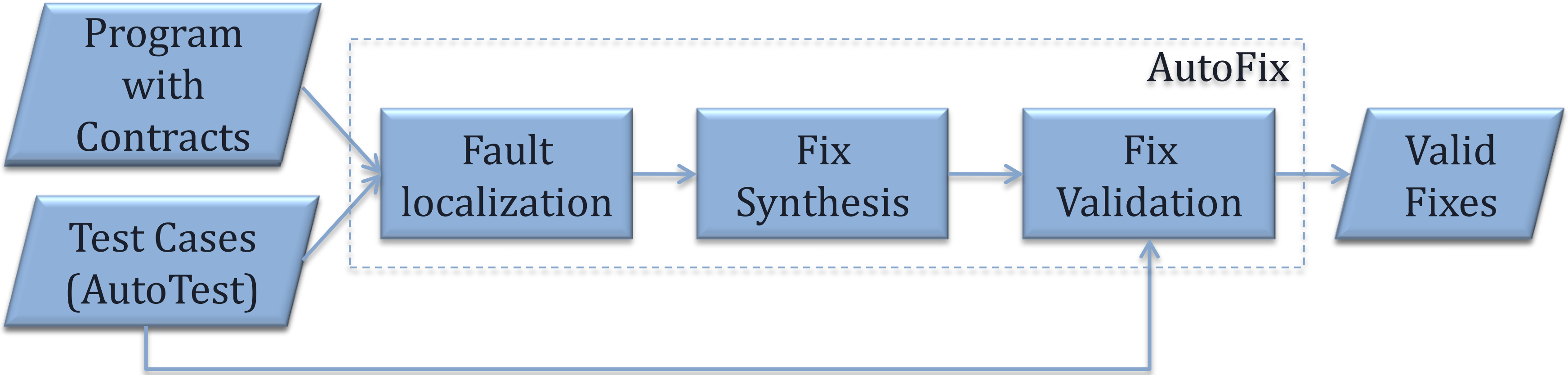
Automated Fixing

# of Programs with Contracts

Yu Pei, Yi Wei, Carlo A. Furia,
Martin Nordio, Bertrand Meyer

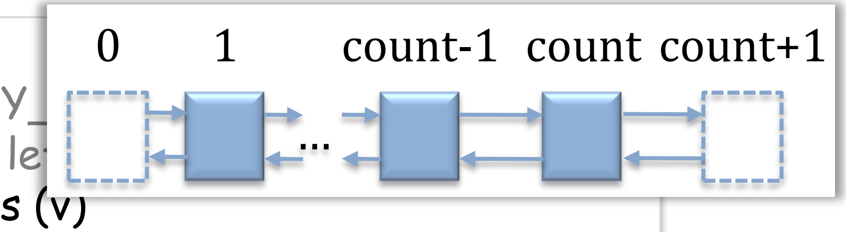
Overview


Program with Contracts
Test Cases (AutoTest)
AutoFix
Fault localization
Fix Synthesis
Fix Validation
Valid Fixes

Model-based Fault Localization

- Dynamic analysis
  - Difference between ***state invariants*** from passing and failing runs as the fault profile
  - State invariants in argument-less boolean queries

```
move_item (v: G)
        -- from TWO_WAY_
        -- Move `v' to the le
    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)
    end
```



