Benchmarking Finite-State Verifiers

George S. Avrunin* James C. Corbett† Matthew B. Dwyer‡

Abstract

A variety of largely automated methods have been proposed for finite-state verification of software systems. Although anecdotal accounts of success are widely reported, there is very little empirical data on the relative strengths and weaknesses of those methods across a broad range of analysis questions and systems. But this information is critical for the transfer of the technology from research to practice. We review some of the problems involved in obtaining this information and suggest several ways in which the community can facilitate empirical evaluation of finite-state verification tools.

1 Introduction

Finite-state verifiers, such as SPIN [5] and SMV [6], are tools that deduce properties of finite-state models of computer systems. They can be used to check properties such as freedom from deadlock, mutually exclusive use of resources, and eventual response to a request. A wide variety of analysis methods have been automated in such tools and applied to a range of computer systems, both software and hardware. Such tools can automate large parts of the model construction and essentially all of the analysis. This high degree of automation, and the robustness of some of the tools, would seem to make finite-state verification tools prime candidates for transfer to software development practice.

The most common analysis method used to detect faults in software systems is, of course, testing, in which the system is executed with particular inputs and the results checked for correctness. The problem with testing is coverage: it is usually infeasible to check more than a very small fraction of the possible executions of the system, and testing can give no information about those executions that are not examined. For a concurrent system, moreover, testing is especially problematic because the nondeterministic behavior typical of such systems means that two executions with the same input data may be quite different. Finite-state verification tools can consider all possible executions of a system, sequential or concurrent, and can be applied at any stage of development at which an appropriate model can be constructed.

*Laboratory for Advanced Software Engineering Research, Department of Computer Science, University of Massachusetts at Amherst, Amherst, MA 01003-4610, USA, email: avrunin@math.umass.edu

†Department of Information and Computer Science, University of Hawai‘i, Honolulu, HI 96822, USA, email: corbett@hawaii.edu

‡Department of Computing and Information Sciences, Kansas State University, Manhattan, KS 66506-2302, USA, email: dwyer@cis.ksu.edu

But, despite the fact that finite-state verification tools seem to offer a number of advantages over the current state-of-the-practice, they have not been widely adopted. Certainly there are many reasons for this, ranging from the scalability of the analysis to the ease of use of the tools, but we believe one significant factor is the difficulty of selecting one from the variety of existing methods. In particular, the performance of these methods varies enormously from system to system, yet there is little empirical data evaluating or comparing the effectiveness of the methods on different kinds of systems.

Consider the related field of compiler optimization, which also involves analysis of system descriptions. When a compiler incorporates a new optimization, the effectiveness of that optimization is evaluated empirically by running the compiler on a suite of standard benchmarks. The expected performance gain for a typical program (or perhaps one with certain characteristics) can be estimated from the benchmarks. Most importantly, the benchmarks allow a comparison of different compilers and optimization techniques.

Contrast this to the situation in the field of finite-state verification. A researcher builds a new verifier (or perhaps enhances an existing verifier) with a new analysis method and reports the performance of the verifier on one or two examples on which it performs better than some existing method. Published case studies typically report on the application of a specific tool to a specific example; rarely are multiple verifiers even tried, let alone compared carefully.

As a result, a practitioner seeking to apply finite-state verification might be more confused than enlightened by the anecdotal nature of the current research literature. It is quite natural for a discipline to begin by creating different methods to solve a problem, but as the discipline matures, it must develop the means to compare and evaluate these methods. Although the field of compilers is much more mature than the field of finite-state verification (and arguably addresses an easier problem), we believe that it is time for the finite-state verification community to develop the means to benchmark the tools and techniques we create. The transfer of our technology from research to practice will require at least a rough characterization of the strengths, weaknesses, and capabilities of our methods.

In this piece, we argue for the importance of empirical evaluation in finite-state verification, discuss the barriers to empirical work in this area, and conclude with concrete suggestions on how our community can help foster empirical work.

