Software Active Online Monitoring Under Anticipatory Semantics

Changzhi Zhao, Wei Dong, Ji Wang, Zhichang Qi

National Laboratory for Parallel and Distributed Processing P.R.China

7/21/2009
Overview

- Software active online monitoring under anticipatory semantics
  - Introduction
  - Anticipatory semantics of monitor
  - Software active online monitoring
    - The problem
    - The architecture
- The tool
software active online monitoring

Introduction

✓ The correctness and dependability of software are very important in safety-critical system.

✓ To make sure that the software systems are really safe, a lot of methods have been extensively studied and applied.

  • Static verification and testing

✓ These efforts still can not assure the system will function correctly in runtime.
software active online monitoring

Introduction

✓ Monitoring of the running system is critical.
  - Online monitoring
  - Offline monitoring
  - Online monitoring is necessary in order to tune the behavior of system in time if needed.

✓ Runtime verification is one of the important methods for software monitoring
  - Suit online monitoring
  - Pay the way for not only Detecting incorrect behavior of a software system but also for reacting and potentially healing the system.
Software active online monitoring

- Introduction
  ✓ Disadvantage of current runtime verification
    - Passive online monitor
    - Can not predict violations in advance
    - Can not prohibit the occurrence of the failure
    - The reason:
      - they are not based on the anticipatory semantics
      - do not concern with the information about the system model under scrutiny.
software active online monitoring

- Introduction
  ✓ Our ongoing work about software active online monitoring
    - Improve the traditional runtime verification to active monitoring
    - Not repairing the fault after it has been detected
    - But predicts the faults in advance and triggers the control actions to prevent the software from failures
    - Exploit limited system knowledge to predict violations
    - Active online monitoring is not only analyzing the current event but also looking ahead into a partial system model to explore the state space in advance.
    - If non-conformance has been detected, correspondence control actions will be taken to prevent the system from reaching a violation.
software active online monitoring

- Anticipatory semantics of monitor
  ✓ In active online monitoring, the triggering of control actions will be based on the verdict of runtime verification, so the monitoring semantic is vital important.
  ✓ Aim at deriving a verdict whether there might be any infinite executions satisfied/dissatisfied a correctness property by considering the finite prefixes.
Anticipatory semantics of monitor

To capture implication of above idea, a monitor should follow two maxims:

- Impartiality: a finite trace is not evaluated to true or false, if there still exists an (infinite) continuation leading to another verdict.
- Anticipation: once every (infinite) continuation of a finite trace leads to the same verdict. Then the finite trace evaluates to this verdicts.

Intuitively, the first maxim postulates that a monitor decides for false, meaning that a misbehavior has been observed; for true, meaning that the current behavior fulfills the correctness property, regardless of how it continues.

The second maxim requires the monitor has the ability to report true or false, once the correctness is indeed violated or satisfied.
Anticipatory semantics of monitor

Given the monitored property \( \varphi \), based on the infinite trace semantic of temporal logic formula and the finite trace which has been observed so far, the definition as follows:

\[
[\pi \models \varphi] = \begin{cases} 
    \text{true} & \text{if } \forall \sigma \in \Sigma^\omega : \pi \sigma \models \varphi; \\
    \text{false} & \text{if } \forall \sigma \in \Sigma^\omega : \pi \sigma \nvdash \varphi; \\
    ? & \text{otherwise.}
\end{cases}
\]
software active online monitoring

- Anticipatory semantics of monitor
  - The above definition is also called three-valued semantics
  - Currently, multiple-valued semantics (more than 3) has been proposed.
  - It is obvious that anticipatory semantics is satisfied with Impartiality and Anticipation.
    - It is impartial since for every prefix $\pi \in \Sigma^*$ with two continuations $\sigma, \sigma' \in \Sigma^\omega$ such that $\pi \sigma \models \varphi$ and $\pi \sigma' \not\models \varphi$, the semantics of $[\pi \models \varphi]$ evaluates to inconclusive $\_\_\_$. 
    - It is anticipatory since once only satisfying or unsatisfying continuations exist, the semantics $[\pi \models \varphi]$ evaluates to the corresponding true or false.
Anticipatory semantics of monitor

Traditional runtime verification is based on the finite trace semantics, it does not obey the two maxims.

For example, one typical definition of finite trace semantics for LTL is:

\[
\begin{align*}
t \models true & \iff true, \\
t \models false & \iff false, \\
t \models A & \iff A \in head(t), \\
t \models X \land Y & \iff t \models X \text{ and } t \models Y, \\
t \models X \lor Y & \iff t \models X \text{ xor } t \models Y, \\
t \models \circ X & \iff (\text{if } \text{tail}(t) \text{ is defined then } \text{tail}(t) \models X \text{ else } t \models X), \\
t \models <>X & \iff (\exists i \leq \text{length}(t)) \ t_i \models X, \\
t \models []X & \iff (\forall i \leq \text{length}(t)) \ t_i \models X, \\
t \models X \cup Y & \iff (\exists i \leq \text{length}(t)) \ (t_i \models Y \text{ and } (\forall j < i) \ t_j \models X).
\end{align*}
\]
软件在线活动监控

- Anticipatory semantics of monitor
  - Intuitively, the finite trace semantics is only concerned with the portion of the execution that we have observed:
    - One solution: regard a finite trace as an infinite stationary trace in which the last event is repeated infinitely.
    - Another solution: for any position which is longer than the length of the finite trace, the property is dissatisfied.
  - The finite trace LTL can behave quite differently from standard infinite trace LTL.
    - For example, $\diamond (\square A \lor \neg \square A)$ is not valid in infinite trace LTL but valid in finite trace LTL.
software active online monitoring

- Anticipatory semantics of monitor
  ✓ Anticipatory semantics is suitable for software active online monitoring.
    - Only if the verdict is false, the corresponding control actions need be triggered
    - Only if the verdict is true, the monitor can terminate
    - Otherwise, the monitoring process should continue.
Software active online monitoring

- The problem
  - It is more than runtime verification
  - It can be decomposed into two part:
    - Analyzing partial system model which is obtained through looking ahead to detect nonconformance (prediction).
    - Applies control actions to the system (prevention).

