Symbolic Verification of Regular Properties

Zhenbang Chen
(zbchen@nudt.edu.cn)

joint work with Hengbiao Yu, Ji Wang, Zhendong Su and Wei Dong

College of Computer, National University of Defense Technology, China
Department of Computer Science, University of California, Davis, USA
06/01/2018
Regular Property Verification

• Regular properties/FSMs are widely used
  • Model-based testing
  • Typestate analysis, e.g., runtime verification
  • API protocol specification and mining
• Verifying regular properties is challenging
Dynamic Symbolic Execution (PLDI’05)
Challenge of Symbolic Execution

- Path explosion problem

*How to boost completing path exploration and finding counterexample?*
Observation and Insight

• Many irrelevant paths exist
• For relevant paths
  • The ones with specific sequences can violate the regular property
• Many are equivalent w.r.t. verification
Observation and Insight

• Many irrelevant paths exist
• For relevant paths
  • The ones with specific sequences can violate the regular property
  • Many are equivalent w.r.t. verification

Prune irrelevant, uninteresting relevant and equivalent paths, and explore counter-example paths earlier
Key Idea

Verify a program satisfies a regular property $P$

$P$, $\neg P$

Pruning

Guiding

Regular property-oriented path slicing

Prune redundant paths

Regular property guided DSE [ICSE’15]

Find counterexamples earlier

Complement
Key Idea-Guiding w.r.t \( \neg P \)

**history** \( \cap \) **future** \( \neq \emptyset \)

Dynamic analysis

Static analysis
Key Idea-Pruning

\[ \text{history} \cap \text{future} = \emptyset \]

Equivalent to explored counter-example paths

No control or data dependence

WOODPECKER [ASPLOS’13]
Path Slicing [PLDI’05]

\[ \approx \]

\[ \text{history} \cap \text{future} = \emptyset \]
Key Idea-Pruning

init, close, read
init, read, close, read
init, read, read, close, read
init, close, read, read

Equivalent to explored counter-example paths
Sneak Preview of Results

• For 39 verification tasks (1 hour for each)
  • 30 are completed by our method
  • DFS (22), pure guiding (22) and slicing (23)
• For the completed verification tasks
  • >8.4X, >8.6X, and >7X time speedups over DFS, pure guiding and path slicing
Synergic Framework

(1) Static analysis

Regular Property (FSM)

(2) DSE

Running & Dynamic analysis

Input generation

Slicing the path

Finished?

Guided branch selection

Report results
An Example

```java
int foo(int m, int n, int[] a) {
    InputStreamReader w = new …;
    if (m > 50) m++;
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] > a[i+1]) {
            int temp = a[i];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }
    if (a[i] == 100)
        w.close();
    while (n-- > 0){
        int j = w.read();
        if (j == -1) break;
        m += j;
    }
    return m;
}
```

**Reader property**

**Cannot read after closed**

**The negation of the property**

**Read after closed**
```java
int foo(int m, int n, int[] a) {
    InputStreamReader w = new ...;
    if (m > 50) m++;
    for (int i = 0; i < a.length-1; i++) {
        if (a[i] > a[i+1]) {
            int temp = a[i];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }
    if (a[i] == 100)
        w.close();
    while (n-- > 0){
        int j = w.read();
        if (j == -1) break;
        m += j;
    }
    return m;
}
```
int foo(int m, int n, int[] a) {
    InputStreamReader w = new …;
    if (m > 50) m++;
    for (int i = 0; i < a.length-1; i++) {
        if (a[i] > a[i+1]) {
            int temp = a[i];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }
    if (a[i] == 100)
        w.close();
    while (n-- > 0){
        int j = w.read();
        if (j == -1) break;
        m += j;
    }
    return m;
}
int foo(int m, int n, int[] a) {
    InputStreamReader w = new ...;
    if (m > 50) m++;
    for (int i = 0; i < a.length-1; i++) {
        if (a[i] > a[i+1]) {
            int temp = a[i];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }
    if (a[i] == 100)
        w.close();
    while (n-- > 0) {
        int j = w.read();
        if (j == -1) break;
        m += j;
    }
    return m;
}
int foo(int m, int n, int[] a) {
    InputStreamReader w = new ...;
    if (m > 50) m++;
    for (int i = 0; i < a.length-1; i++) {
        if (a[i] > a[i+1]) {
            int temp = a[i];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }
    if (a[i] == 100)
        w.close();
    while (n-- > 0){
        int j = w.read();
        if (j == -1) break;
        m += j;
    }
    return m;
}
An Example

```java
int foo(int m, int n, int[] a) {
    InputStreamReader w = new …;
    if (m > 50) m++;
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] > a[i+1]) {
            int temp = a[i];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }
    if (a[i] == 100)
        w.close();
    while (n-- > 0){
        int j = w.read();
        if (j == -1) break;
        m += j;
    }
    return m;
}
```

