Fuzzing Class Specifications

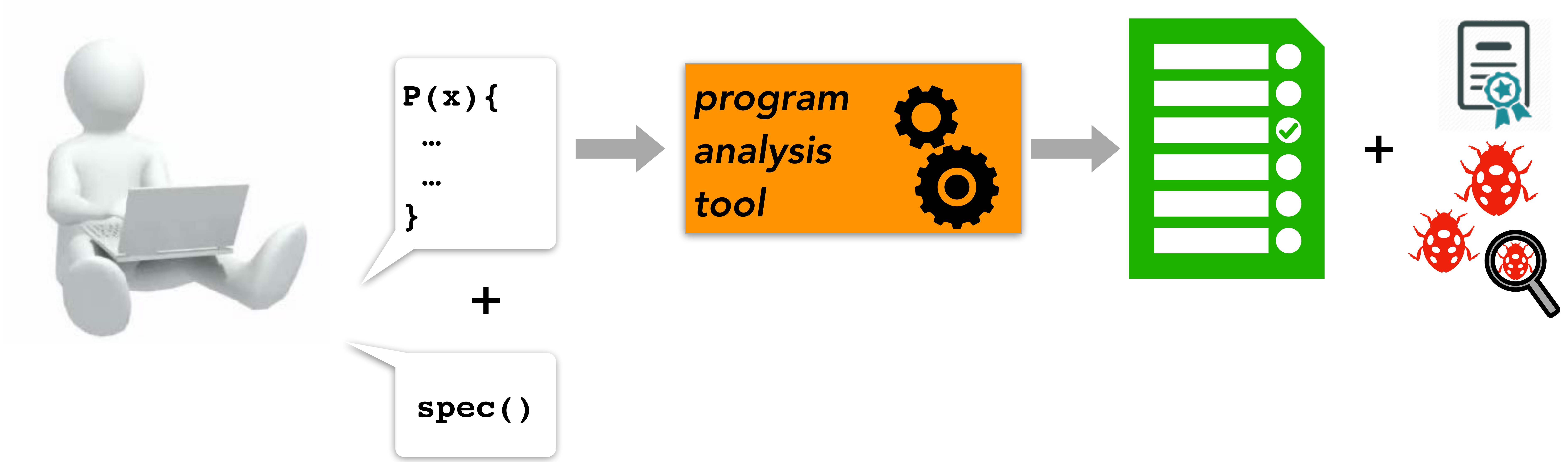
Facundo Molina

Department of Computing, University of Río Cuarto, Argentina
CONICET, Argentina

in collaboration with Marcelo d'Amorim and Nazareno Aguirre

An Automated Analysis Scenario

improved analysis in the presence of specifications



```
P(x){
  …
  …
}
```

+

```
spec()
```

