Panel on Design for Verification

Tevfik Bultan
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
bultan@cs.ucsb.edu

Constance Heitmeyer
Naval Research Laboratory
Code 5546
Washington, DC 20375, USA
heitmeyer@itd.nrl.navy.mil

John O'Leary
Intel Corporation
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124, USA
john.w.oleary@intel.com

Abstract

Although research in automated verification has produced very promising results, the question of how to effectively integrate these results into the software and hardware development processes is still unresolved. Typically, fully automated verification techniques are not scalable, and scalable verification techniques require substantial user guidance. Alternatively, developers could facilitate scalable verification by constructing software and hardware systems in ways that make them easier to verify. In this panel we will discuss the idea of design for verification.

1. Introduction

There has been significant progress in automated verification techniques in the last two decades. When combined with increasing computing power, these techniques are capable of analyzing complex hardware and software systems as demonstrated by numerous case studies. However, scalable automated verification still remains a challenge. Most applications of verification succeed either by requiring some manual intervention or by focusing on a specific type of system or a specific type of problem. It is still unclear if there is a general framework for scalable automated verification.

Scalability of automated verification depends on extracting compact models that hide the details that are not relevant to the properties that are being verified. This typically requires a reverse engineering step in which either user guidance or static analysis techniques (or both) are used to rediscover some information about the input system that may be known to its developers at design time. A design for verification approach, which enables hardware and software developers to document the design decisions that can be useful for verification, may improve the scalability and therefore the applicability of automated verification techniques significantly.

Design for verification can be interpreted in different ways. One approach could be to ask the system designers to structure the systems in certain ways in order to make them easier to verify. This interpretation requires identifying design principles for developing verifiable systems. Another approach could be to elicit extra information from the system designers that would be helpful during automated verification. This interpretation requires identifying mechanisms and languages for passing the design information from the designers to the automated verification tools.

Capturing design intent for the purpose of verification is not a new idea. Verification formalisms based on Hoare logic and Dijkstra's weakest preconditions required specification of loop invariants and pre and post-conditions. Work on abstract data types and, more recently, on design by contract in the software domain, and the recent work on assertion based verification in the hardware domain can all be considered approaches to design for verification. In this panel, we will discuss general principles and challenges of design for verification.

2 Verifiable Designs

One way of creating verifiable designs could be to leverage existing successful verification techniques. For exam-

ple, model checking research has been especially successful in automated verification of finite state specifications. Techniques based on symbolic analysis (such as symbolic model checking with BDDs or bounded model checking with SAT solvers) or explicit state enumeration (such as efficient depth first search techniques, partial order reductions) have been successful in scaling model checking to large finite state models. There has been work on verification of finite state models that appear in design specifications, and there has been work on extracting finite state models from implementations. However, there are not many approaches which support finite state models throughout the development life cycle.

- How can finite state models be effectively used at the design stage in order to facilitate verification?

There are several approaches that have been used repeatedly to improve the scalability of automated verification technologies such as modular verification, abstraction and refinement. Modularity, abstraction and iterative refinement are also principles that are commonly used at the design stage.

- How can the modularity, abstraction and refinement at the design stage be better connected to the verification techniques which depend on these principles?

Modular verification techniques require specification (or discovery) of the module interfaces. However, in order to achieve modularity in verification, the module interfaces have to provide just the right amount of information. If the interfaces provide too much information, then they are not helpful in achieving modularity in verification. On the other hand, if they provide too little information, they are not helpful for verifying interesting properties. Hence, successful modular verification partly depends on the successful declaration of the interfaces which is part of the design activity.

- How should developers design interfaces for them to be useful during verification?

Modeling the environment of the system under verification is one of the biggest challenges in software verification. Similar to the interfaces among the modules of a system, a successful environment model is crucial for scalable verification.

- How can designers contribute to development of environment models for verification?

The issues we raised above are mainly concerned with improving the effectiveness of existing verification techniques. A basic question about design for verification is if we need a whole new way of designing systems for correctness, rather than modifications to existing methodologies that help leverage existing verification techniques.

- Is it possible to develop verifiable designs using an evolutionary approach based on existing methodologies and verification techniques or do we need a revolutionary approach?

3 Language Issues

Some of the questions mentioned above are related to languages. For example state of the art programming languages do not provide adequate mechanisms for representing module interfaces because they provide too little information. Think of an object class in an object oriented language. The interface of an object class consists of names and types of its fields, and names, return and argument types of its methods. Such interface specifications do not contain sufficient information for most verification tasks. For example, such an interface does not contain any information about the order the methods of the class should be called. In order to achieve modular verification, module interfaces should be richer than the ones provided by the existing programming languages.

- How can the design and implementation languages be modified to increase the effectiveness of automated verification techniques?

Leveraging existing successful verification techniques at the design stage would require some language support too. For example, there are numerous opportunities for using finite state machines in software design, such as, using finite state machines to model the states of an object as in UML state machines. Language constructs which promote/support finite state models would be helpful for verification.

- What are the language constructs that would be useful for verification?

The question of how to pass the design information that can be useful for verification to the verification tools is another important issue. A successful design for verification approach needs to connect the language used to specify the design to the language of the implementation and to the language of the verification tool. This could be particularly challenging in software development since most software developers do not decide on the exact software behavior until they construct the source code.

- Is it possible to connect the design, implementation and verification models and languages effectively?

4 Verification in Practice

Most of the reported success stories about the application of automated verification techniques involve experts who

have a formal methods background. However, many of the practitioners who design and build hardware and software systems are engineers who may not have any training in formal methods.

- Can design for verification be effective if the designers are not trained in formal methods?

Most developers are greatly concerned with performance. Hence, they design systems not for ease of verification but for efficiency. And performance is not an unimportant concern—if the system does not execute fast enough, it may not be acceptable. The performance and ease of verification can sometimes be conflicting objectives. For example, designing for performance can cause the duplication of the information throughout the system. In contrast, in designing for verification, it would be better if the information is recorded in one place so that it can be verified once.

- Is it possible to achieve design for verification without sacrificing performance? If not, is it acceptable to sacrifice performance for ease of verification?

5 Topics of Discussion

The questions we raised above are all part of the following general questions that we asked the panel:

- What is a verifiable design? Can we identify verifiable designs?
- Are there design principles and techniques which can be used to develop verifiable designs?
- What classes of information can the designers record in software and hardware artifacts that would support automated verification tools?
- What languages can be used to record this information?
- What are the challenges for developing verifiable designs in practice?