Automatic Fixing of Programs with Contracts

Yu Pei

Chair of Software Engineering, ETH Zurich
December 2, 2015
To Err Is Human

A French rail company has ordered 2000 new trains that are too big for 1300 stations they are due to serve.
Programs Have Faults

- Specification vs. Implementation
  - What a program should do vs. what a program really does
  - When they conform, the program is correct.

- Program faults are discrepancies between the two
  - Unpleasant, unacceptable, or even fatal
  - Expensive
  - Overwhelming to fix manually
Automatic Unit Testing

- **Unit testing**
  - Input: Test: `s.push(o)`
  - Oracle:

- **AutoTest**
  - Automatic test case generation
    - Precondition of the routine as the input filter
    - Postcondition of the routine as the oracle
  - Test case categorization

<table>
<thead>
<tr>
<th>push(element: E)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>require</code></td>
</tr>
<tr>
<td><code>element /= Void</code></td>
</tr>
<tr>
<td><code>ensure</code></td>
</tr>
<tr>
<td><code>count = old count + 1</code></td>
</tr>
<tr>
<td><code>top() = element</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition</th>
<th>Postcondition</th>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>--</td>
<td>Invalid</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>Valid, Passing</td>
</tr>
<tr>
<td>✓</td>
<td>×</td>
<td>Valid, Failing</td>
</tr>
</tbody>
</table>
Outline

Program with Contracts

AutoFix

178 citations in total
An Example Fault

class CIRCULAR [G]
  duplicate (m: INTEGER): CIRCULAR [G]
    -- A duplicate with at most ‘m’
    -- elements copied from ‘Current’.
    require m >= 0
    do
      create Result.make (count)
      ...
    end

make (n: INTEGER)
  -- Initialize ‘Current’ for
  -- ‘n’ elements.
  require n >= 1
  do
    create list.make_list (n)
  end

list: ARRAYED_LIST [G]    -- Storage
count: INTEGER            -- Length of circular
...
An Example Fault

class CIRCULAR [G]
duplicate (m: INTEGER): CIRCULAR [G]
  -- A duplicate with at most ‘m’
  -- elements copied from ‘Current’.
  require m >= 0
  do
    create Result.make (count)
    ...
  end

make (n: INTEGER)
  -- Initialize ‘Current’ for
  -- ‘n’ elements.
  require n >= 1
  do
    create list.make_list (n)
  end

list: ARRAYED_LIST [G]  -- Storage
count: INTEGER  -- Length of circular
...

duplicate (m: INTEGER): ...
  do  -- fix implementation
    if count = 0 then
      create Result.make (1)
    else
      create Result.make (count)
    end
  end
...

duplicate (m: INTEGER): ...
  require  -- strengthen
    count > 0  -- precondition
  m >= 0

make (n: INTEGER)
  require  -- weaken
    n >= 0  -- precondition
The AutoFix Tool

fully automatic and easily accessible
Fixing the Implementation with ImpleFix

- Assumption
  - Contracts are correct

- Target faults
  - Incorrect source states of object transitions as causes
  - Simple changes as fixes

- Three steps
  - Fault localization
  - Fix synthesis
  - Fix validation
Fixing the Implementation with IpleFix

- Abstract execution traces using state snapshots: 
  \[ e, l, v \]
- Compute suspiciousness scores of the snapshots using multiple metrics
- Consider the most suspicious snapshots as potential fault causes

<table>
<thead>
<tr>
<th>L5. create Result.make (count)</th>
</tr>
</thead>
<tbody>
<tr>
<td>([m \geq 0, \ L5, \ True])</td>
</tr>
<tr>
<td>([\text{count=0}, \ L5, \ True])</td>
</tr>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>1.3</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Fixing the Implementation with ImlpeFix

- Construct fix actions to change the snapshot states
  - call remove
  - replace **count** with **count + 1**
- Instantiate candidate fixes from schemas using fix actions and suspicious snapshots

```java
if count = 0 then
    remove
end
create Result.make (count)
```

```java
if count = 0 then
    create Result.make (count + 1)
else
    create Result.make (count)
end
```
Fixing the Implementation with IpleFix

