Mining Scenario-Based Specifications with Value-Based Invariants

David Lo
School of Information Systems
Singapore Management University
davidlo@smu.edu.sg

Shahar Maoz
Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Israel
shahar.maoz@weizmann.ac.il

Abstract

There have been a number of studies on mining candidate specifications from execution traces. Some extract specifications corresponding to *value-based invariants*, while others work on inferring *ordering constraints*. In this work, we merge our previous work on mining scenario-based specifications, extracting ordering constraints in the form of *live sequence charts* (LSC), a visual specification language, with Daikon, a tool for mining value-based invariants. The resulting approach strengthens the expressive power of the mined scenarios by enriching them with *scenario-specific value-based invariants*. The concept is illustrated using a preliminary case study on a real application.

Categories and Subject Descriptors D.2.7[Software Engineering]: Distribution, Maintenance, and Enhancement – Reverse Engineering; D.2.1 [Software Engineering]: Specifications

General Terms Algorithms, Design, Documentation

1. Introduction

A specification usually imposes constraints both on sequencing of method calls or statement executions (ordering constraints), and on the values that a method parameters or some variables at a program point could have (value constraints). One tells a separate picture from the other, and each independently, although interesting, might not be able to present the full picture on the requirements that a system should follow.

Motivated by the lack of documented specifications, recently a number of studies have investigated mining of specifications from program executions. One pioneering work, Daikon, mines for value-based invariants that hold at specified program points [3]. Recently, we have investigated mining an expressive visual sequence-diagram-like scenario-based specification in the form of *live sequence charts* (LSC) using a data mining approach [6]. However, [6] has only considered ordering constraints among method calls. In this work, we merge the two approaches. The resulting approach strengthens the expressive power of the mined scenarios by enriching them with value-based invariants.

We are interested in universal live sequence charts (LSC) [2, 4], which are sequence diagrams divided into pre-chart (whose events are drawn with blue, dashed lines) and main-chart (red, solid lines). An LSC specifies a universal requirement: whenever the events in the pre-chart occur in the specified order, eventually the events in the main-chart must occur in the specified order. The semantics of LSC is defined in [2, 4]. In scenario-based specification mining [6], we are interested in mining statistically significant LSCs, ones which occur frequently in the trace and whose pre- is followed by the main-chart with high likelihood.

Fig. 1(left) shows an example LSC; the blue segment corresponds to the pre-chart, while the red segment corresponds to the main-chart. A chart consists of lifelines, denoting objects participating in a scenario, and messages, corresponding to method invocations. The example LSC states that whenever the `RootPanel` calls the `start()` method of `FTPServer`, eventually, the `FTPServer` will call the `updateStatus()` method of `RootPanel` and `RootPanel` will call the `stop()` method of `FTPServer`. In Fig. 1(right), we have guards on the executions of method calls represented as conditions on the messages between lifelines. While the previous work in [6] has mined for the ordering constraints, it is not able to mine for the value-based invariants and add them to the mined scenario. This is the challenge addressed in our present work. In this abstract we omit formal definitions and summarize technical details.

2. Mining Framework

Mining for LSCs with value-based invariants serving as guards consists of the following 4-steps process:

1. Mine LSCs without value-based invariants using the technique in [6][1]
2. For each mined LSC, slice the trace considering only the events involved in the witnesses of the scenarios corresponding to a selected mined LSC.

OOPSLA 2009, October 25–29, 2009, Orlando, Florida, USA.
ACM 978-1-60558-768-4/09/10.

[1] In this study, for simplicity, we remove object identities from the traces. Interested readers could note the discussion in [5, 6].

3. Provide the sliced traces to Daikon and let it search for value-based invariants.
4. Merge the mined Daikon invariants with the corresponding LSC, attaching them as guards to the methods between the lifelines.

The first step above mines for significant LSCs in the execution traces. An LSC is considered significant if its instances appear more than a minimum number of times and the pre-chart is followed by the main-chart with a high likelihood. We borrow the terms support and confidence from data mining to indicate the number of times an LSC instance appears in the trace and the proportion of pre-charts followed by main-charts. The user specifies minimum threshold parameters for support and confidence to determine whether a chart is detected as significant or not.

