EQUIVALENCE PARTITIONING AS A BASIS FOR DYNAMIC CONDITIONAL INVARIANT DETECTION

Worakarn Isaratham
Supervised by Dr Rosemary Monahan

July 2015
OUTLINE

1. Motivation
2. Background
3. Solution
4. Evaluation
5. Conclusion
Detect formal specification by observing dynamic test run
HOW CAN EQUIVALENCE PARTITIONING ASSIST DYNAMIC DETECTION OF CONDITIONAL INVARIANTS?
BACKGROUND
EQUIVALENCE PARTITIONING

• Divide input/output domains into “equivalence classes”, **conditional** to how data are processed

• All members of a class are processed in the same way, but different from other classes
DYNAMIC INVARIANT DETECTION

- Invariants – properties of programs that hold true for all executions

- Dynamic invariants detection – detect invariants by observing dynamic execution of the target program.
DAIKON INVARIANT DETECTOR
DAIKON INVARIANT DETECTOR

**Chicory**

**Daikon**
DAIKON INVARIANT DETECTOR

- Original Program
- Instrumented Program
- Instrument
- Run
- Data Trace Database
- Detect Invariants
- Test Suite
- Invariants

improvement?
equivalence
partitioning info
CONDITIONAL INVARIANTS

• Invariants in the implication form $p \implies q$

✦ Infeasible to compute exhaustively under dynamic invariant detection
YACON
YACON

Test suite → Extraction → Partitions → Translation → Splitting conditions

Worakarn Isaratham
EXTRACTION PHASE

Test suite → Extraction → Partitions → Translation → Splitting conditions

Yacon
EXTRACTION PHASE

• Boundary value recovery strategy
  ✦ Looking for boundary values in test data

• Test suite invariants recovery strategy
  ✦ Arguments passed to the same method from the same position should be in the same equivalence class

• Support user-defined strategies
BOUNDARY VALUE STRATEGY

1. Run Chicory (Daikon’s instrumenter for Java) to collect trace data

2. Analyse the trace data for adjacent values

3. Create interval-based partitions
BOUNDARY VALUE STRATEGY

12    # interval-based classes
13    PARTITION Sign: int
14    IntervalClasses Minima(0, 1)
15
16    PARTITION SignDouble: double
17    IntervalClasses Mixed(0.0{Min}, 0.0{Max})
18
19    # Mixed type interval-based classes
20    PARTITION Probability: float
21    IntervalClasses Mixed(0.0f{Min}, 1.0f{Max})
1. **Proxify** the test suite (rewrite source files by creating new method for each invocation on target classes)

```java
Target t = new Target();
int i = t.foo("abc", 0);
```

```java
Target t = new Target();
int i = ÝácônProxifier.proxify(t).m_foo_Target_42("abc", 0)
```
TEST SUITE INVARIANTS STRATEGY

2. Compile the proxified code

3. Run Daikon on proxified test suite

4. Transform Daikon’s result into predicate-based partitions
# predicate-based classes and complementary class

```
6 | # predicate-based classes and complementary class
7 | PARTITION SimplePartition: int
8 | EQClass $value == 0
9 | EQClass $value < 10 && $value >= 1
10 | ComplementClass
```
TRANSLATION PHASE
TRANSLATION PHASE

1. Read partitioning files

2. Convert partitions into splitting conditions
EVALUATION
EVALUATION

• Partitioning Recovery Effectiveness

• Invariants Discovery Effectiveness

• Performance (runtime)
RECOVERY EFFECTIVENESS

- Compare generated partitions against expected
- Measure "best-matched distance" of each domain

\[
d(C_1, C_2) = \sum_{i=1}^{n} \min_{j=1..m} \delta(S_i, S'_j) + \sum_{j=1}^{m} \min_{i=1..n} \delta(S'_j, S_i)
\]

\[
\delta(S_i, S_j) = \begin{cases} 
0, & \text{if } S_i = S_j \\
w_s, & \text{if } S_i \subset S_j \text{ or } S_j \subset S_i \\
1, & \text{otherwise}
\end{cases}
\]
RECOVERY EFFECTIVENESS

• Calculate overall effectiveness of the generated partitions

\[
\text{effectiveness} = 1 - 2 \cdot (\text{average best-matched distance of all domains})
\]
• Measure the effect of using Yacon by comparing generated invariants from
  (1) Daikon
  (2) Daikon + Yacon
  (3) Daikon + Expected Partition

• Measure quality metrics
INVariants DISCoverY EFFECTIVENESS

• Assess quality of each invariant
  ✦ Correctness – is it true for all conceivable inputs?
  ✦ Usefulness – can it help programmers in some ways?
  ✦ Relevance – is it a characteristic of this program?
INVARIANTS DISCOVERY EFFECTIVENESS

- Assess quality of partitioning
  - Correctness – correct invariants / reported invariants
  - Usefulness – useful invariants / reported invariants
  - Precision – relevant invariants / reported invariants
  - Recall – relevant invariants / expectation
INVARIANTS DISCOVERY EFFECTIVENESS

