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
  - B. Meyer, A. Fiva, I. Ciupa, A. Leitner, Y. Wei, E. Stapf (2009)
  - 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

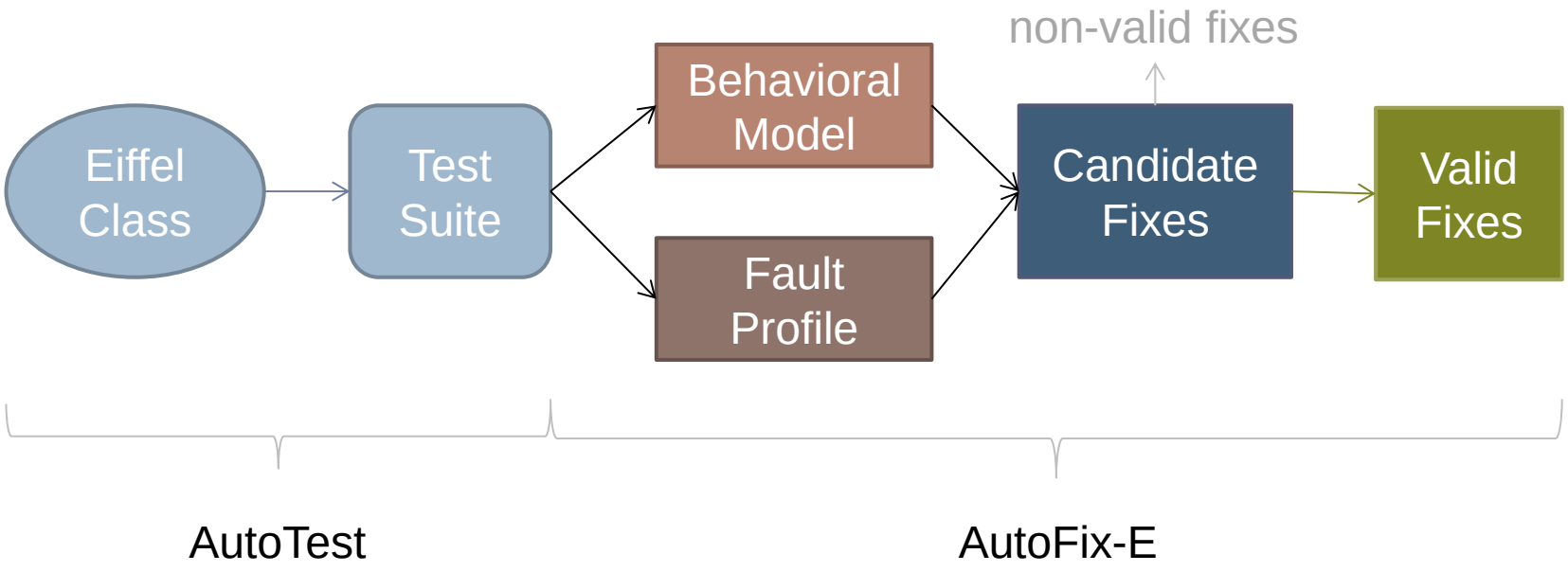
Example

- 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
            go_to(pos)
        end
```

item has precondition **not before** and not after

Workflow

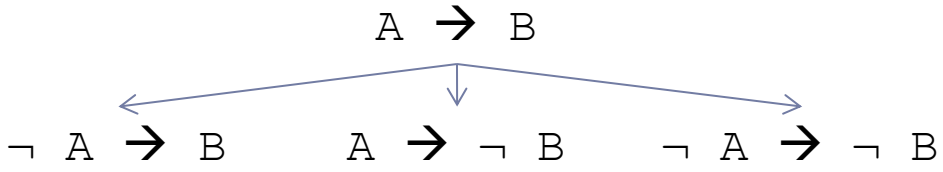

non-valid fixes
Eiffel Class
Test Suite
Behavioral Model
Fault Profile
Candidate Fixes
Valid Fixes
AutoTest
AutoFix-E

Object State

- Predicate set P
  - Boolean queries
  - Complex predicates (implications)

