Runtime monitoring of finite state properties
Seminar: Advanced Verification Techniques

Patrick Backhaus
University of Paderborn

Research Group Specification and Modelling of Software Systems
Prof. Dr. Heike Wehrheim

21. January 2013
Motivation

- each verification technique has its drawbacks

\(^1\) Figures from [Falcone(2010)]
each verification technique has its drawbacks

- model-dependent,
- state-explosion, slightly different behavior of implementation

\(^1\) Figures from [Falcone(2010)]
Motivation

- Each verification technique has its drawbacks:
  - Model-dependent, state-explosion, slightly different behavior of implementation.
  - Incomplete, hard to build good test suite.

1. Figures from [Falcone(2010)]
→ *Runtime Monitoring* as alternative or additional validation

- Does *current execution* preserve specified properties?
- possibility to react at detected malfunction
- trade-off between scalability and coverage

+ almost always applicable
Outline

1 Introduction
   - Terminology
   - Application areas

2 Algorithmic Approaches
   - Object-based Monitoring
   - State-based Monitoring
   - Symbol-based Monitoring

3 Analysis

4 Optimizations

5 Tools

6 Conclusion
Runtime Verification

**Definition**

Runtime verification is the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a run of a system under scrutiny satisfies or violates a given correctness property [Leucker(2012)].

- run of a system = possibly infinite sequence of system’s states
- execution of a system = finite prefix of a run = finite trace
Monitor

Definition

A monitor is a device that reads a finite trace and yields a certain verdict [Leucker(2012)].

- Input: executing software and property
- checks if current trace satisfies given property
- Output: Verdict like true/yes or false/no
Monitor characteristics

Stage & Placement:
- Offline: trace is recorded and analyzed after the run
- Online: trace is analyzed at runtime
  - outline: monitor runs in parallel to the system
  - inline: monitor’s code is implanted into the system code

Monitoring Points:
- Manual: manual instrumentation of code (e.g. source code or byte code)
- Automatic: instrumentation points are discovered by tools
Properties

- properties can be defined as a set of sequences of states
- used by monitors to discover faults
- given in trace predicate formalism, such as *finite state machines*, regular expressions, linear temporal logics, etc.
- monitors are synthesized from them and embedded into the system (instrumentation)
Overview

Figure from lecture slides "Introduction to Runtime Verification" Irisa 2010
Application areas

- software fault detection
- diagnosis, debugging
- recovery
Application areas

- software fault detection
- diagnosis, debugging
- recovery
- performance analysis, profiling
- software optimization
Outline

1 Introduction
   - Terminology
   - Application areas

2 Algorithmic Approaches
   - Object-based Monitoring
   - State-based Monitoring
   - Symbol-based Monitoring

3 Analysis

4 Optimizations

5 Tools

6 Conclusion
General Procedure

- property given as finite state automaton (FSA)
- create monitor instance for each property related program object
- at each call on object an event is generated
- monitors are updated accordingly to the events
- check for violation
- algorithms differ in how they process the sequence of events
Running example

**Figure**: HasNext Property FSA and instrumented code
[Purandare et al.(2012) Purandare, Dwyer, and Elbaum]

- OBSERVE.hasNext(i) → event (i, hasNext)
Running example - annotations

1. starting with an existing monitor for iterator i1 in state 0
2. creation event (i, hasNext) is generated
3. event (i, next) is generated
defines monitor as set of related objects and a state of the FSA

Figure: Example of Object-based Monitoring
[Purandare et al.(2012) Purandare, Dwyer, and Elbaum]
for each state of FSA a set of monitors recording related objects

Figure: Example of State-based Monitoring
[Purandare et al.(2012)Purandare, Dwyer, and Elbaum]
Symbol-based Monitoring

- sets of monitors relevant to each symbol of property FSA

Figure: Example of Symbol-based Monitoring
[Purandare et al. (2012) Purandare, Dwyer, and Elbaum]
Outline

1. Introduction
   - Terminology
   - Application areas

2. Algorithmic Approaches
   - Object-based Monitoring
   - State-based Monitoring
   - Symbol-based Monitoring

3. Analysis

4. Optimizations

5. Tools

6. Conclusion
Strengths and weaknesses of algorithms

- performance highly depends on property and program

- symbol & state-based monitoring: frequently executed self-loops

- object-based monitoring: frequent state changes

- number of states and symbols impact state and symbol-based monitoring
Strengths and weaknesses of RM

- easy to put in practice
- often significant runtime overhead
- strongly dependent on the number of created monitors and generated events
- thus the best choice depends on the property and the program
General approaches

- static analysis and ahead-of-time techniques
- exploiting program and property structure
- state and flow analysis
- reducing the required instrumentation
- Examples:
  - guaranteed successful checks can be removed
  - reducing automaton using equivalence classes
Stutter-Equivalent Loop Transformation

- optimizing monitoring of loops relative to property
- properties typically insensitive to number of loop iterations
- determines loop iteration after which behavior does not change
- only first few iteration of loop needs to be instrumented
  → loops processed in constant time
Outline

1 Introduction
   - Terminology
   - Application areas

2 Algorithmic Approaches
   - Object-based Monitoring
   - State-based Monitoring
   - Symbol-based Monitoring

3 Analysis

4 Optimizations

5 Tools

6 Conclusion
JavaMoP

- framework for software development and analysis
- developer add formal property specification on top of a target programming language
- automatic generation and integration of monitors
- implements object-based monitoring
- supports inline, offline monitoring
Tracematches

- aspect oriented
- monitors observe computation trace for specified sequences of events
- triggers the execution of extra code
- implements state-based monitoring
Conclusion

- Runtime Monitoring is a simple lightweight verification technique
- should complement techniques such as model checking and testing
- well scalable
- no need for formal modelling of the system
- can guide execution
- impact on runtime
Discussion

Questions?
Bibliography I

Ylies Falcone.
Introduction to runtime verification.
Lecture in the M2RI Rennes, September 2010.

Martin Leucker.
Teaching runtime verification.
Springer-Verlag.