Combined Static and Dynamic Verification of Object Oriented Software Through Partial Proofs

Jesús Mauricio Chimento

10 October 2019
Program Verification

- Specification
- Program
- Verification Technique
  - Dynamic Verification
  - Static Verification
Program Verification

- Specification
- Deductive Verification
- Runtime Verification
- Program
- Verification Technique
Deductive Verification

- Properties written as logical formulae

\{ P \} \text{ foo() } \{ Q \}

- Formulae are verified by deduction in a calculus

\begin{align*}
\Gamma, \sigma(b) &\vdash \sigma <s_1 \omega > \phi & \Gamma, \sigma(\neg b) &\vdash \sigma <s_2 \omega > \phi \\
\hline
\Gamma &\vdash \sigma <\text{if } b s_1 \text{ else } s_2 \omega > \phi
\end{align*}
Runtime Verification

- Monitoring of program executions
Which verification technique shall we use?
Which verification technique shall we use?

Properties
Which verification technique shall we use?

- Control-oriented properties
- Data-oriented properties
Which verification technique shall we use?

Control-oriented properties

Data-oriented properties

Runtime Verification

Deductive Verification
Which verification technique shall we use?

Control-oriented properties

Data-oriented properties

Runtime Verification

Deductive Verification
Which verification technique shall we use?

Control-oriented properties

Data-oriented properties

Runtime Verification

Deductive Verification
Which verification technique shall we use?

Control- and Data-oriented properties

- Runtime Verification
- Deductive Verification
Which verification technique shall we use?

- Control- and Data-oriented properties
- Runtime Verification
- Deductive Verification
How to combine verification techniques?
Program Verification

Specification

Program

Verification Technique

S

Program
Program Verification

Verification Technique

Specification

Program
• **Part 1:** Unified Static and Runtime Verification of Object Oriented Software

• **Part 2:** Testing Meets Static and Runtime Verification

• **Part 3:** Inferring Global Trace Conditions from Partial Local Proofs
• **Part 1:** Unified Static and Runtime Verification of Object Oriented Software
Verification Framework

Unified Static and Runtime Verification of Object-Oriented Software

ppDATE

Program P

Program P' (instrumented)

Monitor
Verification Framework

ppDATE

Program P

Unified Static and Runtime Verification of Object-Oriented Software

Program P' (instrumented)

Monitor
Verification Framework

- ppDATE
- Program P
- Program P' (instrumented)
- Monitor
ppDATE Specification Language

- ppDATE:
  - event-triggered transitions
  - Zero or more Hoare triples in each state of the automata
  - Parallel automata, communication
  - Templates, ppDATEs creation
  - Structural Operational Semantics
High-level description of StaRVOOrS

1. Specification
2. Refinement
3. Deductive Verification
4. Translation and Instrumentation
5. Monitor Generation
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}
High-level description of StaRVOOrS

```
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}
```
High-level description of StaRVOOrs

```c
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}
```
High-level description of StaRVOOrS

```plaintext
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}
```
High-level description of StaRVOOrS

foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}

{P} foo() {Q}

\[ \Gamma, \sigma(\text{cond}) \vdash \sigma \langle E1 \omega \rangle Q \quad \Gamma, \sigma(\neg \text{cond}) \vdash \sigma \langle E2 \omega \rangle Q \]

\[ \Gamma \vdash \sigma \langle \text{if cond E1 else E2} \omega \rangle Q \]
High-level description of StaRVOOOrS

```plaintext
foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}

{ P } foo() { Q }
```

<table>
<thead>
<tr>
<th>Q(E1)</th>
<th>Q(E2)</th>
<th>KeY</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Diagram showing steps:
- Deductive Verification
- Specification Refinement
- Translation and Instrumentation
- Monitor Generation

Table: Q(E1) and Q(E2) with KeY values.
High-level description of StaRVOOOrS

foo() {
    if (cond) {
        E1 ;
    } else {
        E2 ;
    }
}

{ P ∧ !cond } foo() { Q }

<table>
<thead>
<tr>
<th>Q(E1)</th>
<th>Q(E2)</th>
<th>KeY</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>✗</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
High-level description of StaRVOOrS

- Deductive Verification
- Specification Refinement
- Translation and Instrumentation
- Monitor Generation
High-level description of StaRVOOrS
High-level description of StaRVOOrS

- Deductive Verification
- Specification Refinement
- Translation and Instrumentation
- Monitor Generation

LARVA
High-level description of StaRVOOrS

- Deductive Verification
- Specification Refinement
- Translation and Instrumentation
- Monitor Generation

