Automatic Testing and Fixing of Programs with Contracts

Yu Pei
Chair of Software Engineering
Dec. 5, 2012
Design by contract

- **Contracts**
  
  ```haskell
  LINKED_LIST . index_of (v: G; i: INTEGER_32): INTEGER_32
  -- Index of `i'-th occurrence of item identical to `v'.
  -- 0 if none.
  require
    positive_occurrences: i > 0
  ensure
    non_negative_result: Result >= 0
  ```

- **Applications**
  - Specification
  - Documentation
  - Testing & fixing
Automatic (random) testing

- Testing
  - Input
  - Oracle

- **AutoTest**: Automatic testing programs with contracts
  - Precondition of the routine under test as the valid input filter
  - Postcondition of the routine as the oracle
The select-prepare-test loop

Sample testing process

```plaintext
create [LINKED_LIST [INTEGER]] v1.make
v2 := 1
v1.extend (v2)
v1.wipe_out
v3 := 125
v4 := v1.has (v3)
v5 := v1.count
```

Object pool: v1, v3, v2, v4, v5
Performance evaluation

- Testing results
  - Precondition of the routine-under-test is violated
    - Invalid test case
  - Precondition of the routine-under-test is satisfied
    - Postcondition satisfied
      - Passing test case
    - Postcondition not established
      - Failing test case (detected fault)

- Evaluation criteria
  - Fault detection rate
  - Input space coverage
Random+ testing

- Essentials
  - Input generation
    - Primitive types: random selection + boundary values
    - Reference types: constructor calls + random selection
  - Diversification
    - With probability $p_{div}$ after each test

- Result
  - Find faults in widely used, industrial-grade code
  - High fault detection rate in the first a few minutes
Adaptive Random Testing

- Essentials
  - Maintain a list of objects $O$ used in testing a routine $r$
  - Select the object with the highest average distance to $O$ for the next test of $r$

- Result
  - Takes less time and generated tests, on average by a factor of 5, to the first fault
Testing with guided object selection

- Essentials
  - Keep track of precondition-satisfying objects
  - Use them with higher probability

- Results
  - 56% of the routines that cannot be tested before are now tested
  - 10% more faults detected in the same time
  - Routines tested 3.6 times more often
**Stateful testing**

- **Essentials**
  - Object states in Boolean expressions
    - before, after, is_empty, i > 0, ...
  - Infer preconditions from existing tests
    - Boolean expressions that always hold as preconditions
  - Prepare inputs violating the inferred preconditions
    - Select objects in the object pool
    - Transit objects using object behavioral model

- **Result**
  - 68% more faults detected with 7% time overhead
Automatic program fixing

Program with Contracts

Test Cases (AutoTest)

Fault localization

Fix Synthesis

Fix Validation

AutoFix

Valid Fixes

Program Test Cases (AutoTest)
Fault localization

- Model-based
  - Difference between state invariants from passing and failing runs as the fault profile

- Code-based
  - State components as candidate fault causes: <exp, loc, val>
  - Suspiciousness scores computed from
    - dyn: frequency of appearance in passing/failing runs
    - cdep: control dependence on the violation position
    - edep: syntactical similarity with the failing assertion
Example

```plaintext
move_item (v: G)
    -- from TWO_WAY_SORTED_SET.
    -- Move \`v\' to the left of cursor.
    require v /= Void ; has (v)
local idx: INTEGER ; found: BOOLEAN
    do
        idx := index
        from start until found or after loop
            found := (v = item)
            if not found then forth end
        end
        remove
        go_i_th (idx)
put_left (v) -- require: not before
end
```

- **Model-based**

<table>
<thead>
<tr>
<th>Location</th>
<th>Passing inv.</th>
<th>Failing inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-14</td>
<td>not is_empty</td>
<td>not is_empty</td>
</tr>
<tr>
<td></td>
<td>not before</td>
<td>before</td>
</tr>
<tr>
<td></td>
<td>not after</td>
<td>not after</td>
</tr>
</tbody>
</table>

- **Code-based**

<table>
<thead>
<tr>
<th>&lt;exp, loc, val&gt;</th>
<th>dyn</th>
<th>cdep</th>
<th>edep</th>
<th>susp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&lt;before, L-14, T&gt;</td>
<td>1.5</td>
<td>1</td>
<td>1</td>
<td>1.13</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&lt;not after, L-12, T&gt;</td>
<td>0.67</td>
<td>0.86</td>
<td>0.33</td>
<td>0.22</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Fix synthesis (1)

- Fixing actions: code necessary for correcting the faulty state (e.g. `before @ L-14`)
  - Enumeration
    - Identify objects that could be modified to affect the state
    - Enumerate all applicable operations on the objects
  - Object behavioral model
    - Actions: `forth`, ...

```
not is_empty before not after

forth

not is_empty not before not after
```
Fix synthesis (2)

- Fix schemas capture common fixing styles
  ```java
  if fail_condition then
    fixing_action
  else
    original_instruction
  end
  ```

- Fix schema instantiation
  ```java
  move_item (v: G)
  do
    idx := index
    ...
    remove
    go_i_th (idx)
  put_left (v)
  end
  ```
Fix validation and ranking

- **Validation**
  - Run the patched program against all passing and failing tests, requiring
    - Passing tests still pass
    - Failing tests now pass

- **Ranking**
  - Static metrics, favoring
    - Simple textual changes
    - Changes close to the failing location
    - Changes involving less original statements
  - Dynamic metric, favoring
    - Behavioral preservation
Experimental evaluation

- 204 randomly detected faults in various programs were used for evaluation

- 86 (or 42%) out of 204 faults got valid fixes

- 51 (or 59%) out of 86 faults got proper fixes

- 93% runs terminated within 15 minutes

- 48 (or 56%) of the faults that AutoFix managed to fix at least once were fixed in over 95% of the sessions
Summary

- Contracts promote automatic testing and fixing
  - **AutoTest**
    - Preconditions as input filters and postconditions as oracles
  - **AutoFix**
    - Contracts as guarantee of semantic correctness

- Project web pages:
  - [http://se.inf.ethz.ch/research/autotest/](http://se.inf.ethz.ch/research/autotest/)
  - [http://se.inf.ethz.ch/research/autofix/](http://se.inf.ethz.ch/research/autofix/)
THANKS