Precision

- **Baseline**
- **Yacon**
- **Reference**

<table>
<thead>
<tr>
<th>Component</th>
<th>Baseline</th>
<th>Yacon</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earthquake</td>
<td>40%</td>
<td>50%</td>
<td>60%</td>
</tr>
<tr>
<td>ComputeTax</td>
<td>10%</td>
<td>15%</td>
<td>20%</td>
</tr>
<tr>
<td>Insurance</td>
<td>20%</td>
<td>30%</td>
<td>40%</td>
</tr>
<tr>
<td>StackAr</td>
<td>30%</td>
<td>40%</td>
<td>50%</td>
</tr>
<tr>
<td>QueueAr</td>
<td>15%</td>
<td>20%</td>
<td>30%</td>
</tr>
<tr>
<td>BinaryHeap</td>
<td>50%</td>
<td>60%</td>
<td>70%</td>
</tr>
<tr>
<td>BinarySearchTree</td>
<td>30%</td>
<td>40%</td>
<td>50%</td>
</tr>
</tbody>
</table>

15 June 2015
INVARIANTS DISCOVERY EFFECTIVENESS

Recall

- Baseline
- Yacon
- Reference

% Recall

- Earthquake
- ComputeTax
- Insurance
- StackAr
- QueueAr
- BinaryHeap
- BinarySearchTree

Worakarn Isaratham
INVARIENTS DISCOVERY EFFECTIVENESS

Reported Invariants

Correctness
Usefulness
Precision
Recall

%
PERFORMANCE

Runtime (seconds)

Baseline

Yacon

Suite Invariants
Boundary Value (excluding Chicory)
Daikon
Chicory

13
10
17
6
29
10
THREATS TO VALIDITY

- Small sample size – 7 programs, 11 test suites
- Small programs – textbook programs less than 250 lines of code
- Selection bias – some programs are selected because they have desired characteristics
- Subjectivity of invariants assessment
- Test suite construction bias – test suites written to fit how Yacon works
CONCLUSION
CONCLUSION

- Information from equivalence partitioning can be effective in uncovering conditional invariants.

- Our recovery strategies work better together than individually.

- The recovery strategies are only effective in limited circumstances.

- The solution increases recall metric, at the expense of overall quality of generated invariants.
FUTURE WORK

• Find more effective partitioning recovery strategies.

• Automatic invariants assessment – to overcome the subjectivity in invariants evaluation and to work at larger scale.

• Compare Yacon against other ways of generating splitting conditions.

• Adapt Yacon to other invariant detectors.
THANK YOU
(BACKUP)
EQUIVALENCE PARTITIONING

“[it] is a technique that is intuitively used by virtually every tester we've ever met.”

“INVARİANTS”

Includes

• Class invariants
• Method preconditions
• Method postconditions
• NOT INCLUDE LOOP INVARIANTS
SPLITTING POLICIES

• Default
  • Procedure Return Analysis

• Shipped with Daikon
  • Static Analysis (for IF, FOR, WHILE statements)
  • Cluster Analysis
  • Random Sampling
## PROGRAM SIZE

<table>
<thead>
<tr>
<th>Program</th>
<th>Program Size</th>
<th>Selected Methods</th>
<th>Selected Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lines</td>
<td>methods</td>
<td></td>
</tr>
<tr>
<td>Earthquake</td>
<td>46</td>
<td>1</td>
<td>(all)</td>
</tr>
<tr>
<td>ComputeTax</td>
<td>244</td>
<td>1</td>
<td>(all)</td>
</tr>
<tr>
<td>Insurance</td>
<td>45</td>
<td>1</td>
<td>(all)</td>
</tr>
<tr>
<td>StackAr</td>
<td>117</td>
<td>9</td>
<td>(all)</td>
</tr>
<tr>
<td>QueueAr</td>
<td>122</td>
<td>9</td>
<td>enqueue (Object)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>percolateDown</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(int)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>find (T, BinaryNode&lt;T&gt;)</td>
</tr>
<tr>
<td>BinarySearchTree</td>
<td>245</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
PROGRAM SELECTION

• All are from textbooks

• StackAr, QueueAr are stack and queue data structures, often used as benchmark programs for Daikon

• BinaryHeap, BinarySearchTree

• Built-in test suites
PROGRAM SELECTION

• Earthquake, ComputeTax, Insurance

• Suitable structure for equivalence partitioning

• No built-in test suites
## TESTING

<table>
<thead>
<tr>
<th>Coverage Metric</th>
<th>Total</th>
<th>Covered</th>
<th>% coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions</td>
<td>8,482</td>
<td>6,862</td>
<td>80.9%</td>
</tr>
<tr>
<td>Lines</td>
<td>1,971</td>
<td>1,608</td>
<td>81.6%</td>
</tr>
<tr>
<td>Branches</td>
<td>788</td>
<td>507</td>
<td>64.3%</td>
</tr>
<tr>
<td>Methods</td>
<td>327</td>
<td>263</td>
<td>80.4%</td>
</tr>
<tr>
<td>Types</td>
<td>62</td>
<td>56</td>
<td>90.3%</td>
</tr>
<tr>
<td>Cyclomatic complexity</td>
<td>724</td>
<td>444</td>
<td>61.3%</td>
</tr>
</tbody>
</table>
Yacon

Daikon

Celeriac

Chicory