LARVA
High-level description of StaRVOOOrS

- Deductive Verification
- Specification Refinement
- Translation and Instrumentation
- Monitor Generation
StaRVOOrS

- Fully automatic tool
- starvoors.github.io

(https://github.com/starvoors/StaRVOOrS-tool)
• **Part 2:** *Testing Meets Static and Runtime Verification*
Test Focus Development

Test Driven Development

Behaviour Driven Development

data

control
Test Focus Development
Test Focus Development

Data
Methods Signature Definition → Test Driven Development → Model Definition → Model-Based Testing

Control

Deductive Verification + Runtime Verification
Test Focus Development

Contracts Definition → Test Driven Development

Deductive Verification

Methods Signature Definition → Contracts Verification

(partial) Proofs

Test Case Generation (symbolic execution)

Test Cases

Model Definition

Runtime Verification

Monitor Generation

Monitor Execution

Monitor Spec.

Monitor Extension

Model Translation

Model-Based Testing

Overall Implementation
**Example**

- Define the methods signature

```java
/**
 * Deletes entry at <tt>key</tt> from the hashtable.
 * @param key of the removed object
 * @return removed object
 */
public Object delete (int key) { }
```
Example

• Define contracts for the methods

/*@ public normal_behavior
   @ requires key >= 0 ;
   @ requires h[hash(key)] != null ;
   @ requires size > 0 ;
   @ ensures \result == \old(h[hash(key)]) ;
   @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
   @ ensures (∀ int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
   @ assignable size,h[*] ;
   @ also
   ..... 
   @*/
public Object delete (int key) { }
Example

- Apply TDD (at least one test per contract)

```java
// public Object delete (int key) {}

@Test
public void test_delete_1() {  
    hash.add(new Integer(42), 0);
    hash.add(new Integer(3), 1);
    HashTable aux = new HashTable(3) ;
    aux.add(new Integer(3), 1);
    Object res = hash.delete(0);
    assertEquals(res, new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```
Example

- Apply TDD (at least one test per contract)

```java
/*@ public normal behaviour
 @ requires key >= 0 ;
 @ requires h[hash(key)] != null ;
 @ requires size > 0 ;
 @ ensures \result == \old(h[hash(key)]) ;
 @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
 @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
 @ assignable size,h[*] ;
 @ also
......
 @*/

public Object delete (int key) {
    if (key >= 0) {
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key];
            h[key] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
```

```java
@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);
    HashTable aux = new HashTable(3) ;
    aux.add(new Integer(3),1);
    Object res = hash.delete(0);
    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```
Example

- Deductive verify the implementation

```java
/**
 * @public normal_behaviour
 * @requires key >= 0;
 * @requires h[hash(key)] != null;
 * @requires size > 0;
 * @ensures result == \old(h[hash(key)]);
 * @ensures h[hash(key)] == null && size == \old(size) – 1;
 * @ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j]));
 * @assignable size,h[*];
 * @also
 * @*/

public Object delete (int key) {
    if (key >= 0) {
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key];
            h[key] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
```
Example

- Proof-based test case generation

```java
/*@ public normal_behaviour
@ requires key >= 0 ;
@ requires h[hash(key)] != null ;
@ requires size > 0 ;
@ ensures \result == \old(h[hash(key)]) ;
@ ensures h[hash(key)] == null && size == \old(size) – 1 ;
@ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
@ assignable size,h[*] ;
@ also
......
@*/
public Object delete (int key) {
    if (key >= 0) {
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key];
            h[key] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
Example

• Proof-based test case generation

```java
/*@ public normal_behavior
   @ requires key >= 0 ;
   @ requires h[hash(key)] != null ;
   @ requires size > 0 ;
   @ ensures \result == \old(h[hash(key)]) ;
   @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
   @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
   @ assignable size,h[*] ;
   @ also
   ..... 
*/
public Object delete (int key) {
    if (key >= 0) {
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key];
            h[key] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
```

`KeyTestGen` generates a (failing) test case which throws an index out of bound exception.
Example

- Proof-based test case generation

```java
/*@ public normalBehaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  ....
  @*/
public Object delete (int key) {
  if (key >= 0) {
    int i = hash(key);
    if (h[i] == null)
      return null;
    else {
      Object ret = h[i];
      h[i] = null;
      size = size - 1;
      return ret;
    }
  } else { return null; }
}
```

KeyTestGen generates a (failing) test case which throws an index out of bound exception.
- Define the model for your (control) property

\[
\text{Model Definition:} \quad \text{Model-Based Testing} \quad \text{Runtime Verification}
\]

\[
\text{Overall Implementation:} \quad \text{Model-Based Testing} \quad \text{Overall Implementation}
\]
Example

- Use MBT to develop the methods

