Checking compliance of Java implementation with TBP Specification

Pavel Jančík
Motivation

Specification:

Implementation:

Protocol compliance check

tbpjava checker

Compatible?

Component1

Component2
component Recorder {
  types {
    states = { NOT_REC, REC }
    hwstate = { OK, BAD }
  }
  vars {
    states state = NOT_REC
  }
  provisions {
    standard {
      ?P1.initialize() : OK ;
      ?P1.startRecording() ;
      ?P1.stopRecording() ;
      ?P1.finalize()
    }
  }
  reactions {
    P1.checkHW() : hwstate {
      state<-REC ;
      switch(!R1.getAudioState()) {
        INIT : { return OK }
        REC : { return BAD }
      }
    }
  }
  threads {
    recording_thread { ... }
  }
}
**Typical errors**

```java
public class PricingManagerImpl implements PM {
    ...

    public Price calcVolPriceForCustCat(String custCatKey, String custType) {
        Price volumePrice = new Price();
        CustCatDiscount custCatDisc = custCatDiscDao.getByCatKey(custCatKey);
        if (custCatDisc == null) {
            return null;
        }
        CustTypeDiscount custTypeDisc = custTypeDiscDao.getByName(custType);
        if (custTypeDisc == null) {
            ...
        }
    }
}
```

```java
component PricingManager {
    ...
    reactions {
        ...
        PM.getAllVolumeDiscounts() {
            !VolumeDiscountDao.getAll()
        }
        PM.calcVolPriceForCustCat() {
            !CustCatDiscDao.getByCatKey(custCatKey);
            !CustTypeDiscDao.getByName(custType)
        }
        PMgr.priceOrder() {
            NULL
        }
    }
    ...
}
```
public class PricingManagerImpl implements PM {
...

public Price calcVolPriceForCustCat(String custCatKey, String custType) {

    Price volumePrice = new Price();
    CustCatDiscount custCatDisc =
        custCatDiscDao.getByCatKey(custCatKey);
    if (custCatDis == null) {
        return null;
    }
    CustTypeDiscount custTypeDisc =
        custTypeDiscDao.getByName(custType);
    if (custCatDis == null) {
        ...
    }
    PMgr.priceOrder() {
        NULL
    }
...}
Usage

Component implementation
DB of param values
Component interfaces
Behavior protocol
Metadata

JPF checker

☑ Compatible

Error found
Event error trace
?Pltf1.init()
!Rltf2.init()
?Pltf1.compute()
Checker structure

- Environment generator
  - Required interfaces → stubs
  - Provisions → “environments”
- Compilation of environment
- JPF + ProtocolChecker
  - Records executed events (calls/returns)
  - Updates state → according automatons
Ukázka prostředí – and-parallel

A | B

Semantics
(A ; B) | C ↔
(A ; B ; C) + (A ; C ; B)

Thread t1 = new Thread() {
    public void run() {
        // Code for operand A
        ... A;
    }
};
Thread t2 = new Thread() {
    public void run() {
        // Code for operand B
        ... B;
    }
};
t1.start(); t2.start();
t1.join(); t2.join();
Environment

Non-determinism in BP
- in provisions
  alternative, repetitions, parallel operators
- in reactions
  return values
⇒ encoded into the environment

- Generated from provisions
  return values → not parsing tree X automatons
Environment – Value DB

- Hand made
  - Error prone process
  - Some hints can be provided
- Quality of DB
  - Critical to quality of checking
  - No coverage guaranty
  - Degradation of stateful objects
- Small domains
  - Only single value domains applicable
Protocol checker

• Input
  - Events - provided/required calls/returns
    • Originate from JPF
      - Incrementally, according exploration of the state space
    • TBP protocol
      - Parsed by TBPLib
      • Automaton(s) for each thread and reaction
  - State
    • Values of TBP variables
    • Node (state) in the automaton – for each thread
• Protocol checker
  • Set of reachable states
State processing example

Thread 1

- logging <- LOG
- return

Evaluations

- After IFile.open
  - return event
  - Evaluation 1: logging = NO_LOG state=AUTHORIZED
  - Evaluation 3: logging = NO_LOG state=AUTHORIZED
  - Evaluation 2: logging = LOG state=AUTHORIZED
  - Evaluation 4: logging = LOG state=AUTHORIZED

Thread 2

- !<IFile.open>
- ?
- !<IFile.read>
- NULL
- ?
- ?
- !<IFile.close>
- return
New TBP Features

- Types & Variables
  - Representation of the state

- Conditions, Assignments
  - Updated automatons - “hidden” actions

- Synchronization

- Threads
  - Multiple automatons
New TBP Features - threads

- Actions originating in the component
- Realized as threads in the implementation

- Code checking
  - Threads are distinguishable
  - Mapping implementation onto TBP specification
New TBP Features - threads

Mapping implementation onto TBP specification

- by hand – annotations
  - efficient (no overhead)
- automatically
  - Processing of needles states
  - Progress
    - Map impl. thread to all TBP threads.
    - Remove infeasible combinations.
    - If program terminates → Check if exists bijection from impl. threads to the TBP threads. (Matching in bipartite graph)
Automaton traversal

How to find successor of the state?
All “interleavings” of the automatons.

Really all?
Ignore edges that don’t depend and modify the variables (NULL, ?)

Save state before each assignment
(can be moved later)
Remember condition (guards) that don’t hold
if assignment enables the guard explore this path too.
JPF Backtracking

How to restore previous state?

- Reverse automaton traversal
  - Reversed automaton are not deterministic
    ⇒ Additional information required (path, original var. values)

- Remember changes of the original state
  - All states are modified

- Store copy of the original state
  - Immutability may help
State matching

For each JPF state store:

- All visited states
  - Complete state
  - Hashes

- Event traces
  - Remember all processed events
  - Imprecise, traces can be shared
  - Cycles – lengthen the trace
## Experiments

<table>
<thead>
<tr>
<th>CashDeskApp</th>
<th>States</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard JPF</td>
<td>632 776</td>
<td>00:10:54</td>
<td>142 MB</td>
</tr>
<tr>
<td>JPF + TBPChecker</td>
<td>1 056 270</td>
<td>00:12:34</td>
<td>1076 MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IpAddressManager</th>
<th>full TBP / single value domain / time modeled</th>
<th>States</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard JPF</td>
<td>2 153 228</td>
<td>00:43:07</td>
<td>125 MB</td>
<td></td>
</tr>
<tr>
<td>JPF + TBPChecker</td>
<td>3 680 375</td>
<td>00:50:33</td>
<td>1775 MB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IpAddressManager</th>
<th>short TBP / single value domain / time modeled</th>
<th>States</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard JPF</td>
<td>56 661</td>
<td>00:00:56</td>
<td>121 MB</td>
<td></td>
</tr>
<tr>
<td>JPF + TBPChecker</td>
<td>100 423</td>
<td>00:01:23</td>
<td>121 MB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IpAddressManager</th>
<th>short TBP / dual value domain / time modeled</th>
<th>States</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard JPF</td>
<td>3 891 151</td>
<td>01:09:35</td>
<td>145 MB</td>
<td></td>
</tr>
<tr>
<td>JPF + TBPChecker</td>
<td>9 227 647</td>
<td>02:26:42</td>
<td>2541 MB</td>
<td></td>
</tr>
</tbody>
</table>

- Prone to state explosion
Questions ...

Thank you.