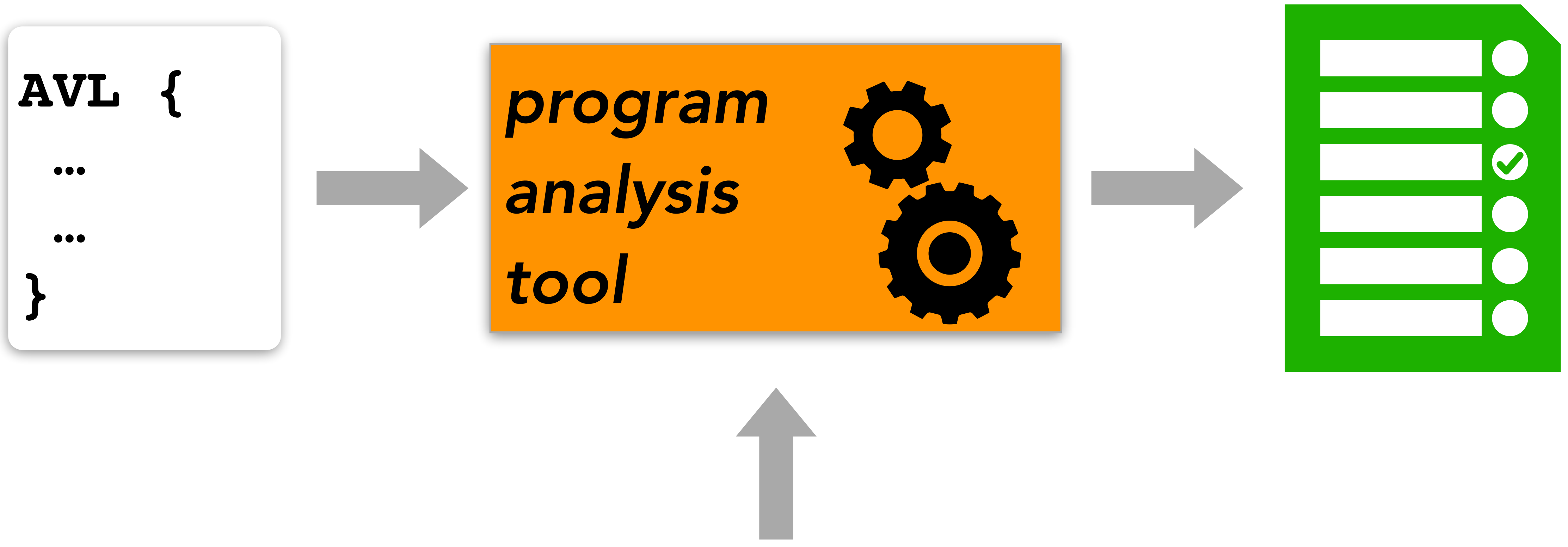
program analysis tool

+

unfortunately, specifications are seldom available

This illustrates the relevance of the ***oracle problem***

Approaches to the Oracle Problem



Manual specification

Analyze software crashes or general software faults

Derive specifications from existing software elements

Specification

Specified oracles

?

Implicit oracles

Derived oracles

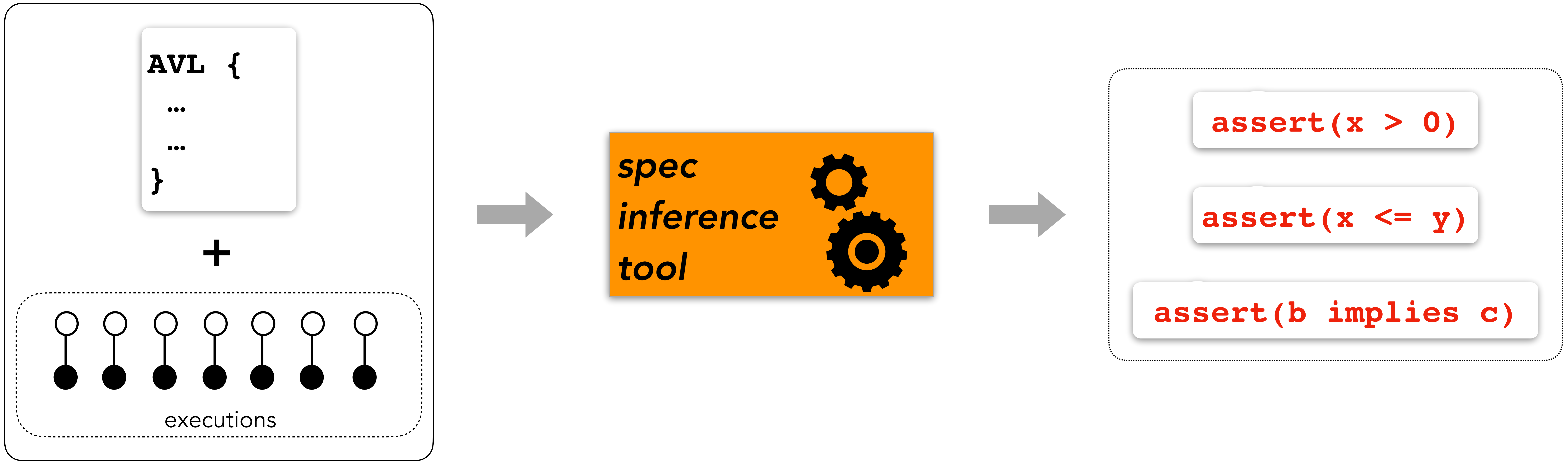
```
/*@
  @ invariant (\forall Node n;
  @    \reach(root, Node, left + right).has(n) implies
  @    ! \reach(n.right, Node, right + left).has(n) &&
  @    ! \reach(n.left, Node, left + right).has(n));
  @*/
```

```
try {
  …
} catch (NullPointerException e) {
  …
}
```

