Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 144 (2006) 109–124
www.elsevier.com/locate/entcs

Temporal Assertions using AspectJ

Volker Stolz[1] and Eric Bodden[1]

Dept. of Computer Science
Programming Languages and Program Analysis
RWTH Aachen University
52056 Aachen, Germany

Abstract

We present a runtime verification framework for Java programs. Properties can be specified in Linear-time Temporal Logic (LTL) over AspectJ pointcuts. These properties are checked during program-execution by an automaton-based approach where transitions are triggered through aspects. No Java source code is necessary since AspectJ works on the bytecode level, thus even allowing instrumentation of third-party applications. As an example, we discuss safety properties and lock-order reversal.

Keywords: Runtime verification, LTL, AspectJ, aspect-oriented programming, alternating automata.

1 Introduction

To avoid misbehaviour, many software products include assertions which check that certain states on the execution path satisfy given constraints and otherwise either abort execution or execute specific error-handling. These assertions are usually limited to testing the values of variables. However, often it would be convenient not only to reason about a single state but also about a sequence of states. This enables the developer to reason about control flow and execution paths. In previous work [15], we discussed a symbolic checker for parametrised LTL formulae over finite paths which allowed us e.g. to reason about a problem found in multi-threaded applications commonly referred to as *lock order reversal*. At runtime, potential problems would be pointed out

[1] Email: {stolz,bodden}@i2.informatik.rwth-aachen.de

doi:10.1016/j.entcs.2006.02.007

to the developer. We have implemented a working prototype with similar functionality for Java applications using a symbolic checker based on the code generation approach we describe in the following. The major contribution of this work is to propose an alternative, automaton-based solution which allows for even more expressiveness, though yielding better performance.

A major drawback of the former approach was that annotations driving the checker had to be inserted into the source code of the application under test. For practical purposes, we supplied an annotated replacement for the concurrency-library. Applications wishing to use this framework thus had to be recompiled.

Also, such source code-based hooks may lead to severe problems with respect to object-oriented properties such as behavioural subtyping: Any specification which is stated for a certain class should also hold for all its subclasses. The use of source code annotations within method bodies does not reflect such implicit rules. Thus we restrict our formalism in such a way that it only allows to reason about well-defined interfaces, meaning method calls and field accesses. The recent success of the utility Valgrind [11] shows that there is sufficient demand for better debugging support (in contrast to techniques like model checking).

In Section 2 we give a short introduction into temporal logic. Instead of symbolically checking a formula, we use a translation into an alternating automaton. Section 3 discusses instrumentation of (Java) applications and focuses on aspect-oriented programming in AspectJ. The combination of LTL and AspectJ in Section 4 yields a state-machine for each formula to be checked through a finite automaton where transitions are driven by an aspect. We discuss an extension to parametrised automata for handling recurring patterns with state and conclude in Section 5.

2 From LTL to alternating automata

In this section we give a finite path semantics for LTL and remind the reader on how to translate LTL formulae into alternating automata.

2.1 Path semantics for LTL

Linear-time temporal logic (*LTL*) [12] is a subset of the Computation Tree Logic CTL* and extends propositional logic with operators which describe events along a computation path. The operators of LTL have the following meaning:

- "Next" ($\mathbf{X}\ \varphi$): The property φ holds in the next step