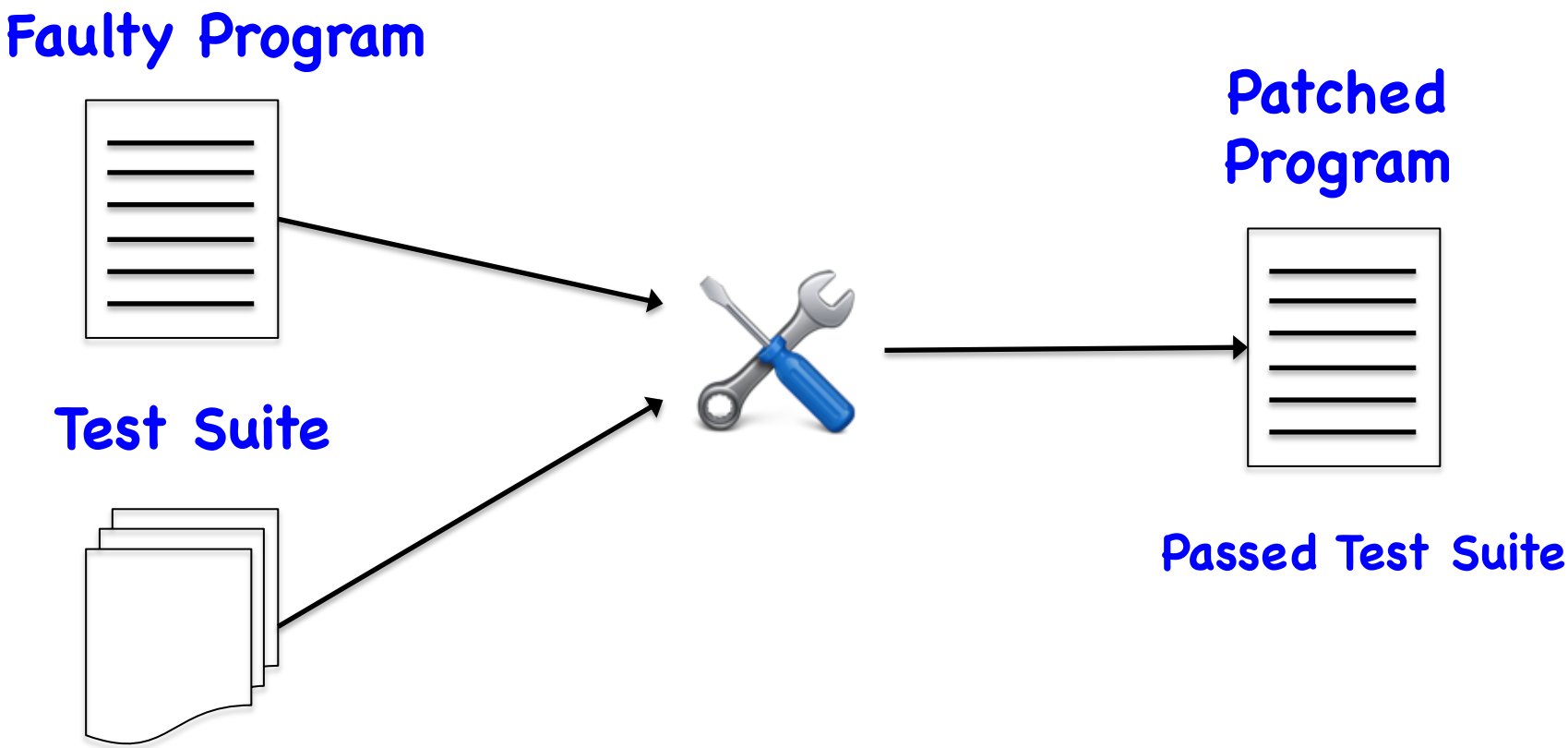
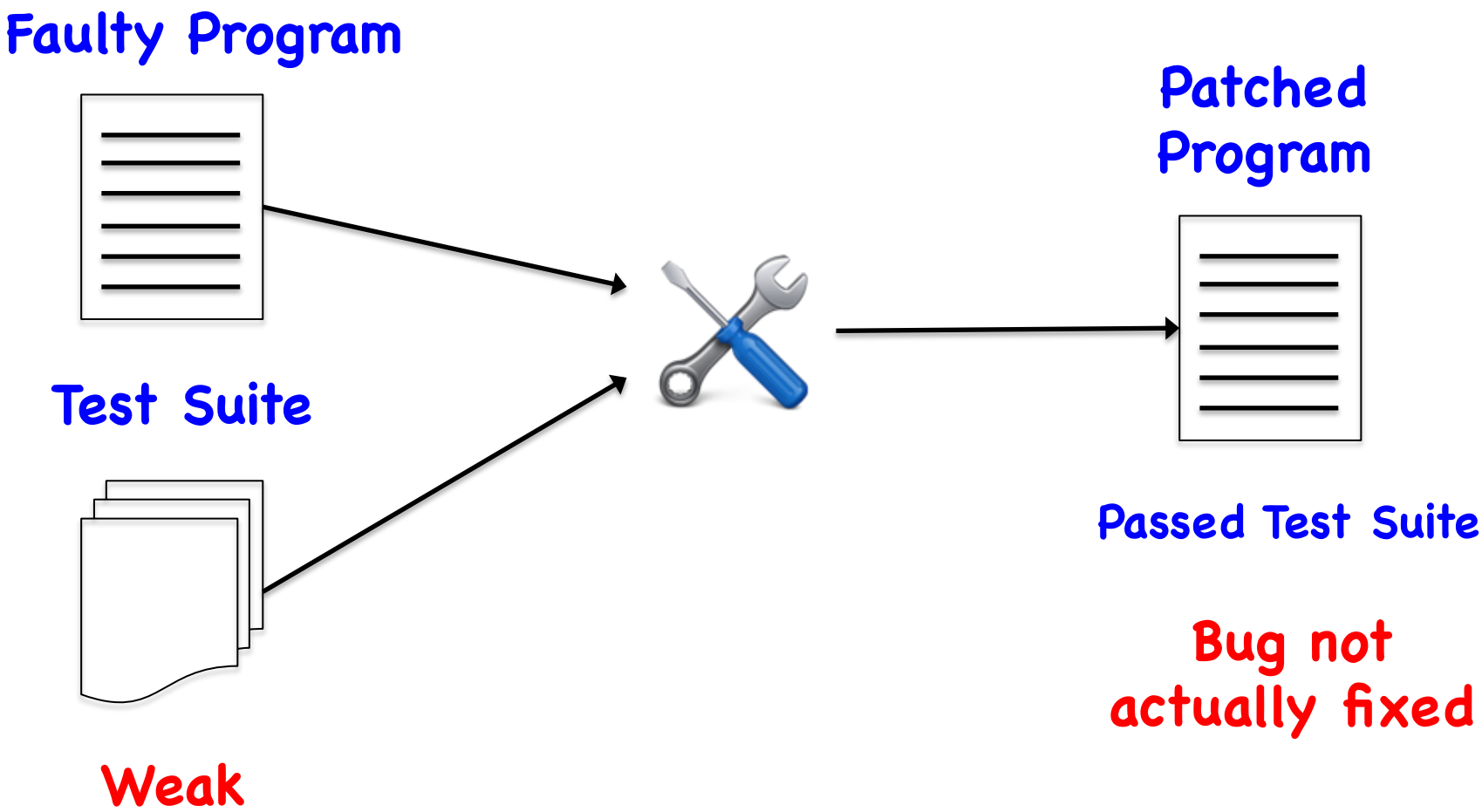
Identifying Test-Suite-Overfitted Patches through Test Case Generation

Qi Xin and Steven P. Reiss
Brown University

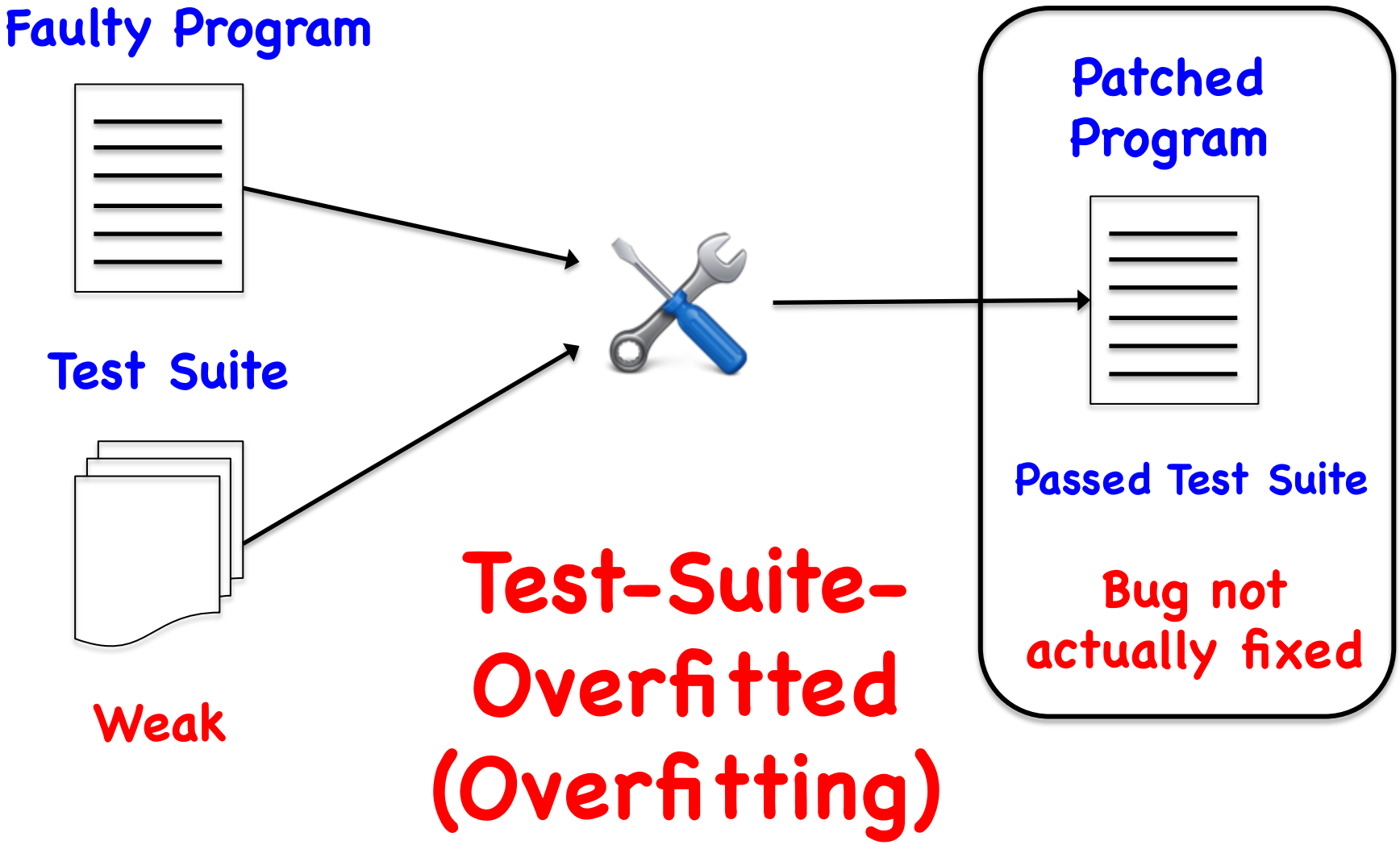
Automated Program Repair



Automated Program Repair



Automated Program Repair



An Overfitting Patch

```
if (fa * fb >= 0.0) { throw new
ConvergenceException(...); }
```