2 The problem

People familiar with even a single finite-state verification tool will know that the performance of that tool can vary significantly with what seem to be small changes in the system (or model) being analyzed or the property being checked. It is very difficult to predict the performance of the tool from any straightforward measure of the size of the system being analyzed or the property being checked. The situation is even more striking when different tools are compared.

For example, we compared the performance of several finite-state verification tools in checking a number of properties of the Chiron user-interface development system

[4]. We modeled the event dispatch mechanism in Chiron, in which a dispatcher task delivers events to "artist" tasks, which then update the display. We constructed two models: one with a single dispatcher for all events, and one in which there is a separate sub-dispatcher for each event (the set of these sub-dispatchers, when composed, is observationally equivalent to the single dispatcher). Figures 1 and 2 show the time (on a Sun Enterprise 3500 with 2 GB of memory) required by two finite-state verification tools, INCA [3] and SPIN, to check one property of the two versions of Chiron with two artists and an increasing number of events. (The data shown are for checking that the dispatcher never delivers an event to an artist that has not registered for that event.)

The figures show a quite striking change in the performance of the Inequality Necessary Condition Analyzer (INCA), which uses an integer programming-based analysis, and a smaller, but still quite substantial, change in the performance of SPIN, which uses state enumeration. (Note that the vertical axis is a logarithmic scale.) A software developer choosing a tool to analyze a large Chiron interface with the single dispatcher model would certainly prefer to use SPIN, since INCA cannot analyze a system with more than a few events, but INCA performs significantly better than SPIN on the version in which the dispatcher is decomposed. And these extreme differences are observed on two behaviorally equivalent models of the same system.

There has been only a small amount of work comparing different tools across a variety of software systems and properties, but these differences in performance are typical of what has been seen. In fact, the performance of the various finite-state verification tools, as measured by time and memory requirements or the scale of system to which they can be applied, varies sharply in ways that we do not understand very well and cannot predict accurately. Of course, these measures of performance are only some of the criteria that software developers would use in selecting a method to apply to a system under development. Factors involving ease of use, such as the difficulty of specifying properties or building a tractably small model, also vary from one finite-state verification approach to another as the systems and properties vary, and, although harder to measure, may be even more important to software developers.

So we have a variety of finite-state verification tools whose performance and ease of use vary substantially when they are applied to check different properties of different systems, but in ways that we cannot predict very well. We cannot make sound suggestions to software developers about which tool to use in a particular situation. Of course, the effort required to apply one tool to check one property of one real system is substantial, so that trying several tools is unlikely to be feasible. Is it any wonder that software developers have not adopted these techniques?

3 The need for an empirical approach

In order to provide guidance to software developers in choosing and applying finite-state verification tools, researchers must better understand the sources of these variations in performance. Empirical studies of the application of finite-state verification tools to a check a wide range of properties of a wide range of systems are necessary to develop this understanding.