- Apply each fix to the program and re-execute all the tests
- Mark as valid the fixes that, when applied, make all the tests pass
- Report the first $n$ valid fixes to the user

```python
if count = 0 then
    remove
end
create Result.make (count)
```

```python
if count = 0 then
    create Result.make (1)
else
    create Result.make (count)
end
```
Experimental Evaluation of ImpleFix

- To understand the behavior of ImpleFix and the quality of generated fixes

- Experimental setup
  - AutoTest for fault detection and test preparation
  - 204 faults from 4 different code bases
  - 9 different settings of testing time for each fault
  - 30 repetitions for each fault and setting
Evaluation Results of ImpleFix

- How many faults can ImpleFix fix?
  - Valid fixes to 86 faults (42%)

- What is the quality of the fixes produced by ImpleFix?
  - Proper fixes to 51 faults (25%)

- What is the cost of fixing faults with ImpleFix?
  - On average ≤20 minutes per valid fix, including the time required for test generation

- How robust is ImpleFix’s performance?
  - 48 (56%) of the faults that ImpleFix managed to fix at least once were fixed in over 95% of the sessions
Correcting the Specification with SpeciFix

- Assumption
  - The implementation is correct

- Goal of fixing
  - Successful executions should be allowed
  - Unsuccessful executions should be forbidden

- Four steps
  - Contract weakening
  - Contract strengthening
  - Fix validation
  - Fix ranking
Correcting the Specification with SpeciFix

- Infer the set $\Omega$ of the weakest preconditions for \texttt{make}.
- Weaken the precondition of \texttt{make}: $P_{\texttt{make}}$ or $\omega$.

\begin{itemize}
  \item $\Omega = \{n \geq 0, \ldots\}$
  \item \texttt{make} requires $n \geq 1$ or $n \geq 0$
  \item require ... \\
\end{itemize}

![Diagram showing contract weakening and strengthening stages]

- Contract weakening
- Contract strengthening
- Fix validation
- Fix ranking

Duplicate \texttt{make} or \texttt{require} \texttt{n} $\geq 1$ or \texttt{n} $\geq 0$. 

\texttt{req: m} $\geq 0$ \\
\texttt{...} \\
\texttt{req: n} $\geq 1$ \\
\texttt{...}
Correcting the Specification with SpeciFix

- Infer the set $\Sigma$ of preconditions for \texttt{duplicate}
- Strengthen the precondition of \texttt{duplicate}: $P_{\texttt{duplicate}} \text{ and } \sigma$

\[ \Sigma = \{ \text{count } \neq 0, \ldots \} \]

\texttt{duplicate}:
\begin{itemize}
  \item require $m \geq 0$ and $\text{count } \neq 0$
  \item require $\ldots$
\end{itemize}

\texttt{make}:
\begin{itemize}
  \item require $n \geq 1$
  \item require $\ldots$
\end{itemize}
Correcting the Specification with SpeciFix

- Apply each fix to the program and re-execute the tests
- Mark as valid the fixes that, when applied, make the tests either passing or invalid
- Use more tests for validation than for fix generation

<table>
<thead>
<tr>
<th>contract weakening</th>
<th>contract strengthening</th>
<th>fix validation</th>
<th>fix ranking</th>
</tr>
</thead>
</table>

- make
  - $\text{require } n \geq 1 \text{ or } n \geq 0$
  - $\text{require } \ldots$
- duplicate
  - $\text{require } m \geq 0 \text{ and } \text{count } \neq 0$
  - $\text{require } \ldots$
Correcting the Specification with SpeciFix

- Prefer fixes resulting in weaker contracts, or more passing tests

1. make
   require \( n \geq 1 \) or \( n \geq 0 \)

2. duplicate
   require \( m \geq 0 \) and count /= 0

...
Experimental Evaluation of SpeciFix

- Experimental subjects
  - 44 faults from 10 standard library classes

- Result
  - Valid fixes to 42 faults, and proper fixes to 11
  - On average, 3 minutes for fixing and 31 minutes for testing per fix

- When both available, proper fixes to contracts are often preferable to proper fixes that change the implementation
Program with Contracts

AutoFix Tool
[ICSE’15]

ImpleFix
[ISSTA ’10, ASE ’11, TSE’14]

SpeciFix
[FASE ’14]

Program with Contracts
Thank you!