REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT ~~DATE TYPE~~	3. DATES COVERED (From - To)
09-09-2002	Final Technical	Nov 1998-Sept 2002

4. TITLE AND SUBTITLE

Transformational Development of Reactive Systems.

5a. CONTRACT NUMBER

N00014-99-1-0131

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

BAA 98-019

6. AUTHOR(S)

Zuck, Lenore
Pnueli, Amir
Goldberg, Benjamin

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NYU, Courant Institute of Math. Sciences
251 Mercer Street
New York, NY 10012

8. PERFORMING ORGANIZATION REPORT NUMBER

F0931

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

ONR, Mathematics, Computer, & Info. Sciences
Division ONR 311
800 N. Quincy Street
Arlington, VA 22217-5660

10. SPONSOR/MONITOR'S ACRONYM(S)

ONR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

N00014-99-1-0131

12. DISTRIBUTION AVAILABILITY STATEMENT

~~NA~~ Approved for public release.

13. SUPPLEMENTARY NOTES

NA

14. ABSTRACT

This is the final technical report for the project. It presents an enumeration of the research results of the effort, including a list of papers published and a brief summary of the content of the papers. The results were primarily in the area of compiler validation, formal methods, and verification. Applications of the work include reactive systems, as mentioned in the title of the effort, as well as more general software systems.

15. SUBJECT TERMS

Compiler validation, formal methods, verification

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		6	Benjamin Goldberg
					19b. TELEPONE NUMBER (Include area code) (212) 998-3495

20020916 084

Transformational Development of Reactive Systems

Final Technical Report

Lenore Zuck, Amir Pnueli, and Benjamin Goldberg
Department of Computer Science
Courant Institute of Mathematial Sciences
New York University
{zuck,amir,goldberg}@cs.nyu.edu

1 Introduction

This report provides an enumeration of the technical contributions made by Professors Amir Pnueli, Lenore Zuck, and Benjamin Goldberg with the support of the ONR grant. This ONR grant was originally awarded to Professor Robert Paige, to perform work in the area of transformational programming of reactive systems. However, due to the passing of Professor Paige, and the assumption of the grant by Professors Pnueli, Zuck, and Goldberg, there was a shift in the focus of the project to better correspond to the expertise of the participants. A more apt title of the project became "The Development and Application of Formal Method Techniques for Reactive Systems". The work spans new methods for verifying software, the theory and implementation of compiler validation, and related techniques. For each paper published with the support of the ONR grant, below is the listing of where the paper appeared, along with a brief description of the work described in the paper

2 Research Results

2.1 Compiler Validation

L. Zuck, A. Pnueli, Y. Fang, and B. Goldberg, "VOC: A Translation Validator for Optimizing Compilers". In *Proceedings of the workshop on Compiler Optimization Meets Compiler Verificaiton (COCV) 2002*, ENTCS 65(2). April 2002. Also submitted for consideration to Journal of Universal Computer Science (J. UCS). June 2002.

There is a growing awareness, both in industry and academia, of the crucial role of formally proving the correctness of safety-critical components of systems. Most formal verification methods verify the correctness of a high-level representation of the system against a given specification. However, if one wishes to infer from such a verification the correctness of the code which runs on the actual target architecture, it is essential to prove that the high-level representation is correctly implemented at the lower level. That is, it is essential to verify the the correctness of the translation from the high-level source-code representation to the object code, a translation which is typically performed by a compiler (or a code generator in case the source is a specification rather than a programming language).

Formally verifying a full-fledged optimizing compiler, as one would verify any other large program, is not feasible due to its size, ongoing evolution and modification, and, possibly, proprietary considerations. The *translation validation* method used in this research is a novel approach that offers an alternative to the verification of translators in general and compilers in particular. According to the translation validation approach, rather than verifying the compiler itself, one constructs a validation tool which, after every run of the compiler, formally confirms that the target code produced on that run is a correct translation of the source program.