Invar. from passing

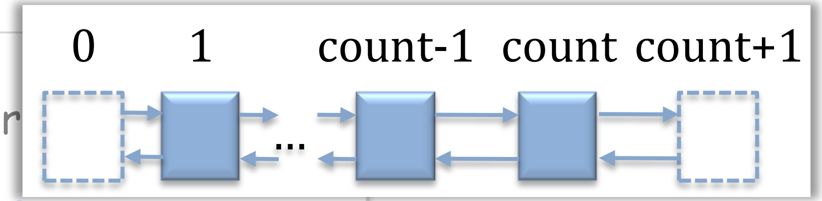
not is_empty
not before
not after
…

Invar. from failing

not is_empty
before
not after
…

Code-based Fault Localization

- Dynamic + Static Analysis
  - ***State components*** as candidate fault causes: <exp, loc, val>
  - Suspiciousness scores computed from
    - frequency of appearance in passing/failing runs
    - control dependence on the violation position
    - syntactical similarity with the failing assertion



```
1: move_item (v: G)
2:         -- Move `v' to the left of cursor
3:     require v /= Void ; has (v)
4:     local idx: INTEGER ; found: BOOLEAN
5:     do
6:         idx := index
7:         from start until found or after loop
8:             found := (v = item)
9:             if not found then forth end
10:        end
11:        remove
12:        go_i_th (idx)  -- <valid_index(idx), L-12, False>
13:        put_left (v)
14:    end
```

<index < idx, L-12, True> identified as a highly likely cause for fault.

Fixing Actions

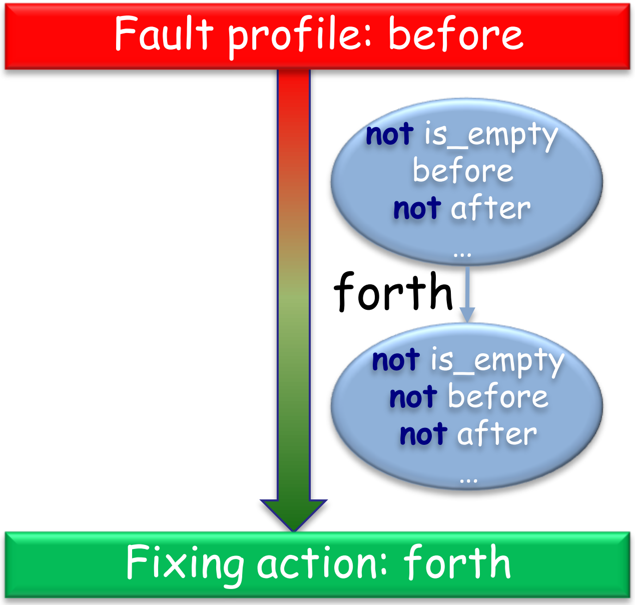
- Results of fault localization
  - Model-based: a list of <loc, inv>
  - Code-based: an ordered list of <exp, loc, val>
    - Order defined by the suspiciousness scores

- Fixing actions: code necessary for correcting the faulty state
  - Object behavioral model
  - Enumeration

Fixing Using Object Behavioral Model (OBM)

- Object behavioral model
  - Suggests how routines change object states
  - Contains a set of possible transitions
  - Can be learned from passing test executions

```
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)
    end
```



Fixing by Enumeration

- Identify values that could be modified to affect the state
- Enumerate all applicable operations on the values
  - Fixing with state modification