Bug

```
if (fa * fb > 0.0) { throw new
ConvergenceException(...); }
```

Correct Patch

```
if (fa * fb >= 0.0) { throw new
ConvergenceException(...); }
```

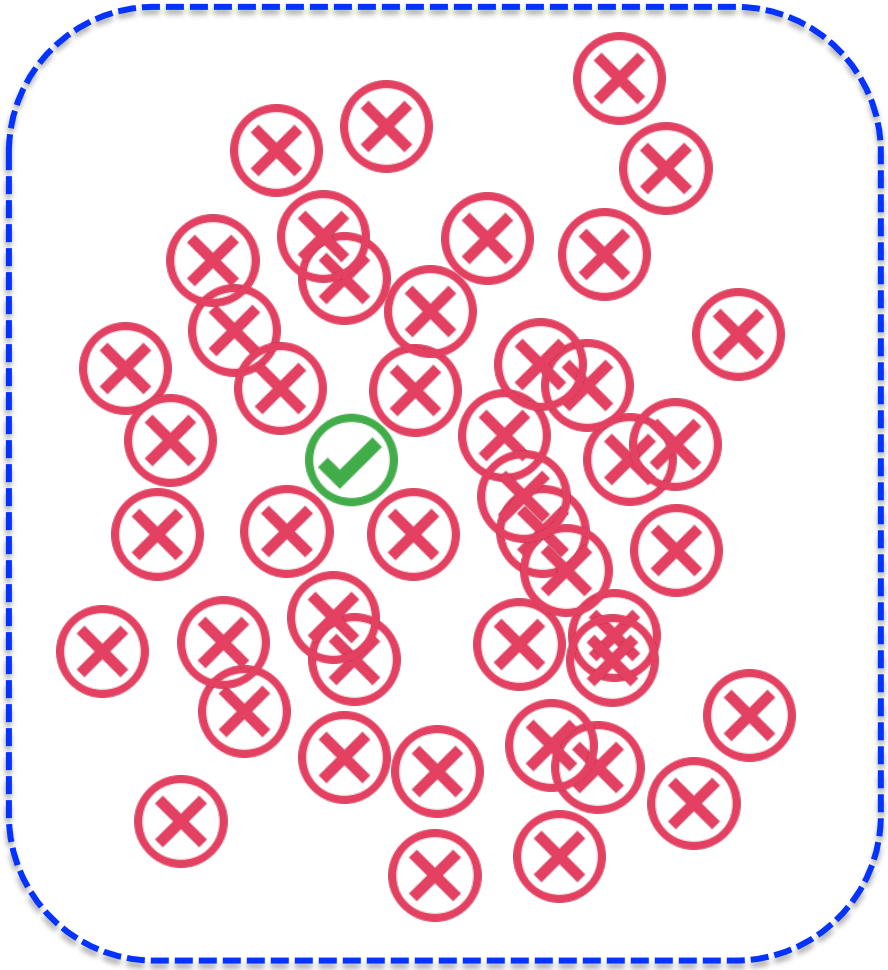
Overfitting Patch

The Overfitting Problem

- Early repair techniques (GenProg, AE, and RSRepair)
  - Generated patches are often overfitting (Smith et al. 15)
  - The majority of generated patches are overfitting (Qi et al. 15)
- The state-of-the-art repair techniques (SPR & Prophet)
  - More than 60% first-found patches are overfitting (Long & Rinard 15, 16)

