer*, 2(4):343–349, 2000.

[7] M. Müller-Olm. The complexity of copy constant detection in parallel programs. In A. Ferreira and H. Reichel, editors, *STACS 2001 (18th Annual Symposium on Theoretical Aspects of Computer Science)*, LNCS 2010, pages 490–501. Springer, 2001.

[8] M. Müller-Olm and H. Seidl. On optimal slicing of parallel programs. In *STOC 2001 (33th Annual Symposium on Theory of Computing)*, to appear.

[9] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM TOPLAS*, 22(2): 416–430, 2000.

[10] H. Seidl and B. Steffen. Constraint-based inter-procedural analysis of parallel programs. In G. Smolka, editor, *Programming Languages and Systems (ESOP'2000)*, LNCS 1782, pages 351–365. Springer-Verlag, 2000.

[11] R. N. Taylor. Complexity of analyzing the synchronization structure of concurrent programs. *Acta Informatica*, 19:57–84, 1983.

[12] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, September 1995.

[13] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.