Empirical Study on Synthesis Engines for Semantics-based Program Repair

Xuan Bach D. Le[1], *David Lo*[1], *Claire Le Goues*[2]

[1]School of Information Systems,

Singapore Management University

[2]School of Computer Science,

Carnegie Mellon University

{dxb.le.2013,davidlo}@smu.edu.sg

clegoues@cs.cmu.edu

SE@CMU is recruiting graduate students!

- Send us your awesome undergrads!
- Tell them to check the box next to “PhD in Software Engineering”, so we’re certain to see them!
- Early deadline: December 1
- Final, we-mean-it deadline: December 15.

Automatic patch generation seeks to improve software quality.

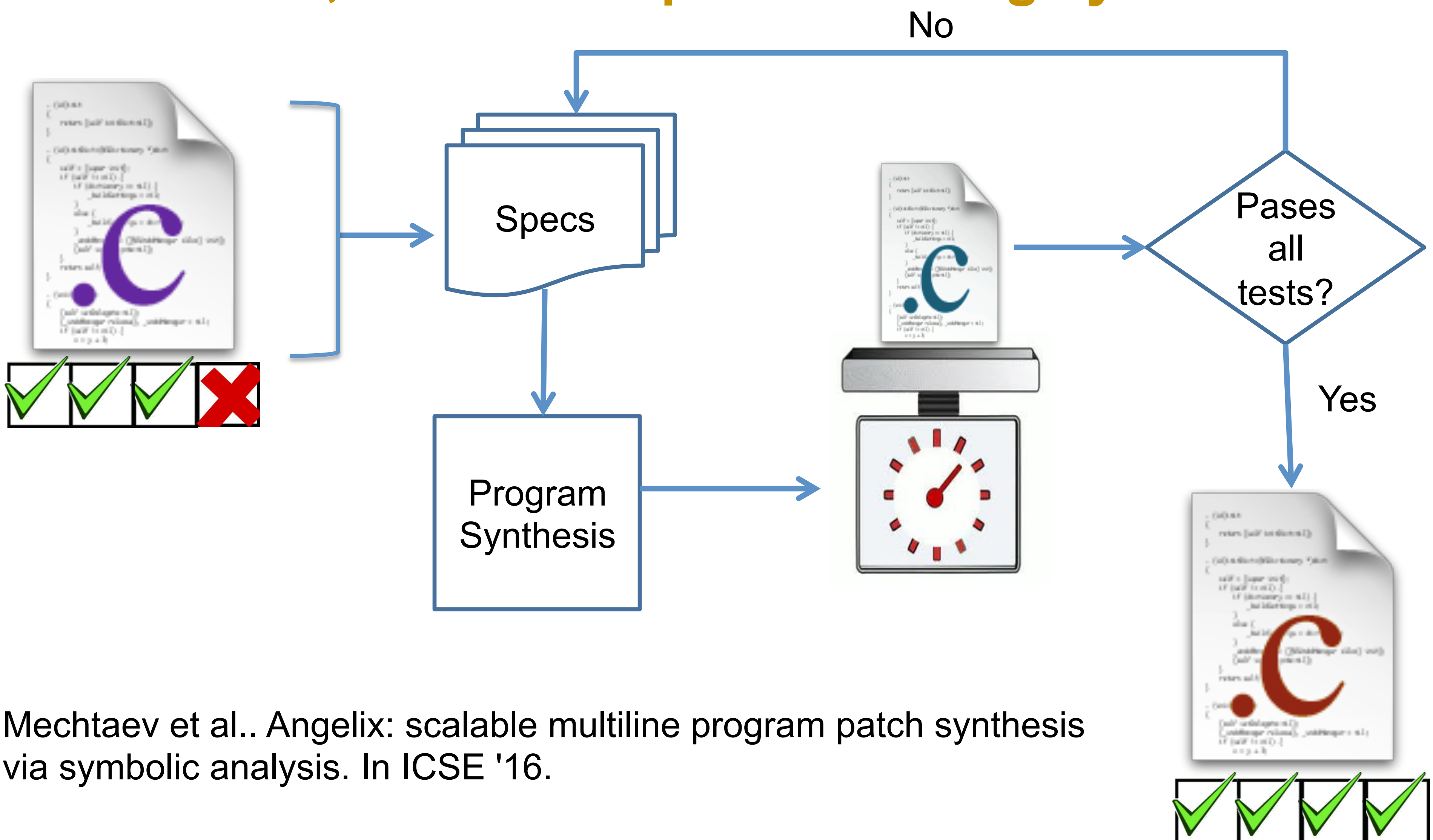
- Bugs in software incur tremendous maintenance cost.