A Study by Long & Rinard

Search Space of test-passing patches



Correct Patch

Overfitting Patch

A Study by Long & Rinard

Hard to Isolate a Correct Patch from so many Overfitting ones

Our Approach

Identifying an Overfitting Patch through Test Case Generation

Our Approach

faultprog

patchprog

Faulty & Patched Programs

Our Approach

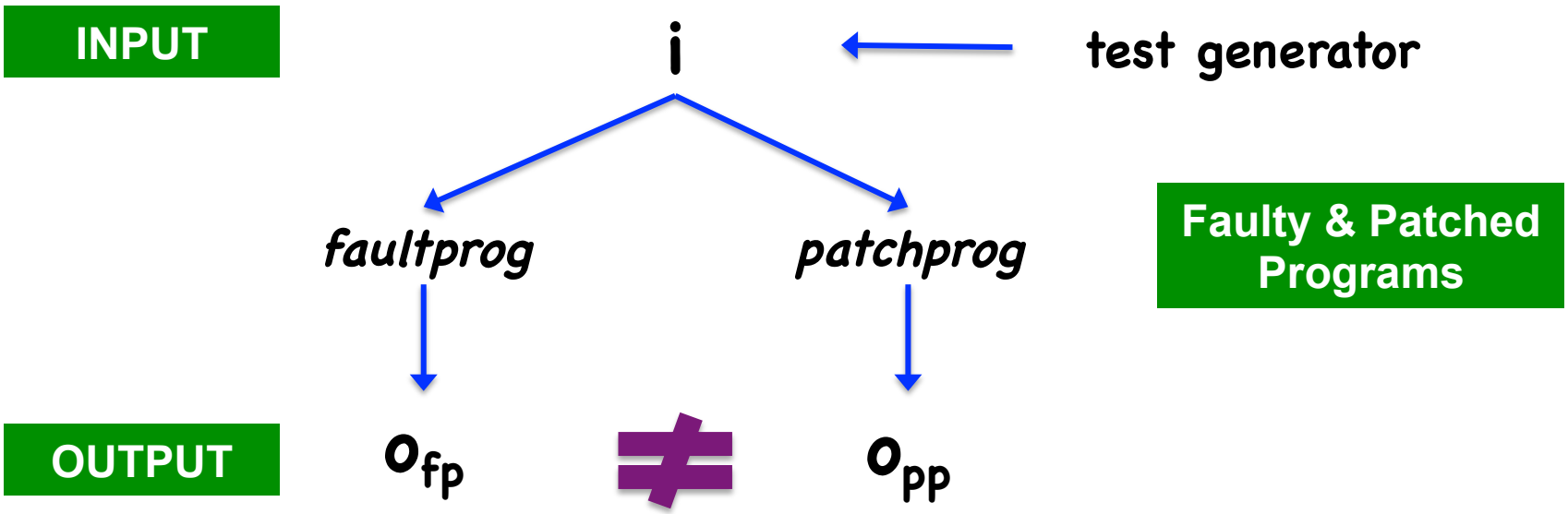
INPUT

i ← test generator

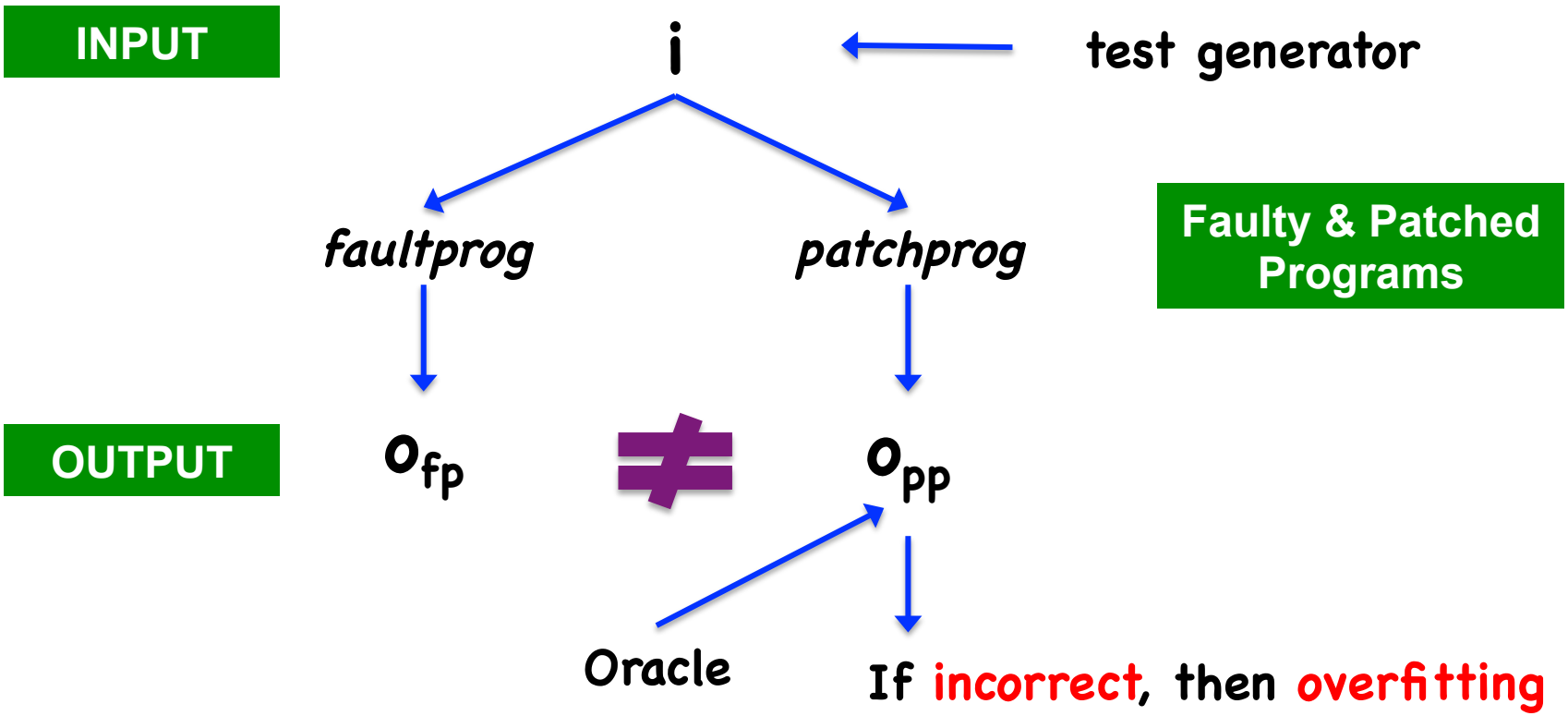
faultprog *patchprog*

Faulty & Patched Programs

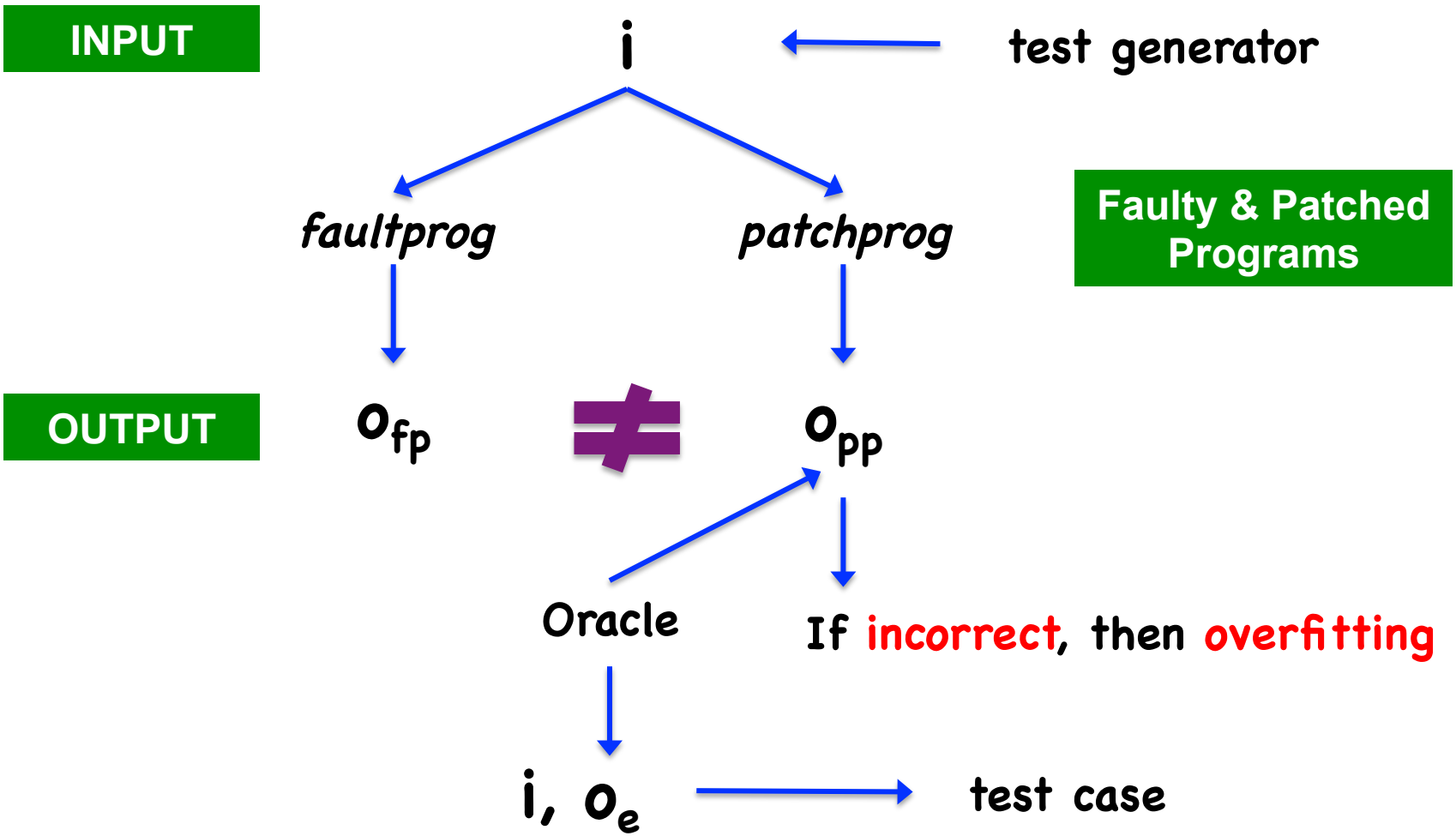
Our Approach



Our Approach



Our Approach



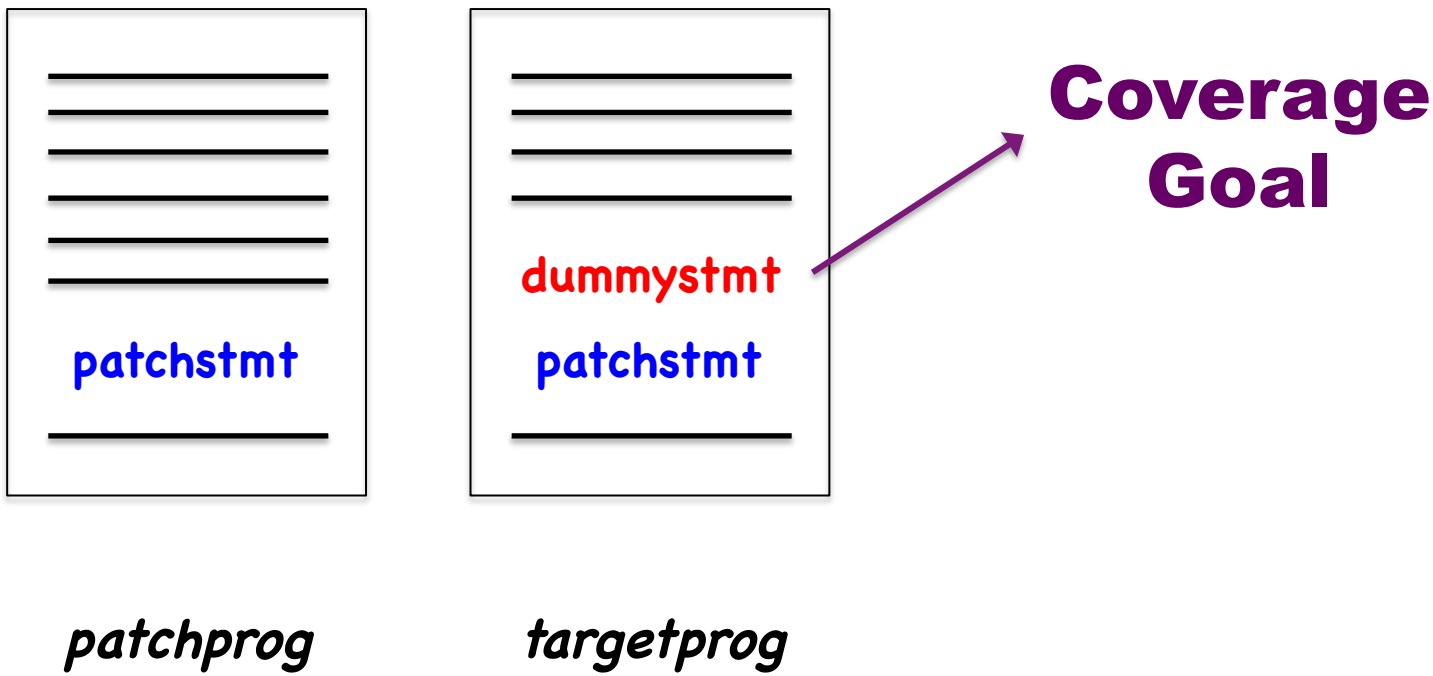
DiffTGen

- Input
  - Faulty program (***faultprog***)
  - Patched program (***patchprog***)
  - Syntactic modifications (Δ_{syn})
    - δ_{syn}=<**faultstmt, patchstmt**>
- Output
  - Test Case
  - Instrumented faulty program (for testing)

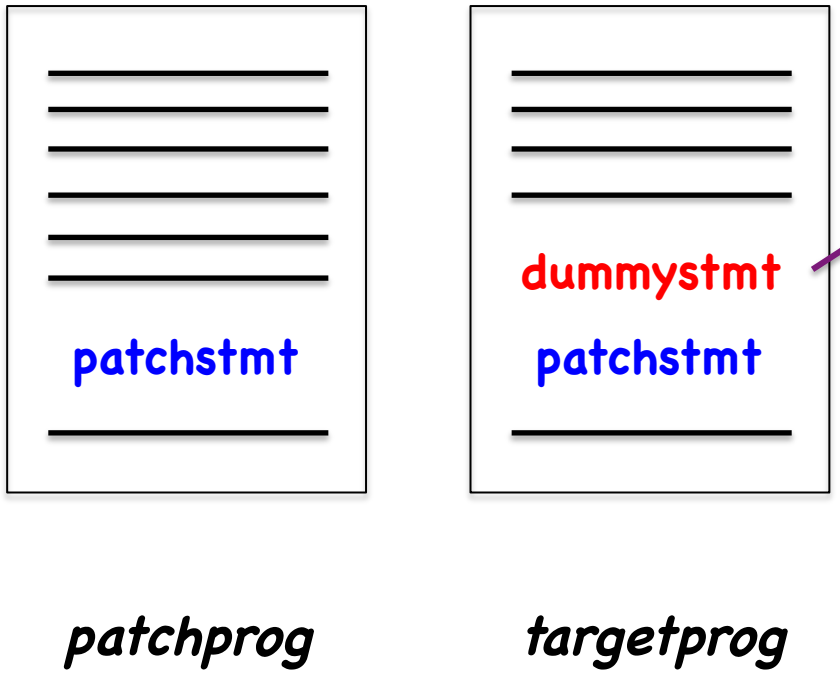
DiffTGen

- Work in three stages:
  1. Test Target Generation
  2. Test Method Generation
  3. Test Case Generation