```java
public void deposit(int money){
    if (u != null)
        u.getAccount().deposit(money);
}
```

/**
 * Deposits money in user's account.
 *
 * @param money amount of money to deposit
 *
 public void deposit(int money){
     if (u != null)
         u.getAccount().deposit(money);
 }
 */
Example

- Finish the overall implementation
  
  (i.e. implement method *main*)

```java
switch (inputLine) {
  case "deposit":
    System.out.print("Enter amount to deposit: ");
    amount = in.next();
    aux = Integer.parseInt(amount);
    f.deposit(aux);
    break;
  case "withdraw":
    System.out.print("Enter amount to withdraw: ");
    amount = in.next();
    aux = Integer.parseInt(amount);
    f.deposit(aux);
    break;
```
Example

- Finish the overall implementation

  *(i.e. implement method *main*)

```java
switch (inputLine) {
    case "deposit":
        System.out.print("Enter amount to deposit: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
    case "withdraw":
        System.out.print("Enter amount to withdraw: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
```
• Execute the monitor against MBT traces

```java
switch (inputLine) {
    case "deposit":
        System.out.print("Enter amount to deposit: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
    case "withdraw":
        System.out.print("Enter amount to withdraw: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
```
Example

- Use runtime verification to validate the model

```
\begin{align*}
\text{u} \neq \text{null} \land \text{deposit} \land \text{u} \neq \text{null} \\
\text{u} = \text{null} \land \text{login} \land \text{u} \neq \text{null} \\
\text{u} \neq \text{null} \land \text{logout} \land \text{u} = \text{null} \\
\text{u} \neq \text{null} \land \text{withdraw} \land \text{u} = \text{null}
\end{align*}
```
Example

- Execute the monitor against MBT traces

switch (inputLine) {
    case "deposit":
        System.out.print("Enter amount to deposit: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
    case "withdraw":
        System.out.print("Enter amount to withdraw: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.withdraw(aux);
        break;
}
Test Focus Development

- Test Driven Development
- Model-Based Testing
- Runtime Verification
- Control
- Data

- Contracts Definition
- Methods Signature Definition
- Deductive Verification
- Test Case Generation (symbolic execution)
- (partial) Proofs
- Test Cases

- Model Definition
- Monitor Generation
- Monitor Spec.
- Monitor Extension
- Model Translation
- Model-Based Testing
- Overall Implementation
- Runtime Verification
- Monitored Execution
- Monitored Execution

61
• **Part 3: Inferring Global Trace Conditions from Partial Local Proofs**
ppDATE Context Dependency

\[
\begin{array}{c}
\{\pi_1\} \text{ foo() } \{\pi'_1\} \\
\{\pi_2\} \text{ goo() } \{\pi'_2\}
\end{array}
\]

trigger | condition $\mapsto$ action

\[
\begin{array}{c}
\{\pi_1\} \text{ foo() } \{\pi''_1\}
\end{array}
\]
Context Independent Use of KeY

- Deductive Verification
- Specification Refinement
- Translation and Instrumentation
- Monitor Generation
Trace condition inference through backwards propagation of conditions
• Trace Pattern

public short start_from(Messages m2) { ... }

public short req(Messages m1) { ... }

public short ack(Messages m0) { ... }

• Trace Condition

m0 != null && m1 != m2 && status >= 0 • [(start_from,m2), (req,m1), (ack,m0)]
Trace Condition Inference

- m.label == 1 | start_from | true | skip
- transaction.balance <= balance | req | true | skip
- true | ack | true | skip
- status == 4 • [ ]
- status == 4
Trace Condition Inference

- m.label == 1 | start_from | true | skip

- transaction.balance <= balance | req | true | skip

- true | ack | true | skip

status == 4

{ true } ack { status == 4 }
Trace Condition Inference

- \( m\.label == 1 \mid start\_from \mid true \mid skip \)
- \( \text{transaction\.balance} \leq \text{balance} \mid \text{req} \mid true \mid skip \)
- \( \text{true} \mid \text{ack} \mid true \mid skip \)
- \( \text{status} == 4 \cdot [ ] \)