In 2006, everyday, almost 300 bugs appear in Mozilla […] far too much for programmers to handle

- Developers presently debug and fix bugs manually.
- Automated program repair:

APR = Fault Localization + **Repair Strategies**

Semantics-based repair extracts value-based specifications using tests + symbolic execution, constructs patches using synthesis.



Mechtaev et al.. Angelix: scalable multiline program patch synthesis via symbolic analysis. In ICSE '16.

Our contributions:

- Pluggable framework to assess many types of syntax-guided program synthesis in the core of Angelix.
- Evaluation of different synthesis techniques for semantic program repair.
  - Key finding: effectiveness of different synthesis engines varies!

Syntax Guided Program Synthesis (SyGuS)

- Key idea: use a restricted grammar to describe syntactic space of possible solutions
- Use different search techniques to search for solutions conforming to provided grammar
- We evaluate: Enumerative, Stochastic, Symbolic, and CVC4

Example of Buggy Program

```
bool min(int x, int y){
    bool cond = x < y;
    if(cond){
        return true;
    }else{
        return false;
    }
}
```

Test #	Value of x	Value of y	Expected output
1	1	2	true
2	2	2	true
3	5	2	false

FAILED

Selective Symbolic Execution

```
bool min(int x, int y){
  bool cond = α;
  if(cond){
    return true;
  }else{
    return false;
  }
}
```

Replace buggy expression by symbolic variable

- ✓ Switch to symbolic execution when necessary, collect path conditions.
- ✓ Infer specs for each test

x	y	expected
2	2	true

PC2: (not α) & x = 2 & y = 2 & expected output[true] = actual output[false]

PC1: α & x = 2 & y = 2 & expected output[true] = actual output[true]

Extract Value-based Specifications

Satisfiable

PC1: α & $x = 2$ & $y = 2$ &
expected output[true] = actual output[true]

Unsatisfiable

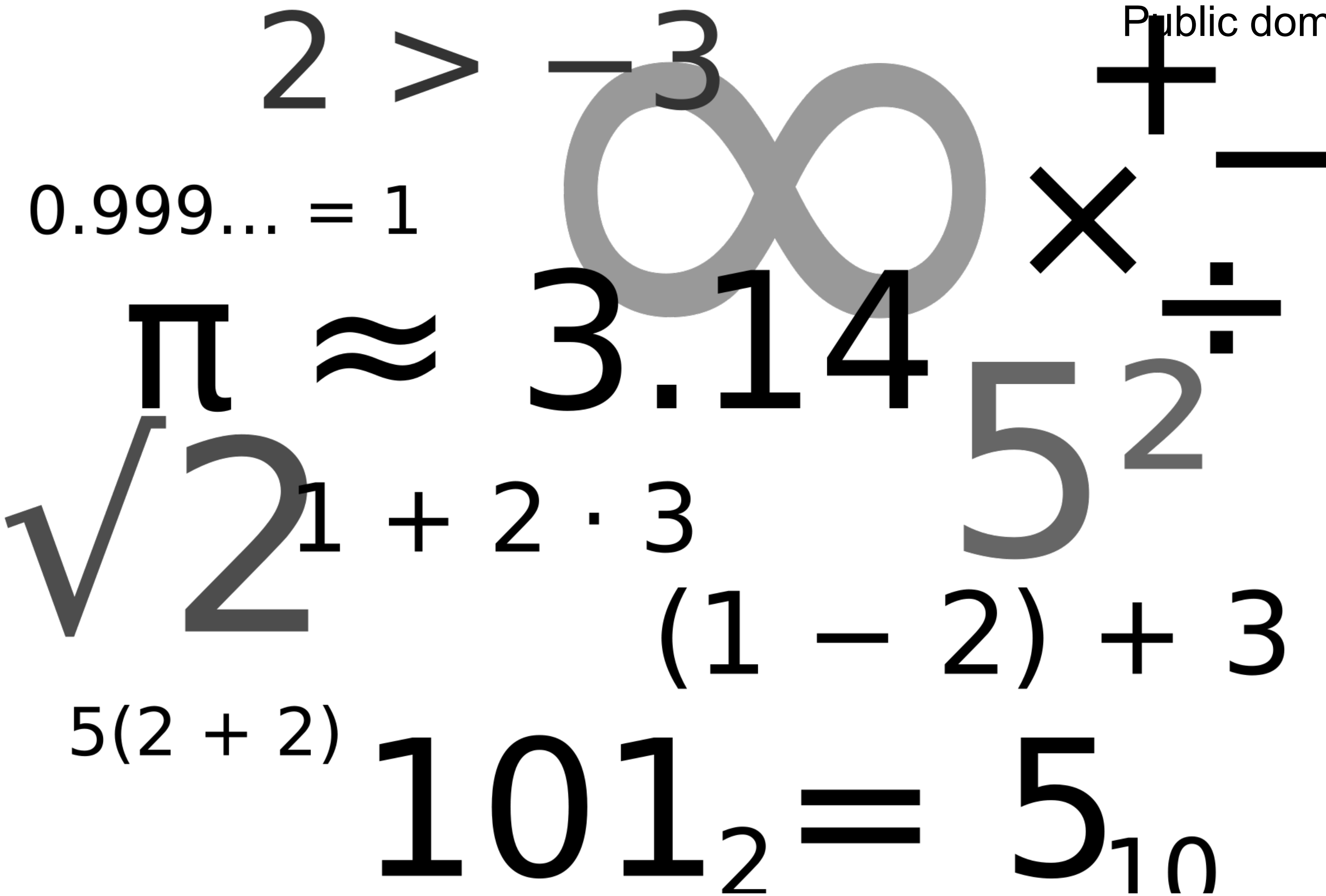
PC2: (not α) & $x = 2$ & $y = 2$ &
expected output[true] = actual output[false]