Test Target Generation



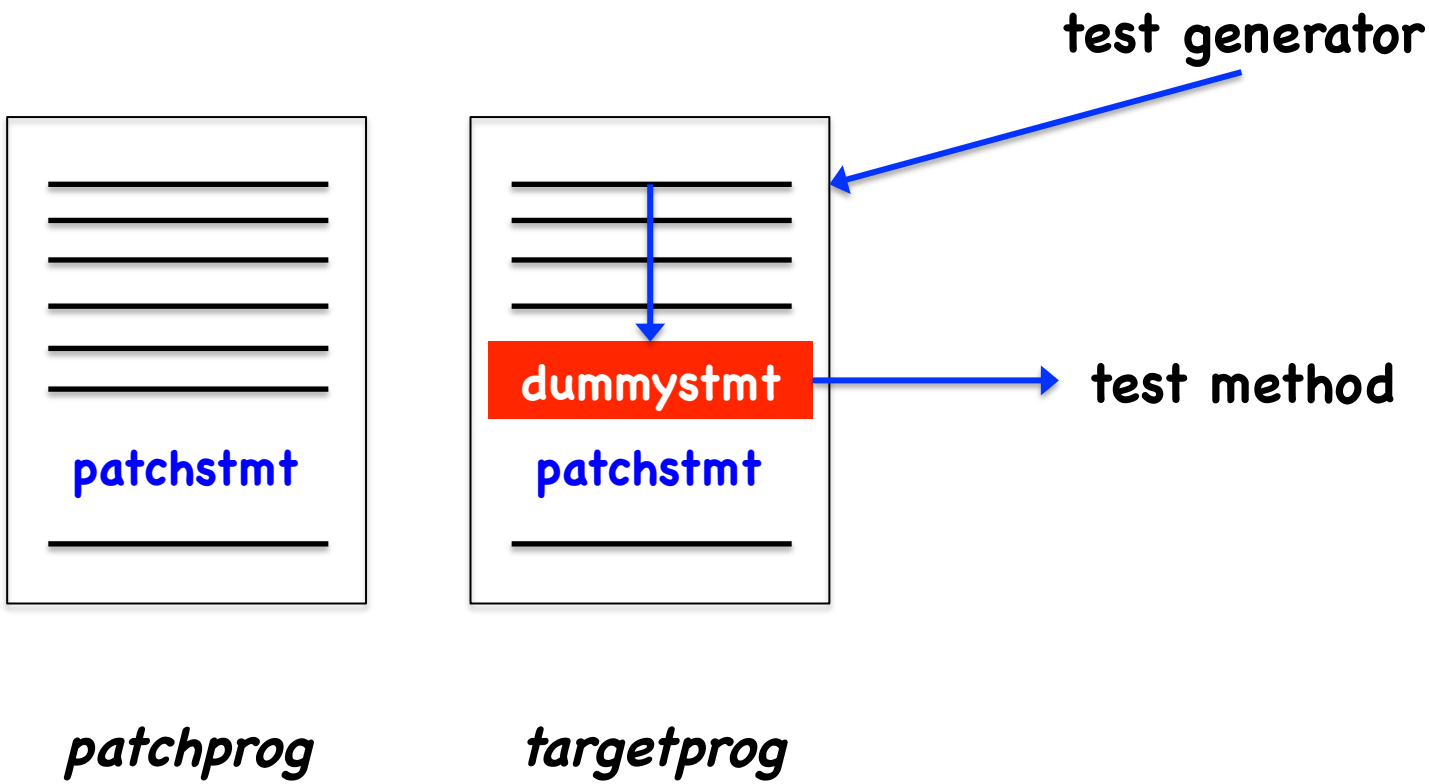
Test Target Generation



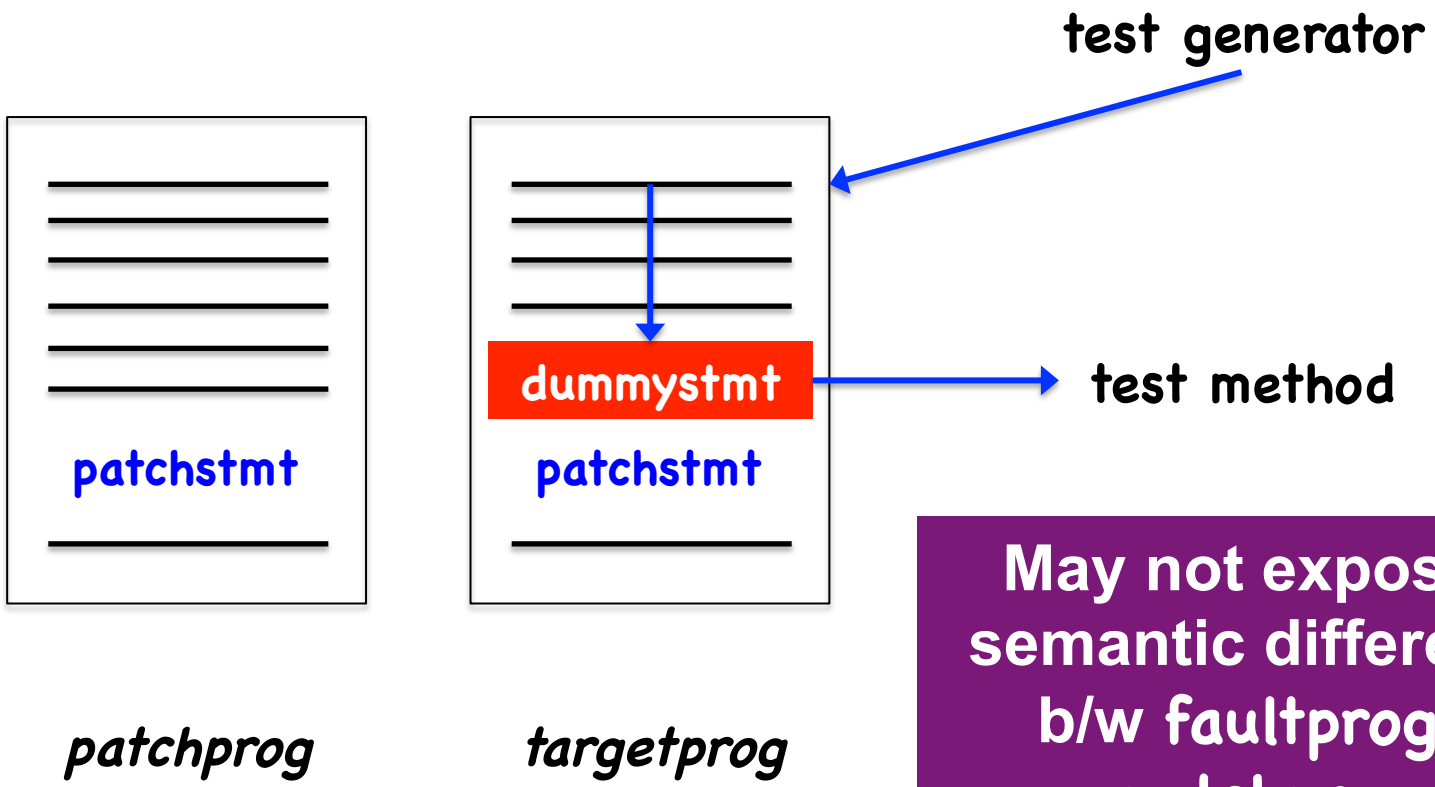
Coverage Goal

A test input covering dummystmt can exercise a δ_{syn}

Test Method Generation

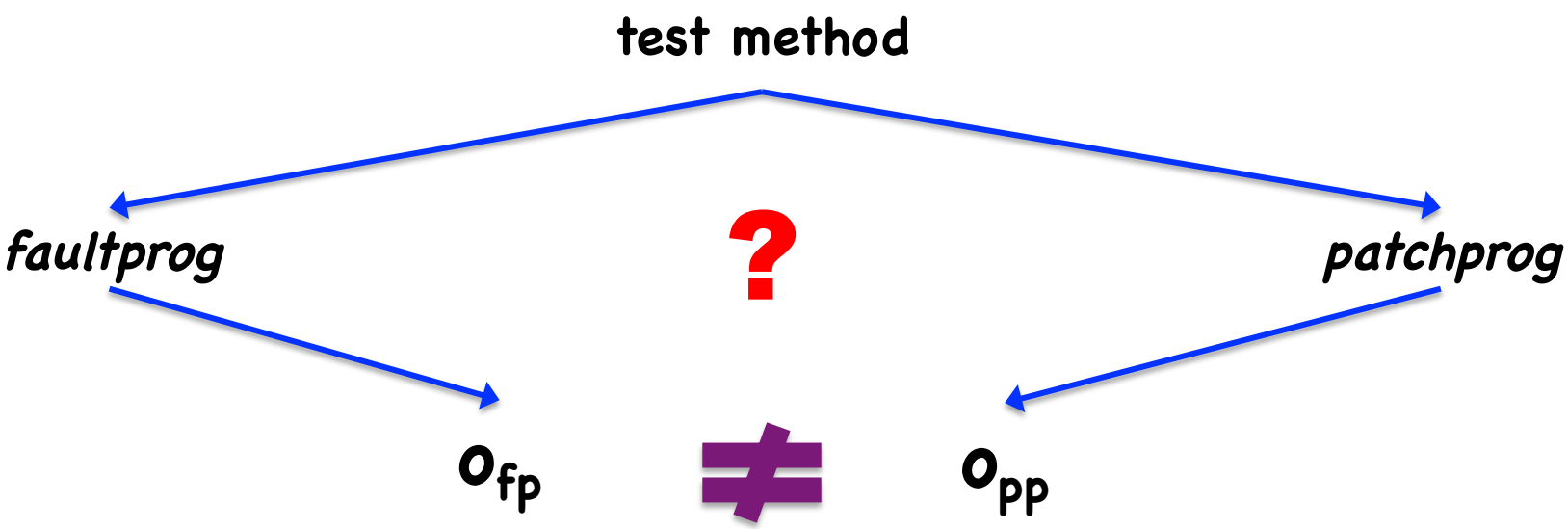


Test Method Generation

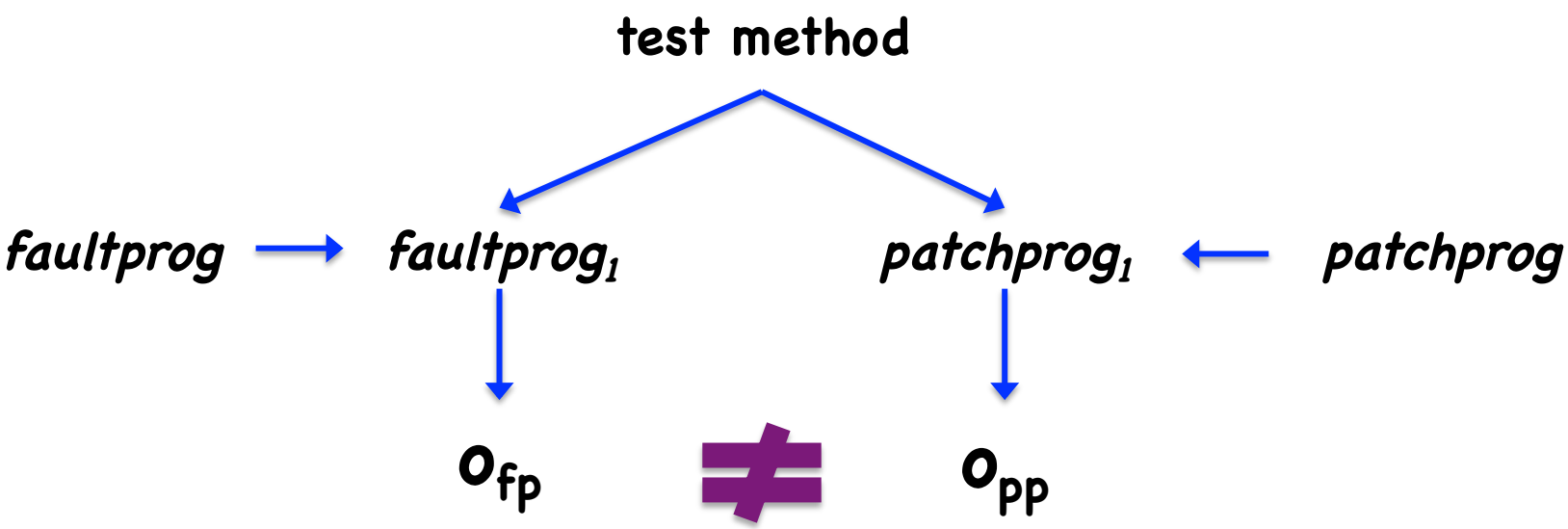


May not expose a semantic difference b/w faultprog & patchprog

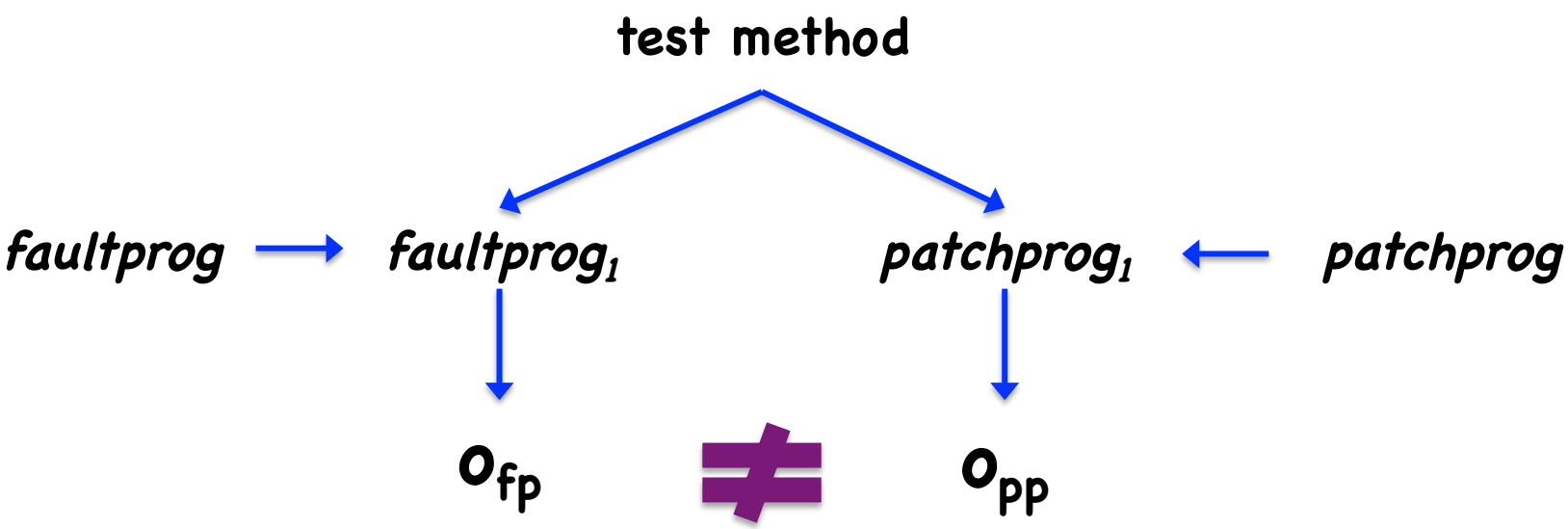
Test Method Generation



Test Method Generation

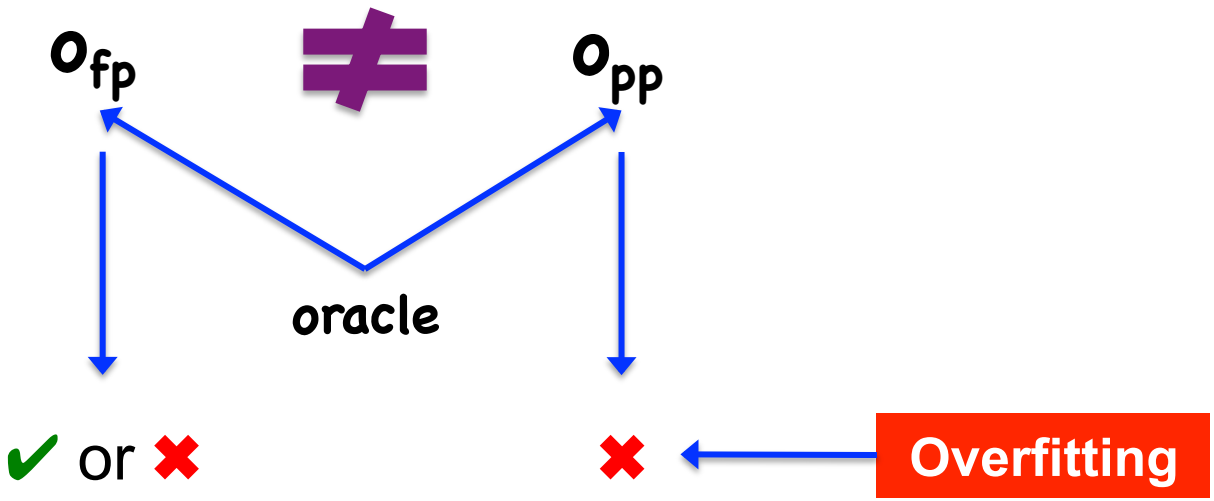