- \( \{ \text{true} \} \text{ ack } \{ \text{status} == 4 \} \)

### Branch Conditions

<table>
<thead>
<tr>
<th>Branch Conditions</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>status == 3</td>
<td>true</td>
</tr>
<tr>
<td>9 other branch conditions</td>
<td>false</td>
</tr>
</tbody>
</table>
Trace Condition Inference

- m.label == 1 | start_from | true | skip

- transaction.balance <= balance | req | true | skip

- status == 3

- true | ack | true | skip

- status == 3 • [(ack,m0) ]

- status == 4
Trace Condition Inference

- m.label == 1 | start_from | true | skip
- transaction.balance <= balance | req | true | skip
- \(\text{status} == 3\) → \{ \text{status} == 3 \} ack \{ \text{status} == 4 \}
- \text{status} == 3 • [(\text{ack, m0})]
- \text{status} == 4
Trace Condition Inference

- \( m\.label == 1 \mid \text{start\_from} \mid \text{true} \mid \text{skip} \)

- \( \text{transaction\.balance} \leq \text{balance} \mid \text{req} \mid \text{true} \mid \text{skip} \)

- \( \text{status} == 3 \)

- \( \text{true} \mid \text{ack} \mid \text{true} \mid \text{skip} \)

- \( \text{status} == 4 \)

- \( \text{status} == 3 \bullet [(\text{ack}, m0)] \)
Trace Condition Inference

\[
\begin{align*}
&\text{m.label == 1 | start_from | true | skip} \\
&\text{transaction.balance <= balance | req | true | skip} \\
&\text{status == 3} \\
&\text{true | ack | true | skip} \\
&\text{status == 4}
\end{align*}
\]

Branch Conditions

<table>
<thead>
<tr>
<th>Branch Conditions</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 branch conditions</td>
<td>false</td>
</tr>
</tbody>
</table>
Trace Condition Inference

- m.label == 1 | start_from | true | skip

- transaction.balance <= balance | req | true | skip

- status == 3

- true | ack | true | skip

- status == 3 • [(ack, m0)]

- true | ack | true | skip

- { transaction.balance <= balance } req { status == 3 }

### Branch Conditions

<table>
<thead>
<tr>
<th>Branch Conditions</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>status == 1</td>
<td>true</td>
</tr>
<tr>
<td>8 other branch conditions</td>
<td>false</td>
</tr>
</tbody>
</table>
Trace Condition Inference

\[
\text{status} == 1 \quad \text{[(req,m1),(ack,m0)]}
\]

\[
\text{m.label} == 1 \ | \ \text{start_from} \ | \ \text{true} \ | \ \text{skip}
\]

\[
\text{status} == 1
\]

\[
\text{transaction.balance} \leq \text{balance} \ | \ \text{req} \ | \ \text{true} \ | \ \text{skip}
\]

\[
\text{status} == 3
\]

\[
\text{true} \ | \ \text{ack} \ | \ \text{true} \ | \ \text{skip}
\]

\[
\text{status} == 4
\]
Trace Condition Inference

- \[ m.label == 1 \mid \text{start\_from} \mid \text{true} \mid \text{skip} \]

- \[ \text{status} == 1 \]
  - \[ \text{transaction.balance} <= \text{balance} \mid \text{req} \mid \text{true} \mid \text{skip} \]

- \[ \text{status} == 3 \]
  - \[ \text{true} \mid \text{ack} \mid \text{true} \mid \text{skip} \]

- \[ \text{status} == 4 \]

**Branch Conditions**

<table>
<thead>
<tr>
<th>Branch Conditions</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>status == 0</td>
<td>true</td>
</tr>
<tr>
<td>9 other branch conditions</td>
<td>false</td>
</tr>
</tbody>
</table>
Trace Condition Inference

- **status == 0**
  - \( m\.label == 1 \) | start_from | true | skip

- **status == 1**
  - transaction\.balance <= balance | req | true | skip

- **status == 3**
  - true | ack | true | skip

- **status == 4**

\( status == 0 \) • \([(start\_from,m2),(req,m1),(ack,m0)]\)
Applications of Trace Conditions

- Global (unit) test case generation
  - Test case generated in initial state
  - Data generated on intermediate steps too
- State invariants verification
  - Property is fulfilled when the state is reached, and while being in that state
  - Every execution of the system fulfills the invariant
- Guided (online) runtime verification
Conclusions

- ppDATE specification language
- StaRVOOrS: (Fully automatic) tool for combined deductive and runtime verification of Java
- Testing focused development methodology enhanced with deductive and runtime verification
- Global trace condition inference from local partial proofs methodology