**Reader property**

**Cannot read after closed**

Only 2 paths are needed to complete verification

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>Unfolding two loops</td>
</tr>
<tr>
<td>Guiding</td>
<td>2nd path, Unfolding two loops</td>
</tr>
<tr>
<td>Path Slicing</td>
<td>Only one branch is sliced</td>
</tr>
</tbody>
</table>
Implementation & Experiment Setup

- Implement for Java based on RGSE
- 16 real world open source Java programs
  - 270K LOC in total
<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhino-a</td>
<td>19799</td>
<td>Javascript interpreter</td>
</tr>
<tr>
<td>soot-c</td>
<td>32358</td>
<td>Static analysis tool</td>
</tr>
<tr>
<td>jlex</td>
<td>4400</td>
<td>Lexical analyzer</td>
</tr>
<tr>
<td>bloat</td>
<td>45375</td>
<td>Java bytecode optimization</td>
</tr>
<tr>
<td>bmpdecoder</td>
<td>531</td>
<td>BMP file decoder</td>
</tr>
<tr>
<td>ftpclient</td>
<td>2436</td>
<td>FTP client in Java</td>
</tr>
<tr>
<td>pobs</td>
<td>5488</td>
<td>Java parser objects</td>
</tr>
<tr>
<td>jpat</td>
<td>3245</td>
<td>Java string parser</td>
</tr>
<tr>
<td>jericho</td>
<td>25657</td>
<td>Jericho HTML Parser</td>
</tr>
<tr>
<td>nano-xml</td>
<td>3317</td>
<td>Non-validating XML parser</td>
</tr>
<tr>
<td>htmlparser</td>
<td>21830</td>
<td>HTML parser in Java</td>
</tr>
<tr>
<td>xml</td>
<td>5138</td>
<td>XML parser in Java</td>
</tr>
<tr>
<td>fastjson</td>
<td>20223</td>
<td>JSON library from alibaba</td>
</tr>
<tr>
<td>jep</td>
<td>42868</td>
<td>Mathematics library</td>
</tr>
<tr>
<td>udl</td>
<td>26896</td>
<td>UDL language library</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>259642</strong></td>
<td>15 open source programs</td>
</tr>
</tbody>
</table>
Implementation & Experiment Setup

• Implement for Java based on RGSE

• 16 real world open source Java programs
  • 270K LOC in total

• Properties
  • JDK’s single- and multi-objects typestate properties
  • User defined

• Verify each program/property in 1 hour
Within one hour, guiding and SRV can fail to detect a violation after thousands of iterations. To inspect the slicing can boost guiding by reducing the number of iterations counterexamples. For the 24 tasks with counterexamples found, 23 and 22 programs respectively, while DFS and pure path slicing can only a violation within one hour, while SRV needs only 102.6 seconds. When a violation is very deep and there possibly exist a large number of relevant paths, it cannot be detected without the finding in 7 (29%) tasks #iterations using our slicing is two orders of magnitude less than that using path slicing. Completed tasks by us but not by path slicing Cannot finish after 24 hours except jep

Results

• Use the fewest iterations to complete path exploration
• For 24 tasks with counterexamples, slicing can boost the finding in 7 (29%) tasks
• #iterations using our slicing is two orders of magnitude less than that using path slicing
• Completed tasks by us but not by path slicing
• Cannot finish after 24 hours except jep
G+S’ can complete the most number of tasks under a given time threshold
Conclusion

Regular Property Verification
- Regular properties/FSMs are widely used
- Model-based testing
- Typestate analysis, e.g., runtime verification
- API protocol specification and mining
- Verifying regular properties is difficult

Key Idea
Verify a program satisfies a regular property $P$

Synergic Framework

Sneak Preview of Results
- For 39 verification tasks (1 hour for each)
  - 30 are completed by our method
  - DFS (22), pure guiding (22) and slicing (23)
- For the completed verification tasks
  - >8.4X, >8.6X, and >7X time speedups over DFS, pure guiding and path slicing
Thank you
Any Questions?