Model Construction for Finite-state Verification :

Application of Abstract Interpretation and Partial Evaluation Techniques

Quarterly Report
NAG 21209*

David Schmidt Matthew Dwyer John Hatcliff

November 2, 1998

The primary goal of the project is to develop tools for compiling software system source code to finite-state models suitable for model-checking. The tools are founded upon abstract interpretation and partial evaluation technology. The sections below summarize progress on project foundations, applications, and tool-building.

Foundations

Because the project makes extensive application of novel technology from abstract interpretation and partial evaluation and integrates it with model checking, foundational work plays a nontrivial role in the research.

Schmidt's reformulation of Cousot-Cousot-style abstract interpretation into operational semantics format recently appeared in *Journal of Lisp and Symbolic Computation* [8]; these results provide the formal justification for the application of abstract interpretation into the project. For example, they are used to justify what we call *filter formulas*: LTL formulas that encode a form of assume/guarantee reasoning which can be used in component-based verification [4]. Schmidt has also started a research collaboration with Bernhard Steffen of Dortmund University, the leading figure in the integration of abstract-interpretation technology into model checking. Their initial work has produced an expository paper [7], which integrates the two authors' perspectives. The material was presented as an invited talk at the 1998 Static Analysis Symposium at Pisa.

Hatcliff *et al.* combined aspects of abstract interpretation and partial evaluation to obtain *abstraction-based program specialization* [5]. Dwyer and Hatcliff showed how slicing techniques can be used to optimize specialization and to guide users in choosing appropriate abstractions for model creation [1].

This work provides a rigorous semantic foundation for our approach to model creation.

Applications

Hatcliff *et al.* used the abstraction-based specialization techniques discussed above to create models for various Ada concurrent configurable systems used in scientific computing applications [6]. Given the results of pre-processing the software using partial evaluation, Dwyer and Pasareanu applied the work on filter formulas above to obtain component-based model-checking of the concurrent Ada components [2]. In addition, Dwyer and Pasareanu showed how filter formulas can be use to verify properties of generic container implementations of structures such as lists, queues, *etc.*[3].

This work illustrates the feasibility of partial evaluation and abstract interpretation model creation techniques on real-world software systems.

*Kansas State University. ({`schmidt, dwyer, hatcliff`}`@cis.ksu.edu`)

Tools

We originally proposed to build a set of model creation tools for processing Ada source code. Ada was chosen because we intended to reuse Ada tools developed by researchers at the University of Massachusetts with whom we are collaborating. After assessing the relative interest in Ada versus Java, and with encouragement from directors of the NSF Experimental Software Systems program who are funding the work at Massachusetts, we and the University of Massachusetts researchers decided to retarget our tool sets to Java.

Our strategy has been to apply the foundational principles discussed above to build a complete pipeline of tools that convert Java source to inputs of several different model checking systems (Spin, SMV, INCA, FLAVERS). The pipeline has the following components.

1. **Slicing:** A slicer takes in (a) a specification to be verified and (b) the Java source. It yields a residual program where the program components unrelated to the properties mentioned in the specification have been removed.

2. **Abstraction-based partial evaluation:** An abstraction-based partial evaluator takes in (a) the sliced program and (b) an abstraction definition for each class used in the Java program. It yields an abstract version of the input program.

3. **Translation to guarded command intermediate language:** The abstracted program is converted to a guarded command intermediate language. The goal here is to move to a transition system framework without committing to a particular model checker input language.

4. **Translation to input languages of moded checking tools:** The guarded transition program is translated to the input language of several model checking systems including Spin, SMV, INCA, and FLAVERS. We hope to eventually target the SAL intermediate language being developed at SRI and Stanford.

Currently, only a small core of Java is treated. We are in the process of extending the tools to treat method calls based on a representation for control flow graphs developed by Banerjee and Schmidt. We will expand the tools incrementally to incorporate more language features.

References

[1] Matthew Dwyer and John Hatcliff. Slicing software for model construction. Technical Report 98-14, Kansas State University, Department of Computing and Information Sciences, 1998. (submitted for publication).

[2] Matthew B. Dwyer and Corina S. Pasareanu. Filter-based model checking of partial systems. In *Proceedings of the Sixth ACM SIGSOFT Symposium on Foundations of Software Engineering*, November 1998. to appear.

[3] Matthew B. Dwyer and Corina S. Pasareanu. Model checking generic container implementations. Technical Report 98-10, Kansas State University, Department of Computing and Informatio n Sciences, 1998. in review.

[4] M.B. Dwyer and D.A. Schmidt. Limiting state explosion with filter-based refinement. In *Proceedings of the 1st International Workshop on Verification, Ab stract Interpretation and Model Checking*, October 1997.

[5] John Hatcliff, Matthew B. Dwyer, and Shawn Laubach. Staging static analysis using abstraction-based program specialization. In *LNCS 1490.* Principles of Declarative Programming 10th International Symposium, PLILP'98, September 1998.

[6] John Hatcliff, Matthew B. Dwyer, Shawn Laubach, and Nanda Muhammad. Specializing configurable systems for finite-state verification. Technical Report 98-4, Kansas State University, Department of Computing and Information Sciences, 1998.

[7] D.A. Schmidt and B. Steffen. Data-flow analysis as model checking of abstract interpretations. In *Proc. 5th Static Analysis Symposium*. Springer LNCS, 1998.

[8] David A. Schmidt. Trace-based abstract interpretation of operational semantics. *Lisp and Symbolic Computation*, 10:237–271, 1998.