There are at least two reasons why an analytic approach is unlikely to be suffi-

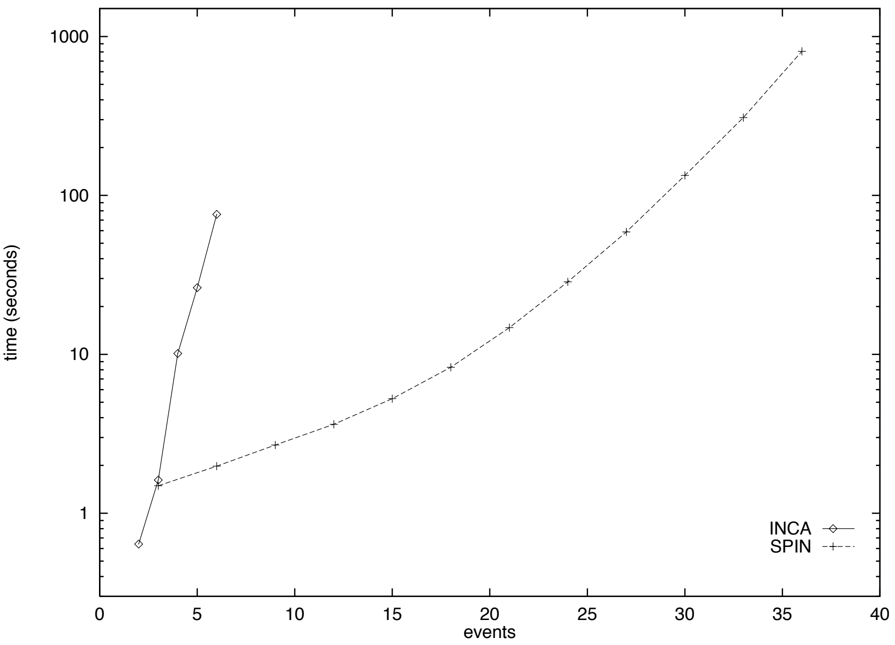

Figure 1: Verification with Single Dispatcher Process

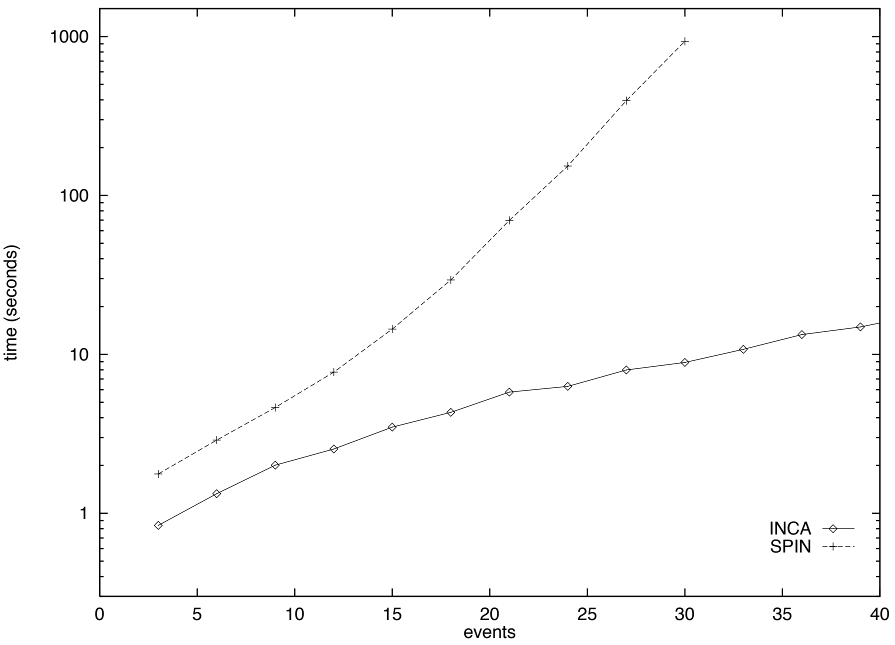

Figure 2: Verification with Multiple Dispatcher Processes

cient. First, the underlying algorithms of many of the leading finite-state verifications approaches are themselves not well understood, in the sense that we cannot accurately predict their performance in particular cases. For instance, we know only a little about the factors that determine the size of the ordered binary decision diagrams (OBDDs) used in SMV and related symbolic model checkers. Similarly, INCA relies on integer programming algorithms whose performance in particular cases is hard to determine a priori.

Second, an evaluation of a method requires some measure of how the method performs on typical systems that arise in practice. For any given method, it is almost always possible to find or construct an example on which that method performs better than other methods (these examples can usually be found in the paper describing the method), but for comparison and evaluation, we would like to know how the method performs on a suite of standard benchmarks that approximates a representative sample of systems.

Of course, analytical results on the complexity of analysis methods on specific kinds of realistic systems would be extremely useful in practice. And we believe a wealth of empirical data on the performance of the methods might inspire such results, which often result from the attempt to explain an observed variation in a specific case. But broad pragmatic comparisons will require an empirical approach.