Each LSC induces a set of witnesses, namely segments of the trace where the LSC is satisfied. To extract for value-based invariants pertaining to a specific scenario we slice the trace so as to consider only the events that are part of these witnesses' trace segments. The sliced traces are fed to the next step for Daikon to mine for value-based invariants.

In the third step, Daikon would consider various values that hold at a set of program points (in particular, we consider method entries and exits) and then compute likely invariants by generalizing the values that it sees for each program point under consideration. Note that for scenario-specific invariants, we are not interested in invariants that hold across the entire execution of a program. Instead, we are interested in properties that hold in the context of a particular scenario under consideration, in all its instances. For example, in an application the same set of methods could be used for various scenarios, however the value-based invariants that hold at each of the separate scenarios could be different. By slicing the input traces according to the scenarios, we are able to instruct Daikon to only mine for *scenario-specific value-based invariants*.

In the final step, we merge the scenario under consideration with the mined scenario-specific value-based invariants, resulting in a more expressive mined specification: LSCs with guards capturing both ordering and value-based constraints. The mined specifications could be presented visually to the user, converted to runtime monitors [8], used for verification, etc.

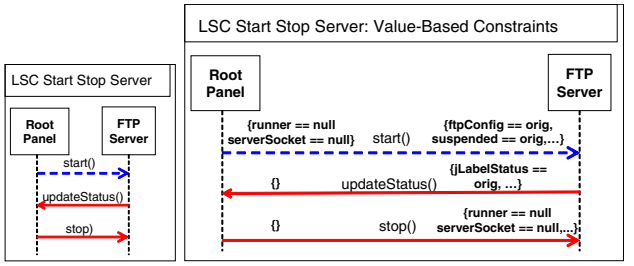

Figure 1. Mined LSCs: Without & With Value Invariants

3. Preliminary Experiments

We experimented with crossFTP [1], in particular methods from the `RootPanel` and `FTPServer` classes. CrossFTP is a commercial open source FTP server built on top of Apache FTP server spanning 18841 LOC. A mined LSC without value-based invariants is shown in Fig. 1 (left). This LSC was enriched with value-based invariants mined via Daikon after trace slicing, to form the LSC shown in Fig. 1 (right). Note the constraints in the enriched LSC. For example, the `runner` and `serverSocket` attributes of `FTPServer` are always null before `start()` method is called. Also, after method `stop()` is called, the `runner` and `serverSocket` are always null. Other invariants state that some instance variables are unchanged by the method call. For example, `ftpConfig == orig` states that the value of the variable `ftpConfig` is the same before and after the call to `start()`. These invariants hold in the context of the mined scenario, but do not necessarily hold when the same methods are used in other contexts.

4. Discussion & Future Work

Lorenzoli *et al.* merge a technique to mine an automaton model with value-based invariants from Daikon [7]. Different from their work, we infer scenario-based specifications with value-based invariants. The automaton model mined in [7] reports the overall model under consideration, while our mined scenarios report strongly observed scenario-based invariants. The scenarios specify not only method call signatures but also their caller and callee information and come with a universal interpretation rather than a weaker existential interpretation.

We have experimented mining of LSCs with value-based invariants on a preliminary case study. Planned future work includes further evaluation using additional case studies.

References

[1] CrossFTPServer. sourceforge.net/projects/crossftpserver/.

[2] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *J. on Formal Methods in System Design*, 19(1):45–80, 2001.

[3] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *TSE*, 27(2):99–123, 2001.

[4] D. Harel and S. Maoz. Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. *Software and Systems Modeling*, 7(2):237–252, 2008.

[5] D. Lo and S. Maoz. Mining Symbolic Scenario-Based Specifications. In *PASTE*, 2008.

[6] D. Lo, S. Maoz, and S.-C. Khoo. Mining Modal Scenario-Based Specifications from Execution Traces of Reactive Systems. In *ASE'07*, 2007.

[7] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic Generation of Software Behavioral Models. In *ICSE*, 2008.

[8] S. Maoz and D. Harel. From Multi-Modal Scenarios to Code: Compiling LSCs into AspectJ. In *FSE*, 2006.