Reactive System Validation Using Automatic Reasoning Over a Fragment Library

Robert J. Hall
AT&T Labs
Florham Park, NJ 07932
hall@research.att.com

Contact author regarding the forthcoming full paper (http://www.research.att.com/~hall/home.html).

1 Introduction

Reactive systems such as telephone switches, traffic controllers, and software agents are difficult to specify correctly with respect to true user requirements, because typically they must perform heterogeneous behaviors that interact robustly with each other and with complex environments. While it is relatively easy for the user to confidently state some scenarios of correct behavior, it is difficult to be sure that the specification behaves as desired on all possible scenarios.

In previous work on the ISAT (Interactive Specification Acquisition Tools) project [4], a central thrust has been to address the validation of reactive systems specifications (expressed in one of a number of accepted executable specification languages) through tools based on sound scenario generalization: a technique that automatically discovers a set of general situation/action rules guaranteed to soundly describe the specification’s behavior. Essentially, requirements are stated as concrete scenarios, i.e., interleaved sequences of events and observations, where the observation may be of an action or a state value. For example, here is a simple telephone scenario

```
INIT!
INIT-USER!(user1, "1234")
INIT-USER!(user2, "5678")
OFFHOOK!(user2)
OFFHOOK!(user1)
observe HEARS-DIAL-TONE?(user1) = true
DIAL!(user1, "5678")
observe HEARS-BUSY-TONE?(user1) = true
observe RINGING?(user2) = false
```

Each requirement scenario is analyzed into concrete fragments, each of which represents a single concrete observation in response to a single stimulus in a given state. Each such fragment is then given to the IBG-CF algorithm [4], which computes a sound generalization of it, producing a generalized fragment that schematically describes a situation/action rule soundly capturing one aspect of the system's response to a given stimulus in a given class of initial states. One such generalized fragment extracted automatically from the scenario above (given a particular telephone switch specification) is

Assuming:

\[ \text{OHHOOK!(user1) = false} \]
\[ \text{OHHOOK!(user2) = false} \]
\[ \text{EXT->USER(digit-string) = user2} \]
\[ \text{MODE(user1) = dialing} \]

Then Succeeds:

\[ \text{DIAL!(user1, digit-string)} \]
\[ \text{observe HEARS-BUSY-TONE?(user1) = true} \]

ISAT tools assist the user in compiling a collection of generalized fragments, together describing the behavior of the system in all situations. Acquiring this set, a fragment library, reduces the validation problem from checking an infinite set of concrete behaviors to checking a usually finite set of general behaviors.

2 The Problem

While a user might be able to confidently validate the generalized fragment shown above by inspection since calling an offhook user should result in a busy signal (when the system only has one line per user), more complex behaviors can be impractical to validate by inspection. This is particularly true when the fragment describes an intermediate protocol step, for example, because correct is often stated in terms of all possible protocol outcomes. The paper illustrates this problem with a fragment describing a step in the CS-NC protocol used by the personal channel.
agent (PCA[5], the paper’s primary case study. An example correctness property proved in the paper is

Property V: Given two properly initialized PAs, CS correctly transmits the data from one to the other, and an eavesdropper’s action(s) will be detected by at least one of the PAs, assuming (1) every protocol message sent is eventually received, and (2) only an eavesdropper can discover keys or channel identifiers (with significant probability).

3 The Approach

In the full paper, I describe novel techniques for applying automated reasoning to an evolving fragment library to address this problem. The techniques help bridge the gap between recognizable statements of what the user requires and automatically generalized descriptions of the systems behaviors. The techniques embody three automated reasoning principles. The *extension modal principle* states that in order to validate the behavior of a system in context, it can be useful to create an extended model whose behaviors strictly include those of an instance of the original system. Validate the extension model, and then conclude validity of the original system. A tool, Check-Extension, allows automatic checking of the extension property. The *fragment checking principle* asserts the soundness of applying a computational predicate describing a property to all possible partial state transitions (fragments) in order to verify the property. The *past-state diagram (PSD) principle* asserts the usefulness of proving that a certain abstract PSD models a specification, allowing one to deduce validation properties directly from the PSD. The techniques are embedded in implemented tools, including a layered automated reasoning system (an extension of techniques used in CAKE[7]). The paper illustrates these principles and techniques through application to an extended case study, the PCA[5, 6], involving a reactive system of moderate complexity with both “stand-alone” and “protocol-like” behaviors.

4 Related Work

I summarize related work briefly here. *Finite state model checking[6] is based on constructing a finite state machine model of a system and then exhaustively checking this model for conformance to user-defined properties. It allows automatic and exhaustive verification, but suffers from the state explosion problem. Symbolic model checking[7] handles models with orders of magnitude more states by reasoning symbolically over a description of the boolean next-state function, but itself suffers from the limited expressive power of binary decision diagram representations. ISAI integrates the bottom-up symbolic validation technique of generalization and validation by inspection with top-down automated reasoning for verifying properties. ISAI’s more expressive logic, powerful generalizer, and layered reasoner can handle finite state systems, such as system manipulating lists or other unbounded data structures, expressed in relatively rich input languages; thus, ISAI handles a wider range of systems. ISAI’s also more comprehensive, because the bottom-up techniques automatically discover state, and prove many notable correctness properties that might not otherwise occur to the user. The need for user interaction is the primary drawback of the ISAI approach.*

Traditional software testing is a special case of this approach (i.e., when generalization is just the identity function). Software test coverage metrics (e.g., [3]) provide only heuristic validation, however, since it is possible to completely cover all code branches and yet still have undetected faults. With a validated fragment library, the system is proved complete coverage, by contrast, one is sure that the model is valid for all inputs.

The layered reasoner is not essential; other theorem provers could be used instead, especially those integrating term rewriting, such as A0Cl/eq/rew[1], since ISAI validation requires term simplification.

References