- Assumption:
  - The active monitor and the monitored system reside in different machines and executing concurrently.
  - $N$ is the round-trip communication delay between the active monitor and the monitored system.
  - The processing delay of the active monitor is negligible compared with the communication delay.
software active online monitoring

- The problem
  ✓ Modeling the monitored system $M_s$ as a transition system $M = \langle S, s_0, T, \Sigma \rangle$:
    - $S$ is the state space
    - $s_0$ is the initial state
    - $\Sigma$ is the event set which are controllable by the monitor
    - $T : S \times \Sigma \rightarrow S$ is the transition function
  ✓ The construction and working process is as follow:
    - The user identifies a set of property-relevant operations.
    - The user finds all the sequences of these operations that is allowed in normal and encodes them using transition system $M$.
    - The sequence of property-relevant operations performed during program execution, as the program executes a property-relevant operation, $M$ transitions to a new state.
software active online monitoring

- The problem
  - M-step partial model of the system
    - the runtime model which originates from the current state.
    - \( n \) is the communication delay between monitor and system, then \( n+1 \) is minimum amount of looking ahead required to ensure that control actions will be receive by the system in time.
The problem

✓ The runtime model can be defined as $M_s = \langle S', R, \Sigma, s, F, C, T' \rangle$:

- It is a tree-like structure with bounded depth
- $S'$ is the set of the runtime states
- $R$ is a function which maps runtime states in $S'$ to static states of $M$
- $s \in S'$ is the initial state, $\Sigma$ is the event set.
- $T'$: $S' \times \Sigma \rightarrow S'$ is a transition such that $R(T'(s',e)) = T(R(s'),e)$ where $e \in \Sigma$
- $F \in S'$ is the set of final states of the runtime model within $n+1$ steps
- $C \in S'$ is the set of the control states defined as $C = \{s' \in S' \mid \exists e \in \Sigma, f \in F, T'(s',e) = f\}$. 
The problem

✓ Given the communication delay $n$, the system model $M$, the current state $s$, the problem is:

- Given the runtime model $M_s$ which originate from state $s$.
- $\pi$ is the finite trace which has been observed so far, $\sigma \in \Sigma^*$ is any path from $s$ in runtime model, check whether $F' = \{ \sigma \in \Sigma^* \mid [\pi\sigma] = \varphi = \text{false} \}$ is empty.
- Generate appropriate control actions to ensure that the system does not reach violated states by restricting the runtime model.
- Determine the mechanism for executing the control actions in the system.
The architecture

In terms with the problem described above, we present the architecture as follow:
software active online monitoring

- The architecture
  - The function of each component:
    - Instrumented system:
    - Controller:
    - Partial model generator:
    - Anticipating runtime verifier:
    - Control action generator:
The tool

- The implementation of the tool is based on MAC framework:
The tool

- During the design phase, the system specification and requirements on the system are specified.
- During the implementation phase the system is implemented.
- Based on the requirement specification and the implementation, the user provided monitoring script
  ✓ Contain instructions for instrumenting the code so that low-level information about program state can be passed on to monitor
  ✓ Contain information that can be used to produce an event recognizer that transforms the low-level information into high-level events
The tool

- Runtime phase consists of three components: filter, event recognizer, and runtime checker.
  - The filter extracts low-level information
    - For example, values of program variables and time when variables change their values)
    - Instrument the code
  - Event recognizer map the low-level information to high-level information
    - Based on the values of the monitored variables it receives from the filter, event recognizer detects the occurrence of events that are described in the requirement specification
The tool

✓ Runtime checker

• According to a requirement specification of the system, base on the information of events it receives from the event recognizer, and on the past history, check the correctness of a sequence of events

• Based on anticipatory semantics
The tool

- Steering script (SADL) to SADL parser
- Program (Java byte code) to Instrumentation data
- Monitoring script (PEDL) to PEDL parser
- Requirements (MEDL) to MEDL parser

Instrumentor

Java virtual machine

Instrumented Program

Injector

parsed PEDL

Event recognizer

parsed MEDL

Checker

MaCware

run-time dataflow
compile-time dataflow
The tool

- The reason for keeping the monitoring script distinct from the requirements specification:
  - Maintain a clean separation between
    - the system itself implemented in a certain way
    - High-level system requirements independent of a particular implementation
  - Allow us to perform monitoring of heterogeneous distributed systems
    - A separate event recognizer may be supplied for each module
    - each event recognizer may process the low-level data in a different way
    - All deliver high-level events to the checker in a uniform fashion
The tool

- The tool for active online monitor

System model

Runtime model

Steering script (SADL)

Program (Java byte code)

Monitoring script (PEDL)

Requirements

SADL parser

Program parser

Monitoring script parser

Requirements parser

Instrumentation data

Instrumentation data

Instrumentor

Java virtual machine

Instrumented Program

Injector

parsed PEDL

Event recognizer

Checker

parsed MEDL

MaCware

run-time dataflow

compile-time dataflow
The tool

- Through the case study, we can see that
  - The theory study about active online monitor is feasible and practical
  - The framework is perfect
  - The disadvantage:
    - The generation of the model
      - automatic generation from the program
      - give an language mechanism to make the user descript the model easier.
    - The preciseness of the model
Thanks!