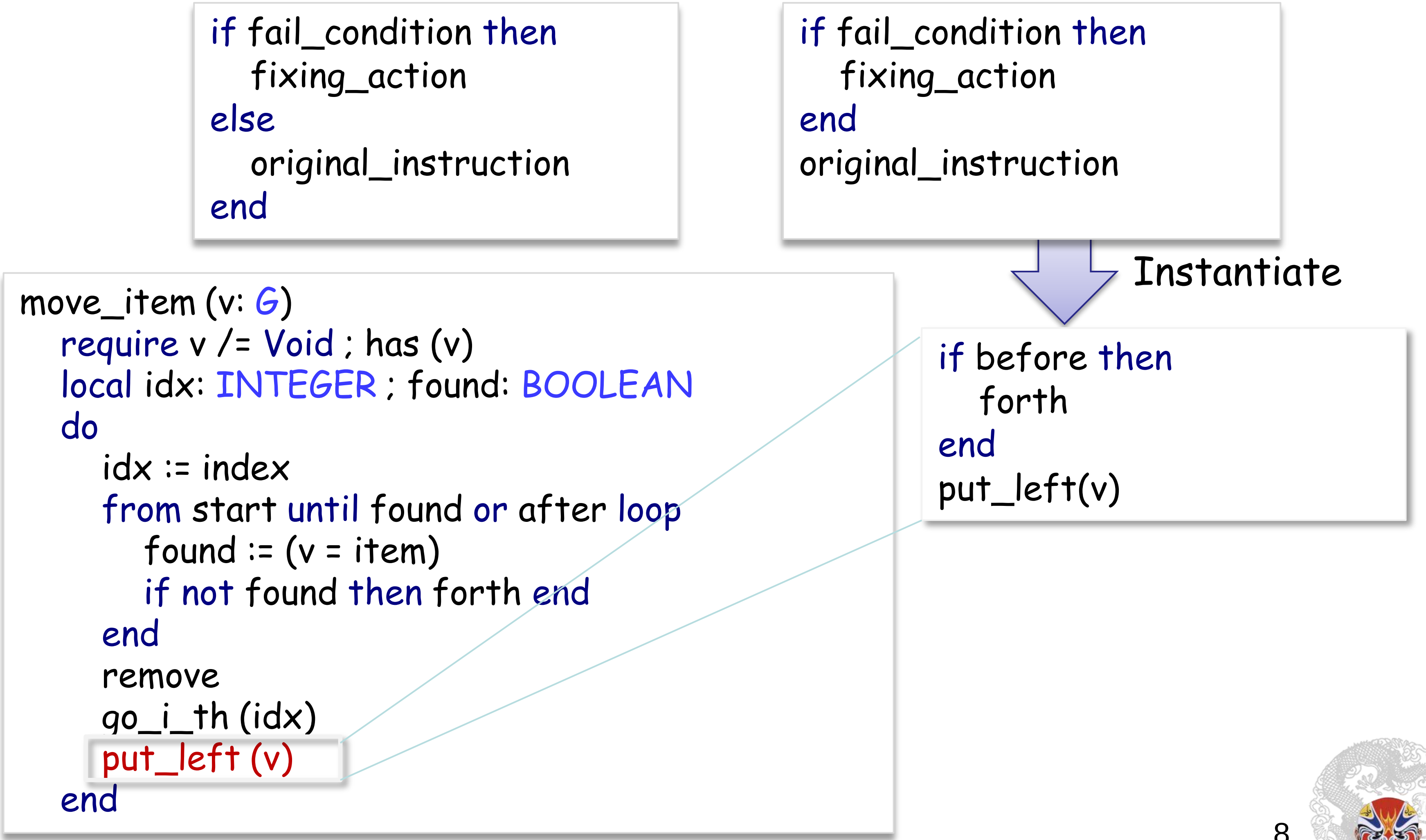
```
    -- <index<idx, L-12, True>
go_i_th (idx)          idx := idx – 1
```

  - Fixing with expression substitution

```
    -- <index<idx, L-12, True>
go_i_th (idx)          go_i_th (idx - 1)
```

Fix Synthesis

- Fix schemas capture common fixing styles.



```
if fail_condition then
    fixing_action
else
    original_instruction
end
```

```
if fail_condition then
    fixing_action
end
original_instruction
```

```
if before then
    forth
end
put_left(v)
```

```
move_item (v: G)
    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)
    end
```

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 favors
    - simple textual changes
    - changes close to the failing location
    - changes involving less original statements
  - Dynamic metric favors
    - behavioral preservation, i.e. passing tests should terminate in similar resulting states

Experimental Results

- Model-based fault localization + fixing actions from OBM
  - 42 faults from EiffelBase & Gobo: fixed 16 (38%)
  - In a small user study, 4 out of 6 of the selected fixes are the same as those from programmers
- Code-based fault localization + fixing actions by enumeration
  - 64 faults from EiffelBase & Gobo: fixed 14 (22%)
  - 9 faults from a student project: fixed 5 (55%)

❖Results considering only proper fixes.

❖Average fixing time is a few minutes per fault

Summary

- A fully automated approach to program fixing, which
  - works with program with contracts,
  - takes (passing and failing) test cases as inputs,
  - exploits dynamic and static analysis techniques,
  - validates candidate fixes through regression, and
  - succeeds in proposing proper fixes to real program faults.

- AutoFix

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

Questions