```
is_empty → after
```

  - Mutations of complex predicates



- Collection $\Pi = P \cup \{\text{not } p \mid p \in P\}$
- Remove redundancies in P using Z3

Fault Profile

- State invariant

$$I_\ell = \{p | p \in \Pi \wedge p \; holds \; at \; location \; \ell\}$$

- Consider all passing runs
  - Infer state invariant I_ℓ^+ for each location ℓ
- Consider all failing runs
  - Infer state invariant I_ℓ^- for each location ℓ
  - Only up to location of failure

Fault Profile: Example

- Construct fault profile

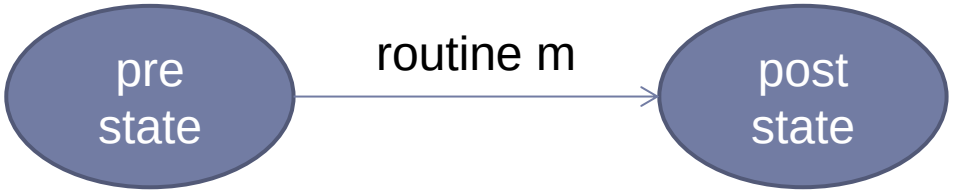
$$\Phi_\ell = \left\{ p \,\middle|\, p \in I_\ell^{+} \wedge p \notin I_\ell^{-} \right\}$$

  - Use tool called Daikon

- Example:

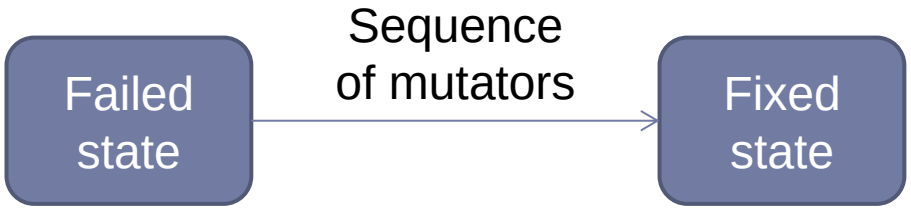
Behavioral Model

- Finite-state automaton representing class' behaviour

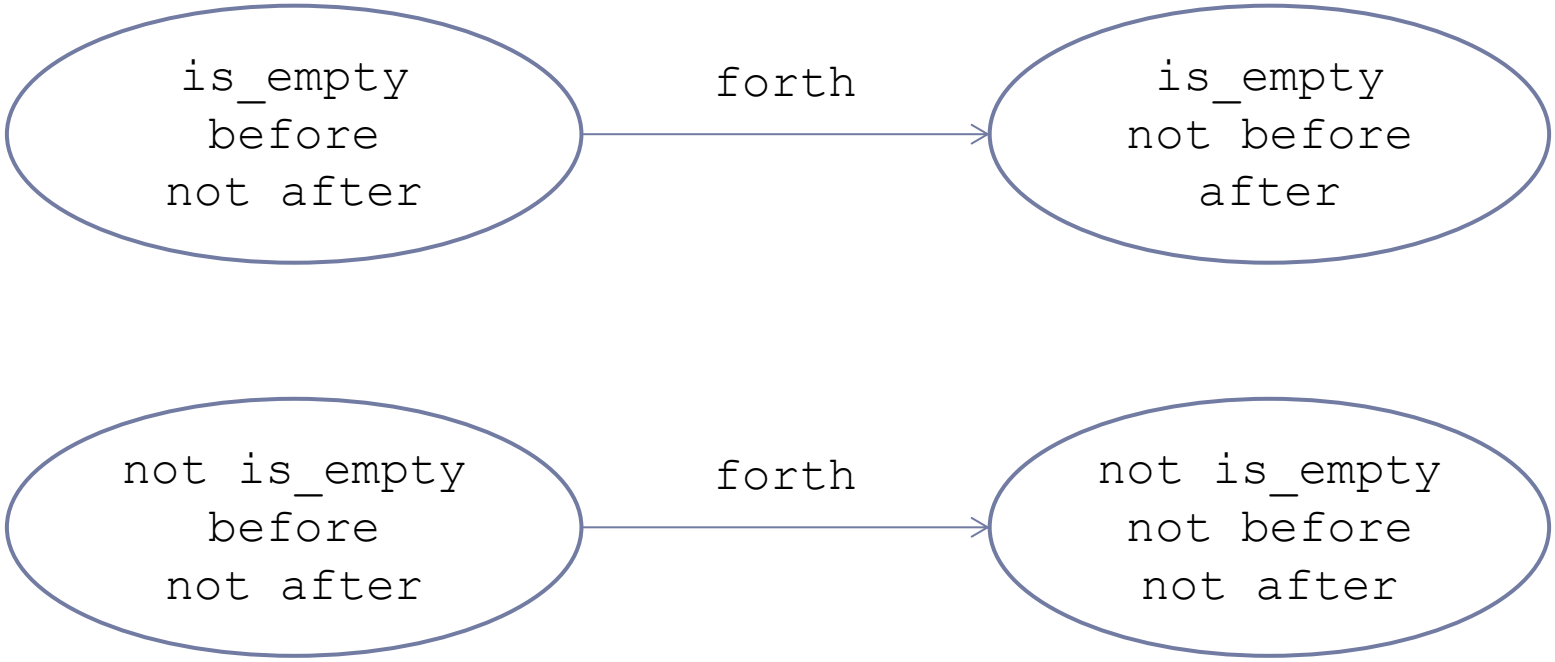


- Extract model from passing runs

- Idea



Behavioral Model: Example



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
        go_to(pos)
    end
```

Candidate Fixes: Example

```
duplicate(n: INTEGER):like Current
    ...
    from
        until
            (counter = n) or else after
        loop
            if before then
                forth
            else
                Result.put_left(item)
                forth
                counter := counter + 1
            end
        end
        go_to(pos)
    end
```

snippet

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 < count
```

  - Require special techniques for fix generation
  - Specific schema for candidate fixes

```
if not constraint then new_stmt else old_stmt end
```

Experimental Evaluation

- 42 Faults from EiffelBase and Gobo

Type of fault	# Faults	# Fixed	# Proper
Precondition	24	11 (46%)	11 (46%)
Postcondition	8	0	0
Check	1	1(100%)	0
Class invariant	9	4 (44%)	2 (22%)
Total	**42**	**16(38%)**	**13(30%)**

- 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
- ...

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