Model: $x = 2$ & $y = 2$ & α = true

Pre-condition: $x = 2$ & $y = 2$

Post-condition: α = true

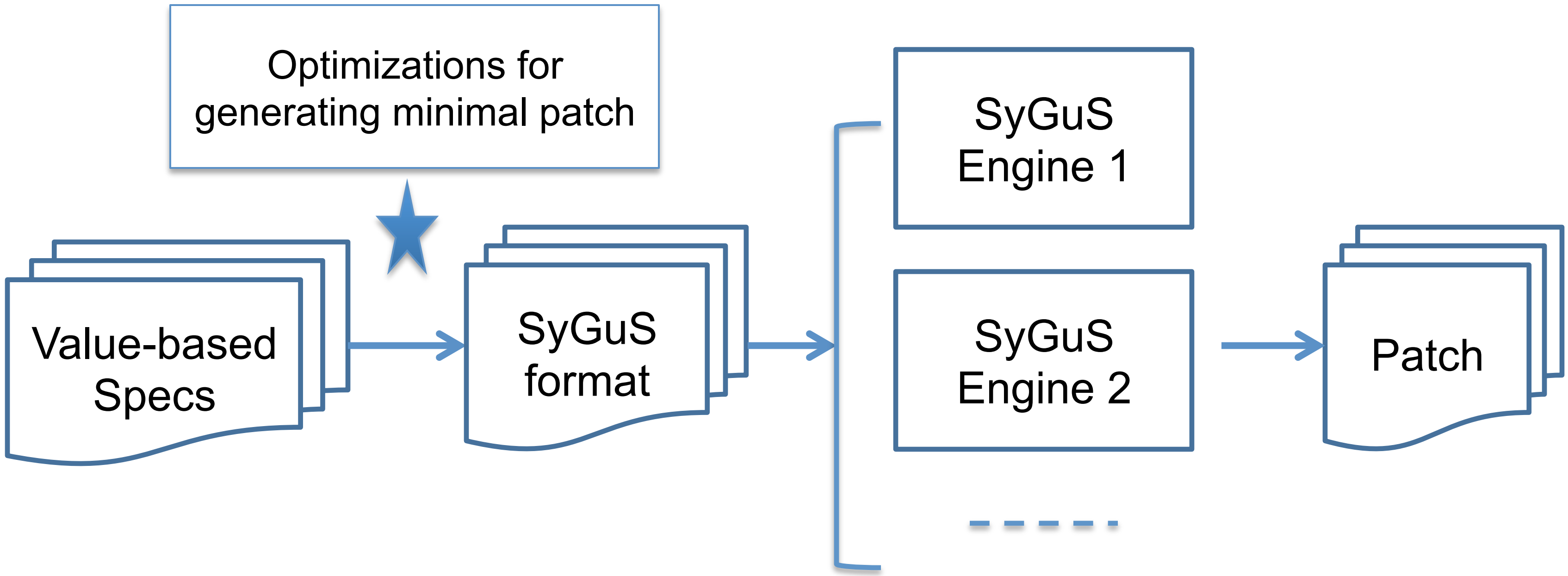
Synthesize α over x and y, permitting a restricted set of components, satisfying spec



Hard constraints on functionality + soft constraints on form + PartialMax SMT = minimal repair

(mathy details elided for brevity.)