4 The lack of progress to date

Few empirical studies have attempted to compare the performance of different finite-state verifiers on different example systems, and even these few have been quite limited. The relative scarcity of empirical work in this area (as compared to compilers, for example) results not from laziness on the part of researchers, but from the inherent difficulty of the problem. Past studies [1, 2] have identified many problems in comparing these tools. Among the most significant are:

- Each of the different finite-state verification tools has its own input formalism for describing the system to be analyzed, and there are a number of different formalisms for specifying the property to be checked. These formalisms may provide, for example, different constructs for communication between processes and may have different expressive power. It is therefore difficult to be sure that the tools being compared are actually working on the same problem.

- Effective use of even a single tool on a system of realistic size and complexity requires significant experience with the tool and a substantial investment of effort. A comparison of several different tools on a wide range of different systems is beyond the scope of many research projects.

- There is no standard set of examples to which to apply a method for evaluation/comparison. Collecting a representative sample of systems to analyze is effectively impossible for an individual. Obtaining real systems from industry is complicated by proprietary restrictions.

- We are interested not only in such measures of the performance of tools as the time and memory they use, but also in factors affecting their ease of use by software developers. Experiments involving human subjects, however, are difficult to design correctly and expensive to conduct.

These issues can be addressed in a number of ways. For example, automated translation between models can be helpful in constructing appropriate input for different tools from a single system, although validating such a translation is not straightforward. It is not enough to make sure that the models are as close to semantically equivalent as possible; it must be shown that the translation does not introduce bias against a particular tool or tools. This involves additional effort by individuals who are expert in the use of the particular tools. Similarly, a publicly-available collection of standard benchmark examples would allow different research groups to study the performance of different tools on the same examples, and would make examples available to the entire research community. The open source movement may make it easier to obtain real examples.

5 What the research community should do

We urge finite-state verification researchers to increase their efforts to carry out empirical comparison and evaluation of finite-state verification tools. Specific actions that can help:

- Make tools freely available.

- Publish full examples. If complete examples cannot be made available, perhaps at least "sanitized" versions, such as concurrency skeletons, can be. While space limitations in journals and conferences may prevent the inclusion of full source in papers, it can certainly be provided on the web. We are creating a web repository of examples and encourage others to do likewise.

- Use standard notations, languages, and models whenever possible to facilitate interchange.

- As a community, develop a set (or sets) of realistic benchmark problems, perhaps drawing from common design patterns or software architectures.

- Increase the empirical component of research papers. As standard benchmarks become available, researchers should apply their tools/techniques to these examples so they can be compared. As a reviewer, insist that authors include empirical evaluations and comparisons in their papers.

Most importantly, the empirical comparison and evaluation of finite-state verification tools must be regarded as an important part of the field. It is only through such empirical work that we can understand the strengths and weaknesses of the different approaches, and understanding those strengths and weaknesses is critical for the transfer of these techniques from research to practice.

Acknowledgment

This work was partially supported by the National Science Foundation under grants CCR-9703094 and CCR-9708184 and by NASA under grant NAG-02-1209.

References

[1] A. T. Chamillard, L. A. Clarke, and G. S. Avrunin. An empirical comparison of static concurrency analysis techniques. Technical Report 96-84, Department of Computer Science, University of Massachusetts, 1996. Revised May 1997.

[2] J. C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Trans. Softw. Eng.*, 22(3):161–179, Mar. 1996.

[3] J. C. Corbett and G. S. Avrunin. Using integer programming to verify general safety and liveness properties. *Formal Methods in System Design*, 6:97–123, January 1995.

[4] K. Forester, C. MacFarlane, M. Cameron, and G. Bolcer. Chiron-1 user manual. Arcadia Document UCI-93-07, University of California, Irvine, Sept. 1993.

[5] G. J. Holzmann. The model checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[6] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.