Test Method Generation



Instrumentation (x to x_1): Create printing statements for arguments, return value, and throwables of δ_m

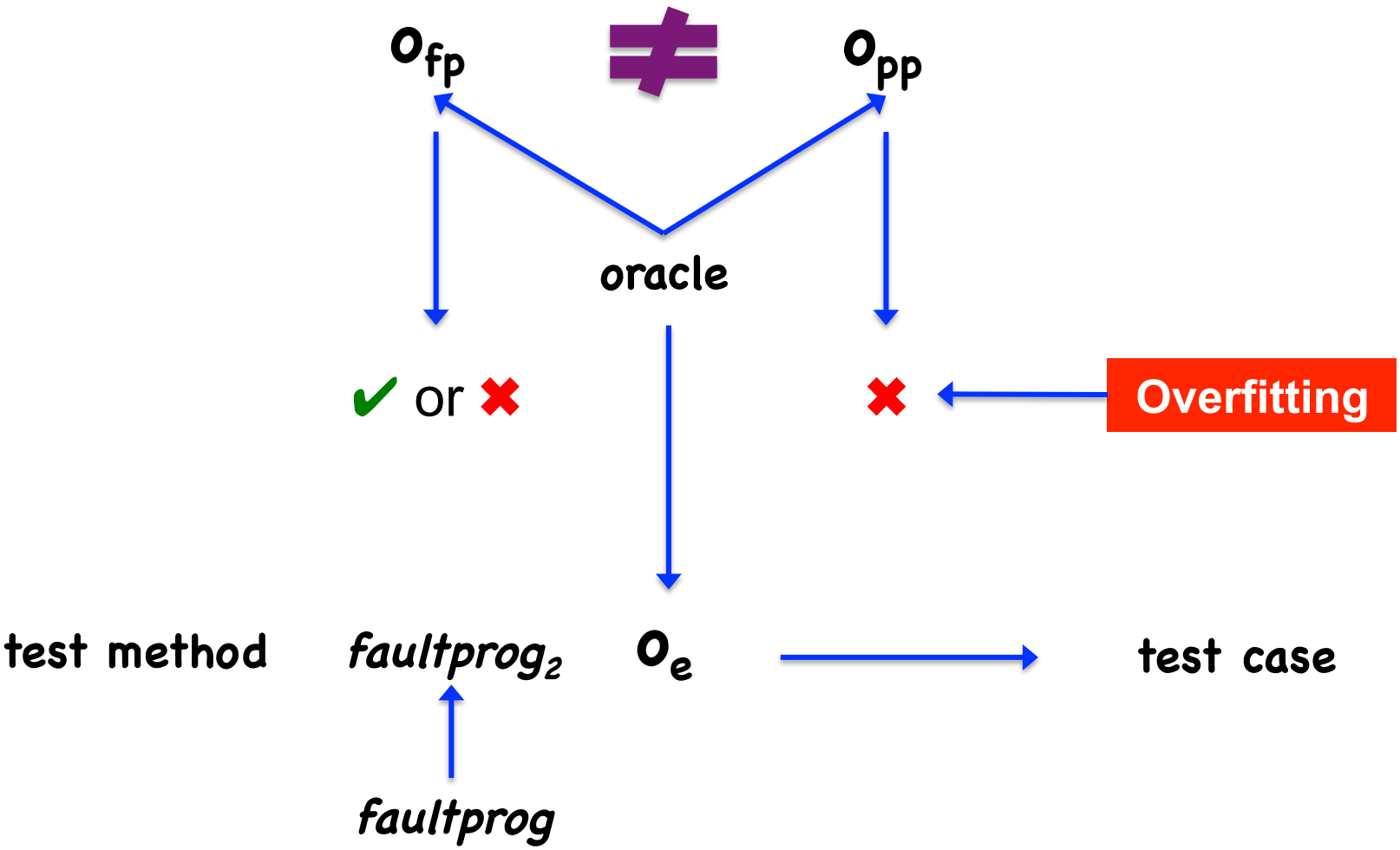
Test Case Generation



o_{fp} is ✔, o_{pp} is ✖: Regression-Containing

o_{fp} is ✖, o_{pp} is ✖: Defective

Test Case Generation



Empirical Evaluation

- RQ1
  - Can DiffTGen identify overfitting patches effectively?
- RQ2
  - Can DiffTGen help a repair technique produce less overfitting patches and more correct ones?

