Automated Fixing of Programs with Contracts

Yi Wei, Yu Pei, Carlo A. Furia, Lucas S. Silva, Stefan Buchholz, Bertrand Meyer, Andreas Zeller

Presented by Christine Zeller
Motivation

- Programming is not just about writing code
  - Find errors
  - Fix errors
- Automating these steps is helpful
  - Automatic testing tools help finding errors
  - What about fixing them?
Background

- **AutoTest**
  - Automated Testing Framework
  - Paper will be presented in this seminar

- **Pachika**
  - V.Dallmeier, A. Zeller, B.Meyer (2009)
  - Tool to generate potential fixes for bugs
  - Used with failing testcases for Java Programs
AutoFix-E

- Find fixes using
  - Contracts
  - Boolean Query Abstraction

- Plan:
  1) Assess Object State
  2) Construct Fault Profile and Behavioral Model
  3) Generate Candidate Fixes
  4) Validate Fixes
Example

TWO_WAY_SORTED_SET
duplicate(n: INTEGER):like Current

local
  pos: CURSOR
  counter: INTEGER

do
  pos := cursor
  Result := new_chain
  Result.finish
  Result.forth
  from
  until
    (counter = n) or else after
  loop
    Result.put_left(item)
    forth
    counter := counter + 1
  end
  counter := counter + 1
end

go_to(pos)

end

\textit{item} has precondition \textbf{not before} and not after
Workflow

- Eiffel Class
- Test Suite
- Behavioral Model
- Fault Profile
- Candidate Fixes
- Valid Fixes

AutoTest

AutoFix-E
Object State

- Predicate set $P$
  - Boolean queries
  - Complex predicates (implications)
    
    $\text{is\_empty} \rightarrow \text{after}$

- Mutations of complex predicates
  
  \[
  \begin{align*}
  \neg A & \rightarrow B \\
  A & \rightarrow \neg B \\
  \neg A & \rightarrow \neg B
  \end{align*}
  \]

- Collection $\Pi = P \cup \{\neg p \mid p \in P\}$
- Remove redundancies in $P$ using Z3
Fault Profile

- State invariant
  \[ I_\ell = \{p | p \in \Pi \land p \text{ holds at location } \ell \} \]

- Consider all passing runs
  - Infer state invariant \( I_\ell^+ \) for each location \( \ell \)

- Consider all failing runs
  - Infer state invariant \( I_\ell^- \) for each location \( \ell \)
  - Only up to location of failure
Fault Profile: Example

- Construct fault profile
  \[ \Phi_\ell = \{ p \mid p \in I_\ell^+ \land p \notin I_\ell^- \} \]

- Use tool called Daikon

- Example:
  
  before and off  hold only in failing runs
  
  before implies not off  Fault Profile
Behavioral Model

- Finite-state automaton representing class’ behaviour

- Extract model from passing runs

- Idea

```
Failed state   Failed state   Fixed state
```

```
pre state          routine m          post state
```
Behavioral Model: Example

is_empty
before
not after


not is_empty
before
not after

forth

is_empty
not before
after

forth

not is_empty
not before
not after
Candidate Fixes

- Put everything together
- Predefined templates:

(a) snippet
old_stmt

(b) if fail then
    snippet
end
old_stmt

(c) if not fail then
    old_stmt
end

(d) if fail then
    snippet
else
    old_stmt
end
Candidate Fixes: Example

duplicate(n: INTEGER): like Current

... from

  until
  
    (counter = n) or else after

  loop
    Result.put_left(item)
    forth
    counter := counter + 1
  end
  counter := counter + 1

  go_to(pos)

end
Candidate Fixes: Example

duplicate\(n: \text{INTEGER}\): like Current

... 
from

until

\((\text{counter} = n) \text{ or else after}\)

loop

\begin{verbatim}
if before then
    forth
else
    Result.put_left(item)
    forth
    counter := counter + 1
end
\end{verbatim}

go_to(pos)
end
Fix Validation

- Run all testcases on fixes
  - A fix is valid if it passes all failing and passing runs

- Additionally: Ranking
  - Static metrics
    - Textual change
    - Branches introduced
  - Dynamic metrics
    - Runtime behaviour
Improvement

- Linearly constrained assertions
  - E.g. \( i > 1 \) and \( i < \text{count} \)

- Require special techniques for fix generation

- Specific schema for candidate fixes

```python
if not constraint then new_stmt else old_stmt end
```
Experimental Evaluation

- 42 Faults from EiffelBase and Gobo

<table>
<thead>
<tr>
<th>Type of fault</th>
<th># Faults</th>
<th># Fixed</th>
<th># Proper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>24</td>
<td>11 (46%)</td>
<td>11 (46%)</td>
</tr>
<tr>
<td>Postcondition</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Check</td>
<td>1</td>
<td>1(100%)</td>
<td>0</td>
</tr>
<tr>
<td>Class invariant</td>
<td>9</td>
<td>4 (44%)</td>
<td>2 (22%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>42</strong></td>
<td><strong>16 (38%)</strong></td>
<td><strong>13 (30%)</strong></td>
</tr>
</tbody>
</table>

- Average fixing time: 2.6 minutes
- Small study with programmers
  - 4 of 6 proposed valid fixes were same as programmers’
Future Work

- Improve behavior model
- Different fault types
- Find faults in contracts
- Languages without contracts
- Improving ranking metric
- ...

18
Conclusion

- Limitation: all classes used data structure related
- Status from 2010
  - New Version of AutoFix developed in 2011
  - Different approach: code-based instead of model-based
- Still an open field of research