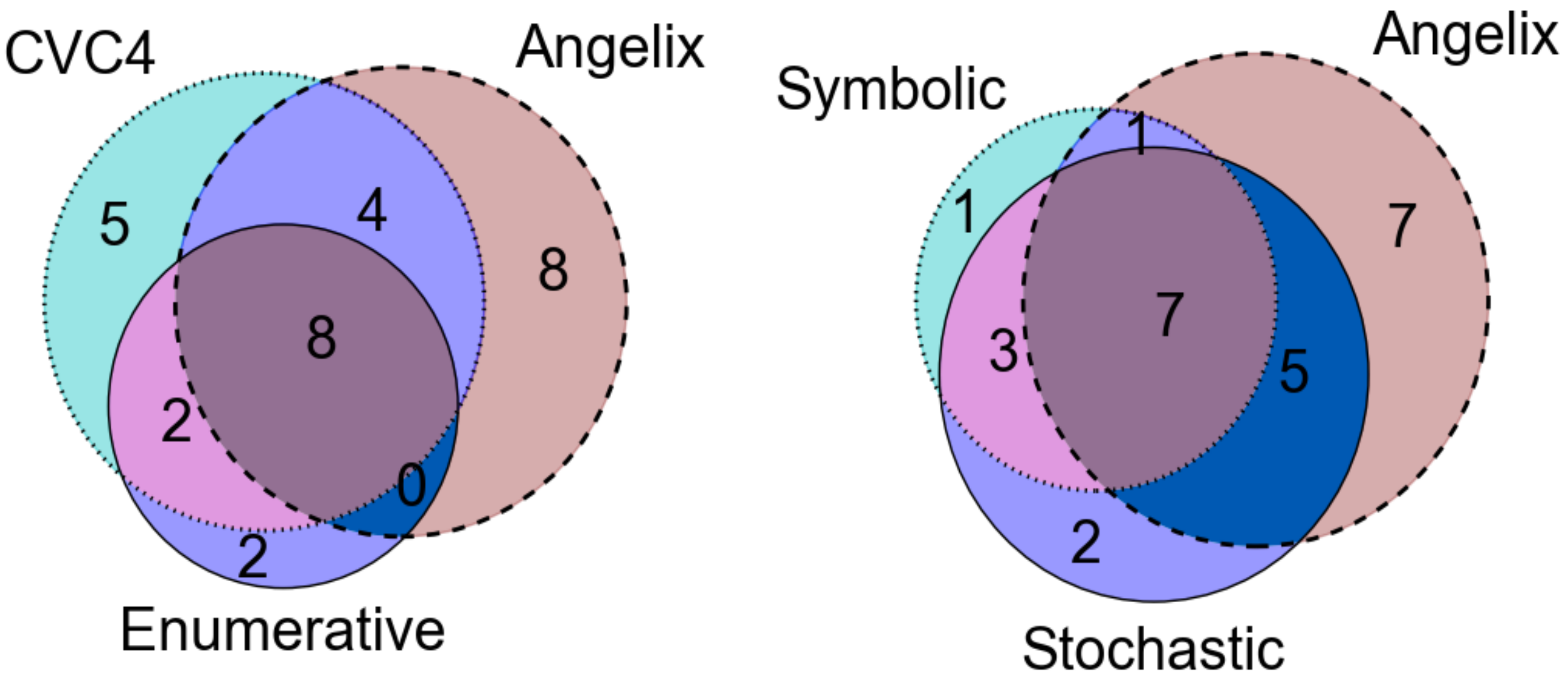
Our framework converts specs inferred by Angelix to generic SyGuS format.



Experiments

- 188 programs from IntroClass benchmark
  - Use black-box tests for repair, white-box tests for testing quality of generated patches.
  - Only report the patches that *generalize* to the held-out tests!
- Synthesis techniques that help generate more correct patches are better
- Evaluate on: Enumerative, Stochastic, Symbolic, CVC4, and Angelix's synthesis engine

Synthesis techniques vary in the bugs they can correctly address.



Summary

- We evaluate the effectiveness of different synthesis techniques in the context of program repair, finding:
  - Performance of synthesis techniques varies
  - Forging results of synthesis techniques increases effectiveness of program repair, e.g., fix more bugs
- We plan to develop more effective synthesis techniques for repair.

Code: https://github.com/xuanbachle/syntax-guided-synthesis-repair