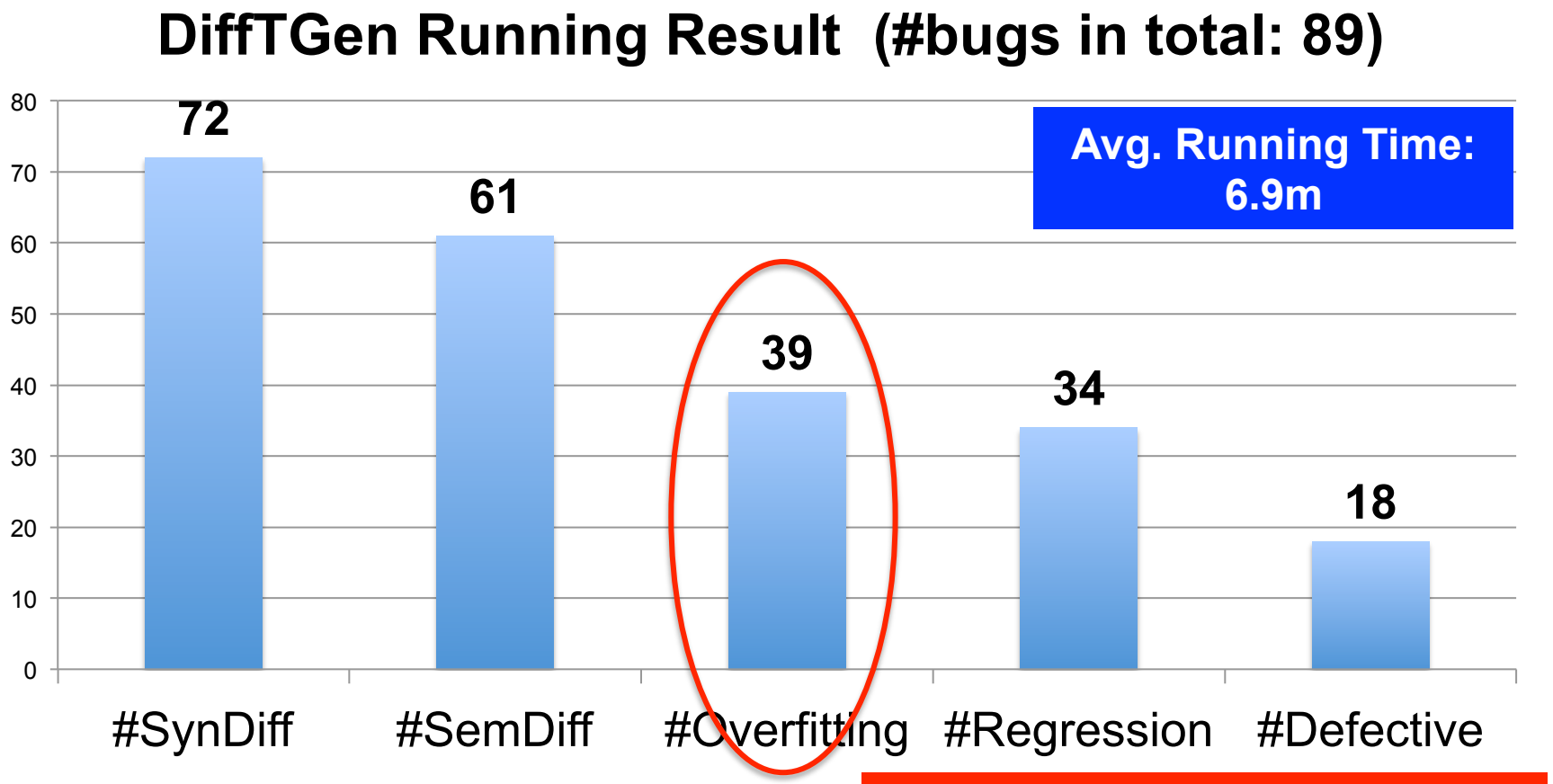
Experimental Setup

- Patch Dataset
  - 89 patches collected from two repair experiments (by Martinez et al. and by Le et al.)
    - Generated by four repair techniques
    - Generated for bugs from Defects4J dataset
  - Manually determine 10 patches to be non-overfitting
    - By comparing them to the developer patches from Defects4J
    - The other 79 patches are likely to be overfitting

Experimental Setup

- DiffTGen Implementation
  - ***faultprog***, ***patchprog***, Δ_{syn}
    - Manually provided
  - Oracle
    - bug-fixed program (developer-patched program) from Defects4J
  - Test generator
    - EvoSuite-1.0.2
  - Test method generation
    - Run EvoSuite in 30 trials with the timeout being 60s for each trial

Result



Can identify 49.4% (39/79) overfitting patches in less than 7 minutes on average

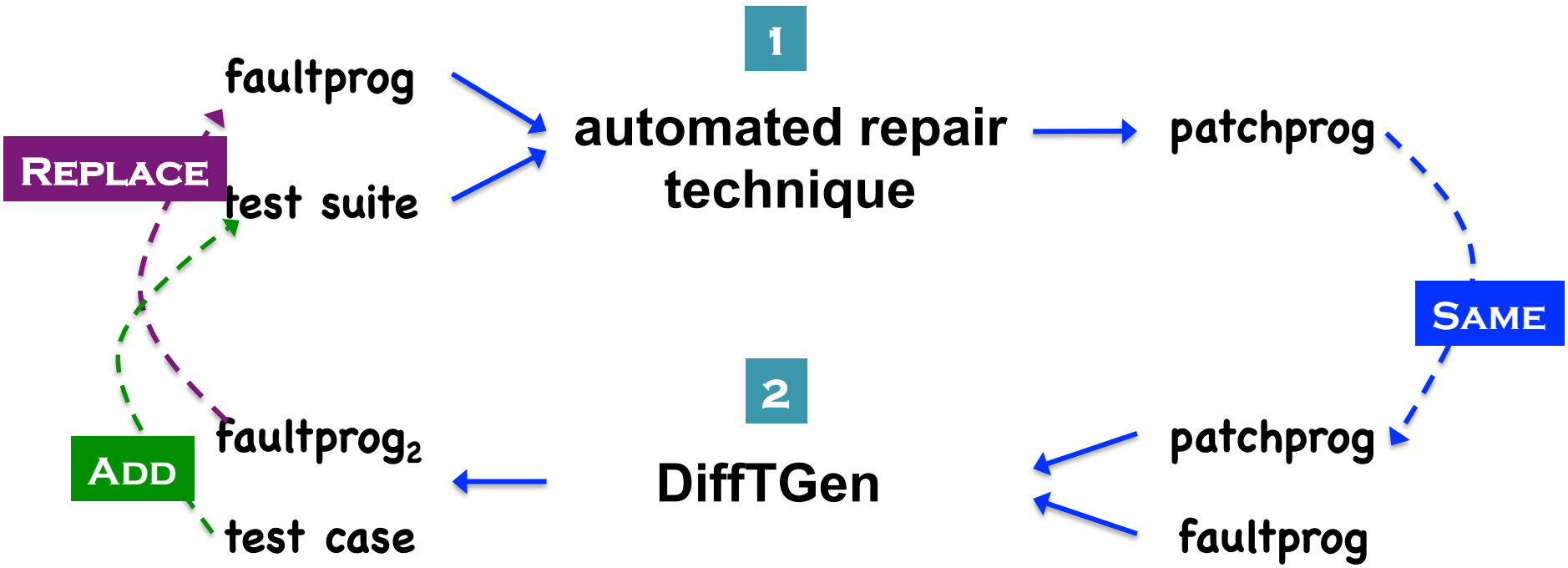
Empirical Evaluation

- RQ1
  - Can DiffTGen identify overfitting patches effectively?
- RQ2
  - Can DiffTGen help a repair technique produce less overfitting patches and more correct ones?