We have developed a methodology *VOC* for the translation validation of optimizing compilers. We distinguish between *structure preserving* optimizations, for which we establish a simulation relation between the source and target code based on computational induction, and *structure modifying* optimizations, for which we develop specialized "meta-rules". We have alos implemented em VOCS64--a prototype translation validator that automatically produces verification conditions for the global optimizations of the SGI Pro-64 compiler.

L. Zuck, A. Pnueli, Y. Fang, B. Goldberg and Y. Hu, "Translation and Run-Time Validation of Optimized code", In *Proceedings of the Workshop on Runtime Verification (RV) 2002*, ENTCS 70(4). July 2002.

In additionto expanding the work on compiler validation, this paper described our work work on *run-time validation of speculative loop optimizations*, which involves using run-time tests to ensure the correctness of loop optimizations which neither the compiler nor compiler-validation techniques can guarantee the correctness of. Unlike compiler validation, run-time validation has not only the task of determining when an optimization has generated incorrect code, but also has the task of recovering from the optimization without aborting the program or producing an incorrect result. This technique has been applied to several loop optimizations, including loop interchange, loop tiling, and software pipelining and appears to be quite promising.

2.2 Formal Methods and Verification

D. Peled, A. Pnueli, and L. Zuck, "From falsification to verification,", *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS)*, Springer Verlag LNCS 2245, December 2001, pages 292–304.

In this paper, we described an improvement to the linear temporal logic model checking process, enhancing ability to automatically generate a deductive proof that the system meets its temporal specification. We emphasized the point of view that model checking can also be used to *justify* why the system actually works. We showed that, by exploiting the information in the graph that is generated during the search for counterexamples, when the search of counterexamples fails, we can generate a fully deductive proof that the system meets its specification.

D. Peled and L. Zuck, "From model checking to a temporal proof," in *Proceedings of the 8^{t}h International SPIN Workshop on Model Checking of Software*, Springer Verlag LNCS 2057, May 2001, pages 1–14.

Model checking is used to automatically verify temporal properties of finite state systems. It is usually considered to be 'successful', when an error, in the form of a counterexample to the checked property, is found. In this paper, presented the dual approach, where, in the absence of a counterexample, we automatically generate a proof that the checked property is satisfied by the given system. Such a proof can be used to obtain intuition about the verified system. This approach can be added as a simple extension to existing model checking tools.

A. Pnueli, S. Ruah, and L. Zuck, "Automatic : deductive verification with invisible invariants," in *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*. Lecture Notes in Computer Science 2031 Springer, April 2001, pages 82–97.

The paper presented a method for the automatic verification of a certain class of parameterized systems. These are bounded-data systems consisting of *Nprocesses*(N being the parameter), where each process is finite-state. First, we showed that if we use the standard deductive INV rule for proving invariance properties, then all the generated verification conditions can be automatically resolved by finite-state (BDD-based)

methods with no need for interactive theorem proving. Next, we showed how to use model-checking techniques over finite (and small) instances of the parameterized system in order to derive candidates for invariant assertions. Combining this automatic computation of invariants with the previously mentioned resolution of the VCs (verification conditions) yields a (necessarily) incomplete but fully automatic sound method for verifying bounded-data parameterized systems. The generated invariants can be transferred to the VC-validation phase without ever been examined by the user, which explains why we refer to them as "invisible".

We illustrated the method on a non-trivial example of a cache protocol, provided by Steve German of IBM's T.J. Watson Research Center.

T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck, "Parameterized verification with automatically computed inductive assertions," in *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*, Springer LNCS 2102, July 2001, pages 221-234.

The paper presented a method, called the method of verification by invisible invariants, for the automatic verification of a large class of parameterized systems. The method is based on the automatic calculation of candidate inductive assertions and checking for their inductiveness, using symbolic model-checking techniques for both tasks. First, we showed how to use model-checking techniques over finite (and small) instances of the parameterized system in order to derive candidates for invariant assertions. Next, we showed that the premises of the standard deductive INV rule for proving invariance properties can be automatically resolved by finite-state (BDD-based) methods with no need for interactive theorem proving. Combining the automatic computation of invariants with the automatic resolution of the VCs (verification conditions) yields a (necessarily) incomplete but fully automatic sound method for verifying large classes of parameterized systems. The generated invariants can be transferred to the VC-validation phase without ever been examined by the user, which explains why we refer to them as "invisible". The efficacy of the method is demonstrated by automatic verification of diverse parameterized systems in a fully automatic and efficient manner.

Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck, "Network Invariants in Action". To appear in CONCUR 2002.

The paper presented the method of *network invariants* for verifying a wide spectrum of properties, including liveness, of parameterized systems. This method can be applied to establish the validity of the property over a system $S(n)$ for every value of the parameter n. The application of the method requires checking abstraction relations

between two finite-state systems. We presented a proof rule, based on the method of Abstraction Mapping by Abadi and Lamport, which has been implemented on the TLV model checker and incorporates both history and prophecy variables. The effectiveness of the network invariant method is illustrated on several examples, including a deterministic and probabilistic versions of the dining-philosophers problem and an algorithm for distributed termination detection.

A. Pnueli, J. Xu, and L. Zuck, "The $(0,1,\infty)$-Counter Abstraction", *Proceedings of the 14^{th} Conference on Computer Aided Verification (CAV'02)*, Springer Verlag LNCS 2404, July 2002, pages 107–122.

In this paper, we introduced the $(0,1,\infty)$-*counter abstraction* method by which a parameterized system of unbounded size is abstracted into a finite-state system. Assuming that each process in the parameterized system is finite-state, the abstract variables are limited counters which count, for each local state s of a process, the number of processes which currently are in local state s. The counters are saturated at 2. The emphasis of the paper was on the derivation of an adequate and sound set of fairness requirements (both weak and strong) that enable proofs of liveness properties of the abstract system, from which we can safely conclude a corresponding liveness property of the original parameterized system. We illustrated the method on few parameterized systems, including Szymanski's Algorithm for mutual exclusion. The method was also extended to deal with parameterized systems whose processes may have infinitely many local states, such as the Bakery Algorithm, by choosing few "interesting" state assertions and $(0,1,\infty)$-counting the number of processes satisfying them.

L. Zuck, A. Pnueli, and Y. Kesten, "Automatic Verification of Probabilistic Free Choice", in *Proceedsings of the 3^{rd} International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI) 2002*, Venice, January 2002. Springer Verlag LNCS volume 2294.

In this paper, we described an automatic method for establishing P-validity (validity with probability 1) of simple temporal properties over finite-state probabilistic systems. The new approach replaced P-validity with validity over a non-probabilistic version of the system, in which probabilistic choices are replaced by non-deterministic choices constrained by compassion (strong fairness) requirements. "Simple" properties are temporal properties whose only temporal operators are (eventually) and its dual (always). In general, the appropriate compassion requirements are "global," since they involve global states of the system. Yet, in many cases they can be transformed into "local" requirements, which enables their verification by model checkers. We demonstrated

our methodology of translating the problem of P-validity into that of verification of a system with local compassion requirement on the "courteous philosophers" algorithm of [LR81], a parameterized probabilistic system that is notoriously difficult to verify, and outlined a verification of the algorithm that was obtained by the TLV model checker.

T. Arons, A. Pnueli, and L. Zuck, "Verification by Probabilistic Abstraction", Submitted for consideration to POPL'03.

This paper described automatic verification of liveness properties with probability 1, over parameterized programs that include probabilistic transitions. The paper proposed a two novel approaches to the problem: The first uses the measure theoretic notion of validity with probability 1, and allows for a *Planner* that occasionally determines the outcome of a finite sequence of "random" choices, while the other random choices are performed non-deterministically; in fact, they can be determined by an adversary. Using a Planner, a probabilistic protocol can be treated just like a non-probabilistic one and verified as such. The second approach is based on a notion of fairness that is sound and complete for verifying simple temporal properties over finite-state systems. The paper presented a symbolic model checker based on such fairness. It also shows how the network invariant approach accommodate probabilistic protocols.