Automatic Testing and Fixing of Programs with Contracts

Yu Pei
Chair of Software Engineering
Dec. 4, 2013
Design by contract

- Contracts in Eiffel

```eiffel
ARRAYED_CIRCULAR.go_i_th (i: INTEGER)
-- Move cursor to `i'-th position.
require
  valid_cursor_index (i)
ensure
  index = i
```

- Applications
  - Specification
  - Documentation
  - Testing & fixing
Automatic program testing

- Test case
  - 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 cycle

Example testing process

*create* \{ARRAYED_CIRCULAR [INTEGER]\} v1.make

```
v2 := 1
v1.extend (v2)
v1.wipe_out
v3 := 125
v4 := v1.has (v3)
v5 := v1.count
```

v2  v4
v1
v3  v5
object pool
Performance evaluation

- Testing results
  - Precondition of the routine-under-test is violated
    - Invalid test case
  - Precondition of the routine-under-test is satisfied
    - Successful termination with postcondition satisfied
      - Passing test case
    - Premature termination or postcondition violation
      - Failing test case (detected fault)

- Evaluation criteria
  - Number of faults detected
  - Code coverage
Random+ testing

- Essentials
  - Input generation
    - Primitive types: random selection + boundary values
    - Reference types: random selection + constructor calls
  - Diversification
    - With probability $p_{\text{div}}$ 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 (ART)

- Essentials
  - Maintain a list of objects $O$ used in testing a routine $r$
  - Select the objects 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

ARRAYED_CIRCULAR.swap (i: INTEGER)
  -- Exchange item at `i'-th position
  -- with item at cursor position.
  
  require not off valid_index (i)

❖ 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
  - Input space and object states in Boolean expressions
    - before, after, is_empty, i > 0, ...
  - Infer preconditions from existing tests
    - Boolean expressions that always hold
  - 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
class ARRAYED_CIRCULAR

duplicate (n: INTEGER): like Current

--- Copy of sub-chain beginning at current
--- position and having min (`n', count) items.

require n >= 0

do
  create Result.make (count)
  ...
end

make (n: INTEGER)

--- Create a circular with `n' items.

require n >= 1

do
  create list.make (n)
end

....
Program faults and automatic fixing

- Program faults are discrepancies between the contracts (specification) and the implementation

- Automatic fixing
  - AutoFix: assuming contracts, fixing implementation
  - SpecFix: assuming implementation, fixing contracts
AutoFix: fault localization

- State snapshots as candidate fault causes:
  <expression, location, value>

- Compute suspiciousness scores based on heuristics:
  A state snapshot is more suspicious, if it
  - Appears more often in failing runs than in passing runs
  - Is closer to the violation position in the control flow graph
  - Is syntactically more similar to the failing assertion

```
<count = 0, loc, True>
<is_empty, loc, True>
...
```
AutoFix: fix synthesis

- Fix actions: code necessary for changing the faulty state snapshot
  - Identify relevant objects and generate actions to either modify them or replace them with others objects

\[
\text{replace `count' with `count + 1'}
\]

- Fix schemas: common styles of wiring the fix actions into the feature body

\[
\begin{align*}
\textbf{if} \hspace{1em} & \text{count} = 0 \hspace{1em} \textbf{then} \\
& \hspace{1em} \textbf{create} \hspace{1em} \text{Result.make (count + 1)} \\
\textbf{else} \\
& \hspace{1em} \textbf{create} \hspace{1em} \text{Result.make (count)} \\
\textbf{end}
\end{align*}
\]
AutoFix: fix validation and ranking

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

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

- Possible contract faults

- `make_pre` being too strong $\Rightarrow$ contract weakening
- Preconditions of all open features on the stack being too weak $\Rightarrow$ contract strengthening
SpecFix: fix validation and ranking

- Validation
  - Valid fixes should
    - Turn originally failing tests to either passing or invalid tests
    - Leave originally passing tests as still passing
  - Use more tests for validation than for fix generation to overcome overfitting

- Ranking
  - Prefers fixes resulting in more passing tests, or with weaker contracts
Experimental evaluation

- **AutoFix**: 204 randomly detected faults in various programs were used for evaluation
  - 86 (or 42%) faults got valid fixes
  - 51 (or 25%) faults got proper fixes

- **SpecFix**: 44 faults from real-life Eiffel libraries
  - 11 (or 25%) faults got proper fixes
  - Most of them are preferred by programmers to fixes that change the implementation
Summary

- Contracts are specifications in the form of executable code
  - **AutoTest**
    - Detects discrepancies between the implementation and the contracts
  - **AutoFix**
    - Corrects the implementation according to the contracts
  - **SpecFix**
    - Adjusts the contracts to reflect the implementation
To find out more

http://se.inf.ethz.ch/research/autotest/

http://se.inf.ethz.ch/research/autofix/

http://se.inf.ethz.ch/research/specfix/
THANKS