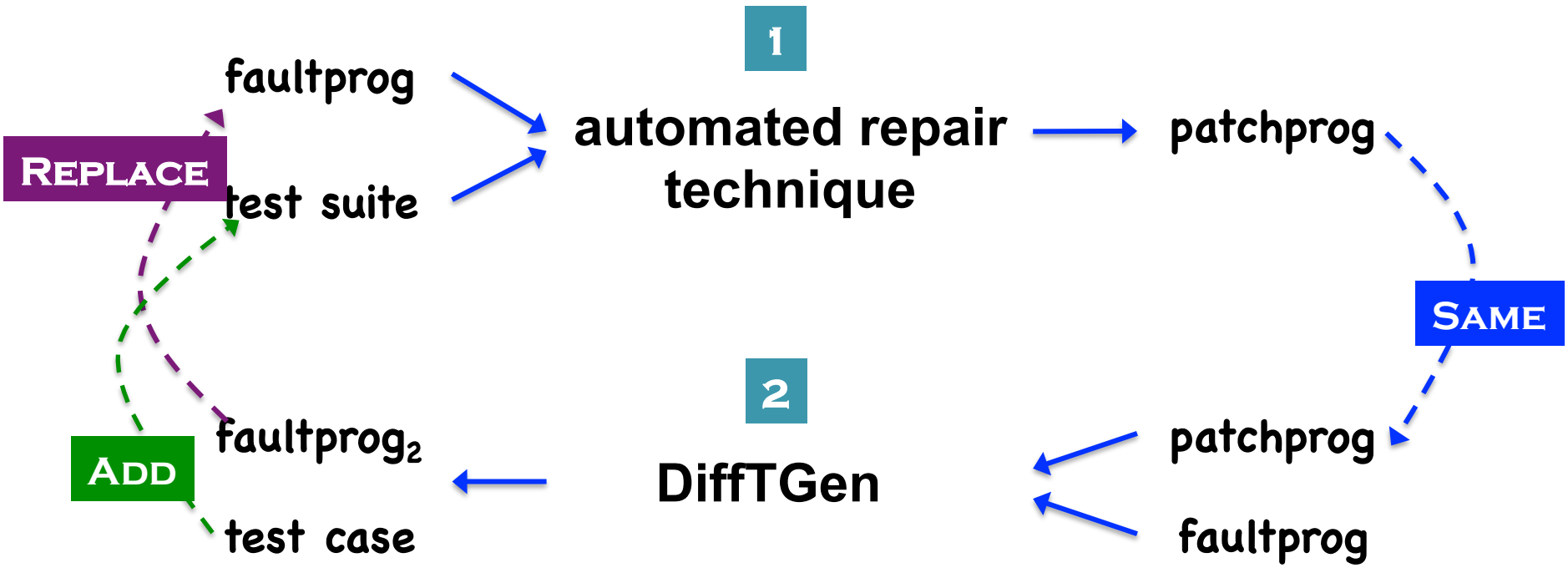
Experimental Process

- For each of the 39 bugs whose patches were found to be **overfitting**,
  - we added the generated test case to the original test suite
  - then …

Experimental Process

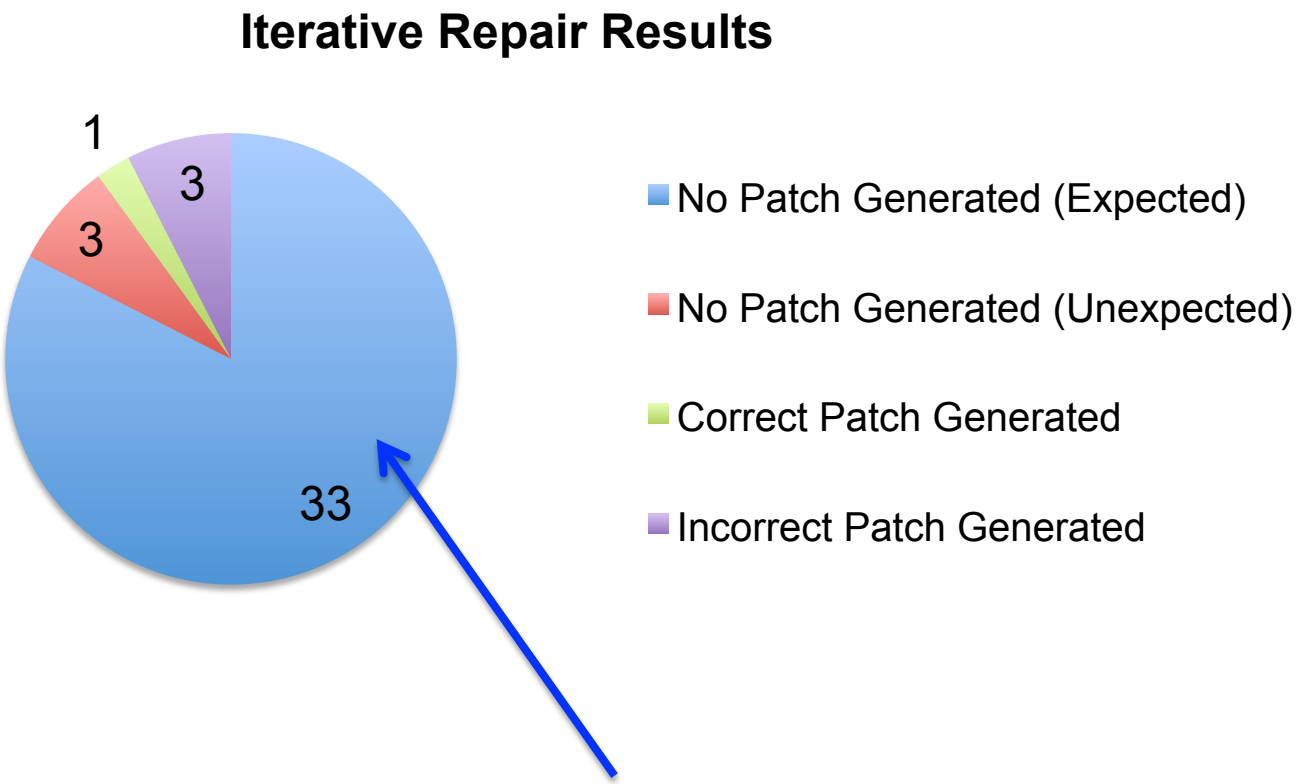


Experimental Process



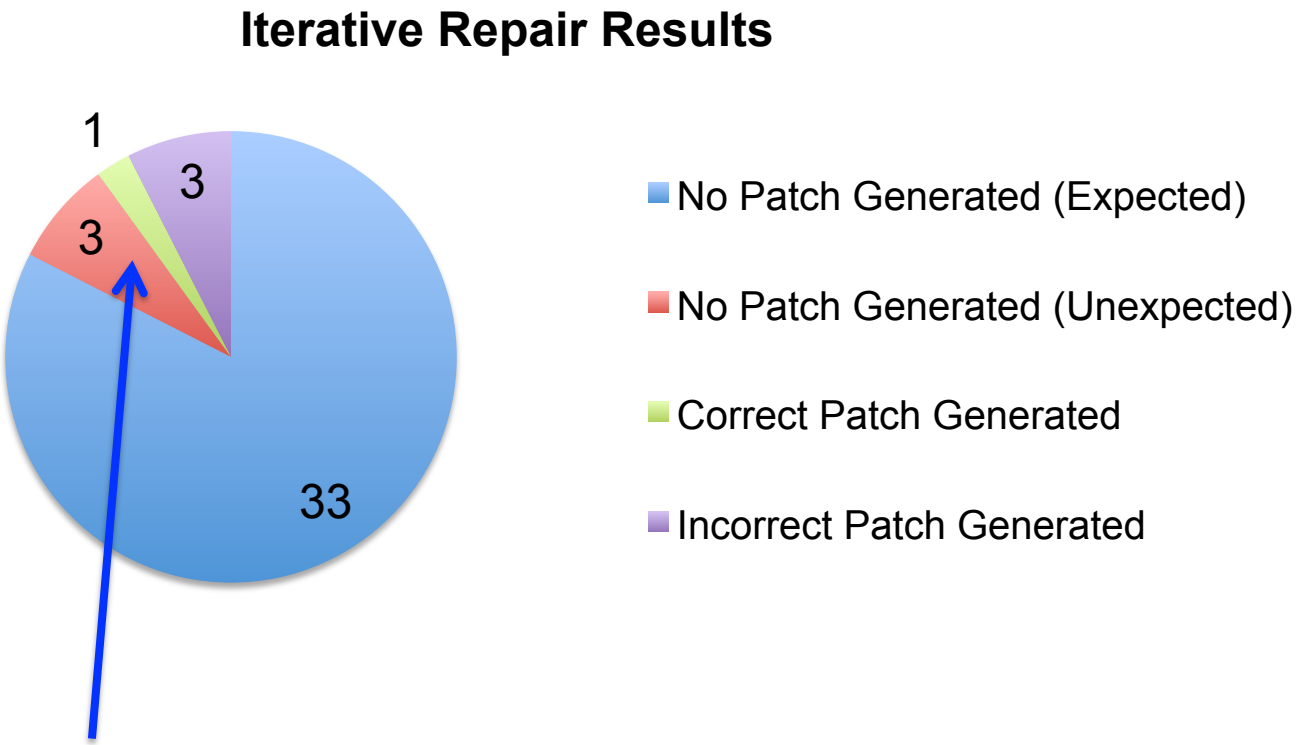
Repeat 1 & 2 until either fails

Result



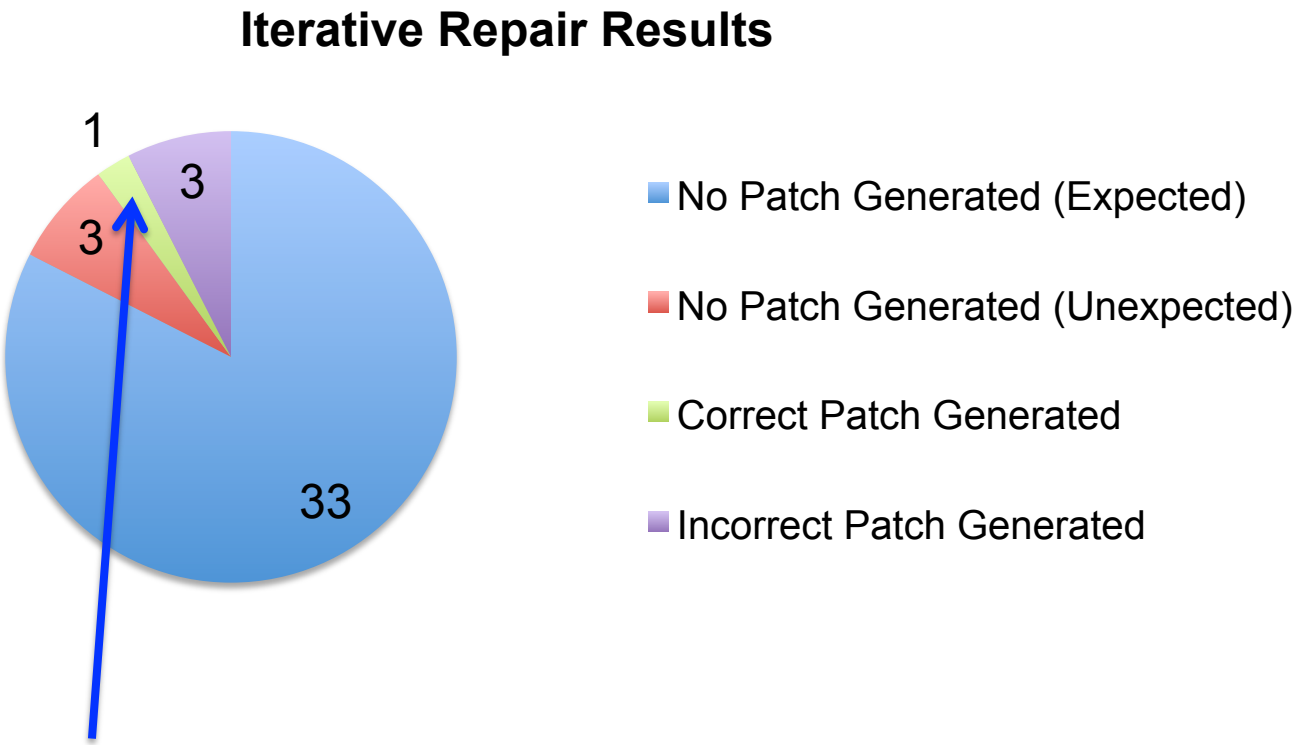
The correct patches *do not* exist in the repair techniques' search spaces, and no patches were generated

Result



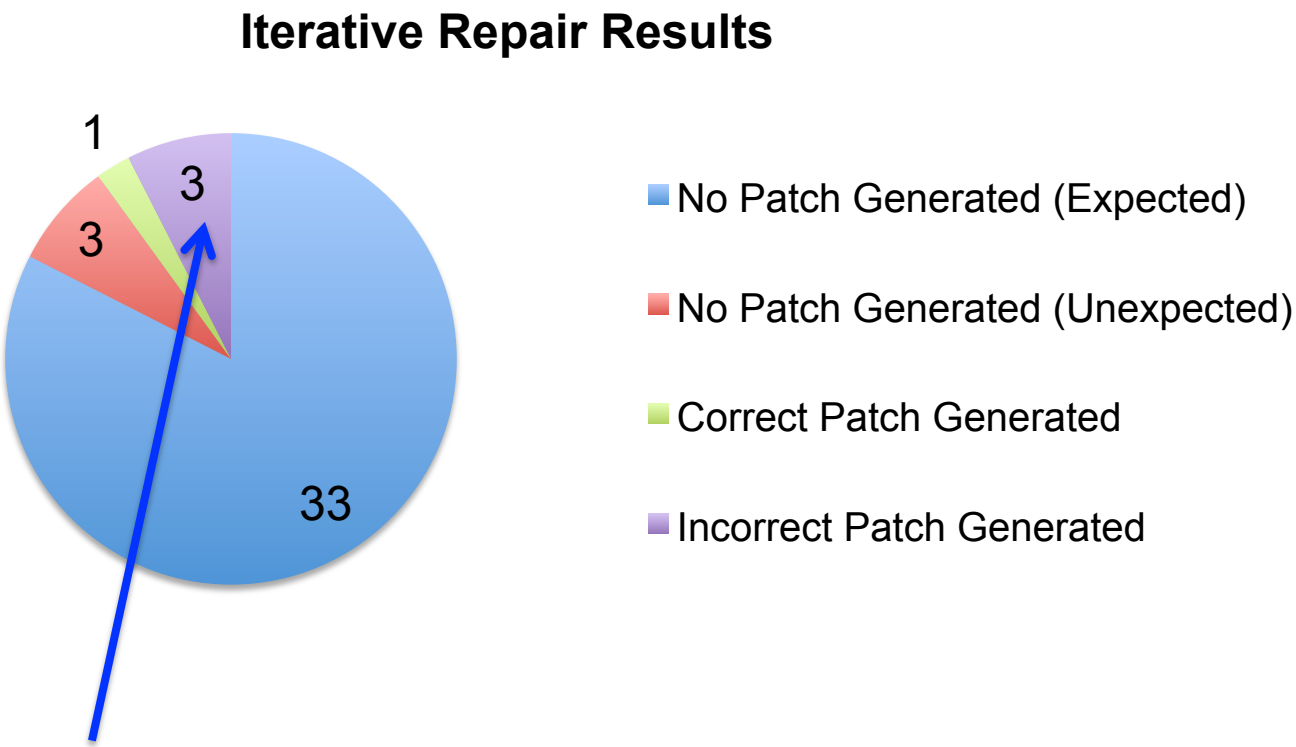
The correct patches *do exist*, but the repair techniques failed to find them

Result



Correct patches were generated through using HDRepair & DiffTGen

Result



Invalid patches generated for one bug, and DiffTGen failed to identify three patches generated for two bugs as overfitting

RQ2

- Based on our results,
  - DiffTGen can help a repair technique avoid yielding an overfitting patch

RQ2

- Based on our results,
  - DiffTGen can help a repair technique avoid yielding an overfitting patch
  - Often times, a repair technique fails to generate a patch using the augmented test suite

RQ2

- Based on our results,
  - DiffTGen can help a repair technique avoid yielding an overfitting patch
  - Often times, a repair technique fails to generate a patch using the augmented test suite
  - Whether a repair technique can eventually produce a correct patch largely depends on its repair ability

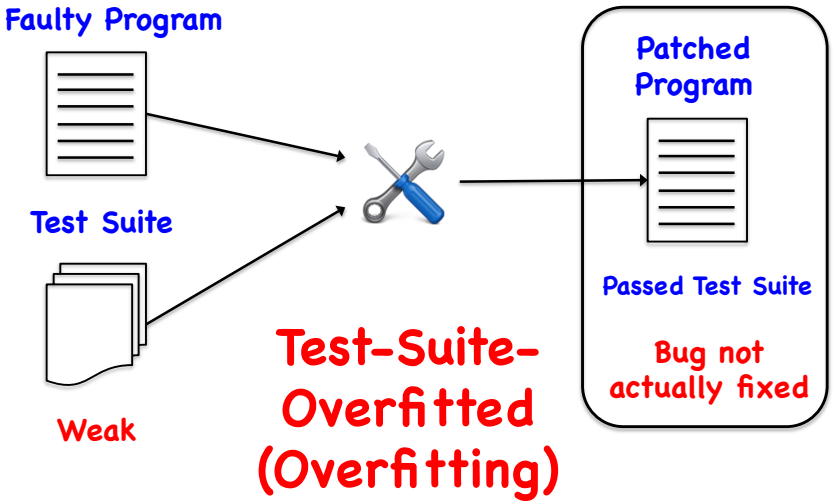
Conclusion

- Current automated program repair technique suffers from patch overfitting
- DiffTGen can identify an overfitting patch through test case generation
- The generated test case can help a repair technique avoid yielding an overfitting patch

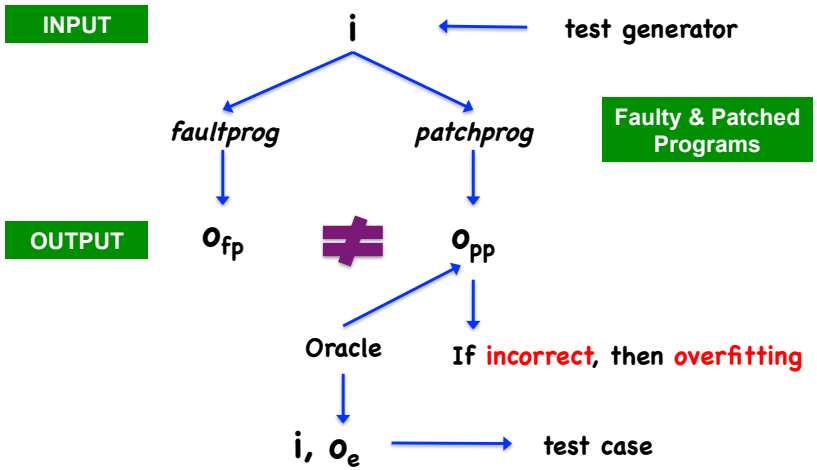
Future Work

- Use a human oracle
  - Need to be human-amenable
- Use more sophisticated differential testing techniques
- Use more repair techniques for experiments

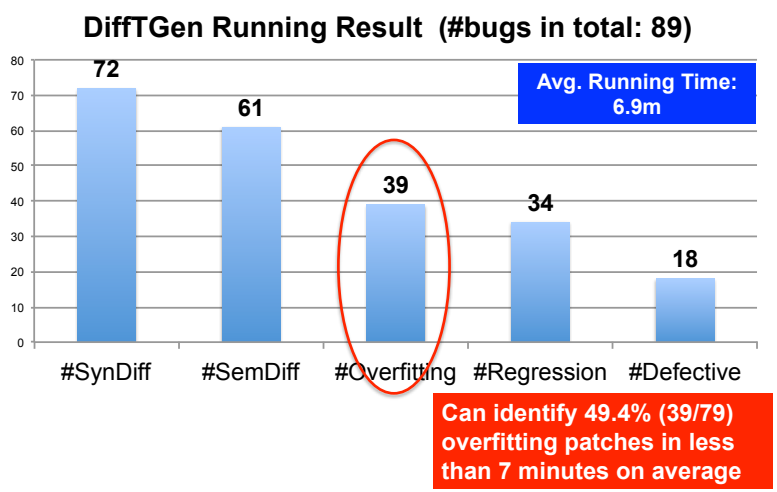
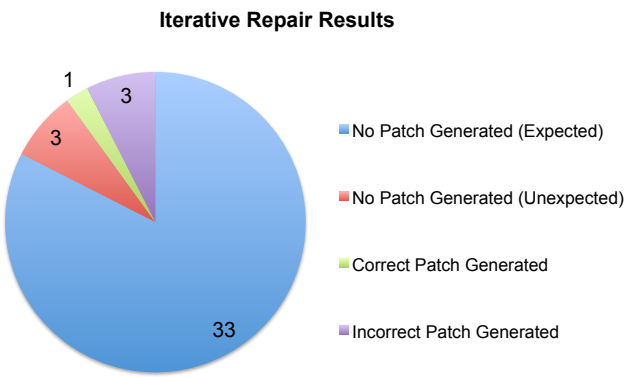
Automated Program Repair



Our Approach



Result



Result