AVL ≡ AVL’

Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. **The Oracle Problem in Software Testing: A Survey**. TSE 2015

Specification Inference



```
AVL {
  …
  …
}
```

+

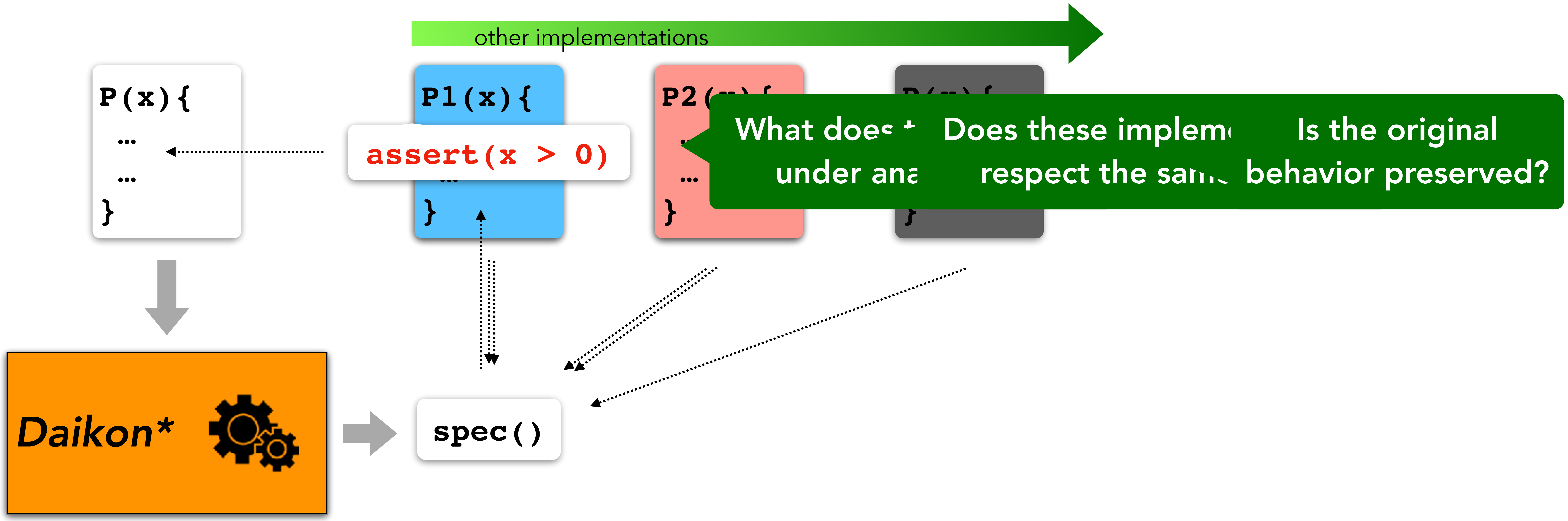
executions

spec inference tool

```
assert(x > 0)
assert(x <= y)
assert(b implies c)
```

Infer a *spec* capturing the current program behavior

Applications of Inferred Specs



Regression Testing

Differential Testing

Program Comprehension

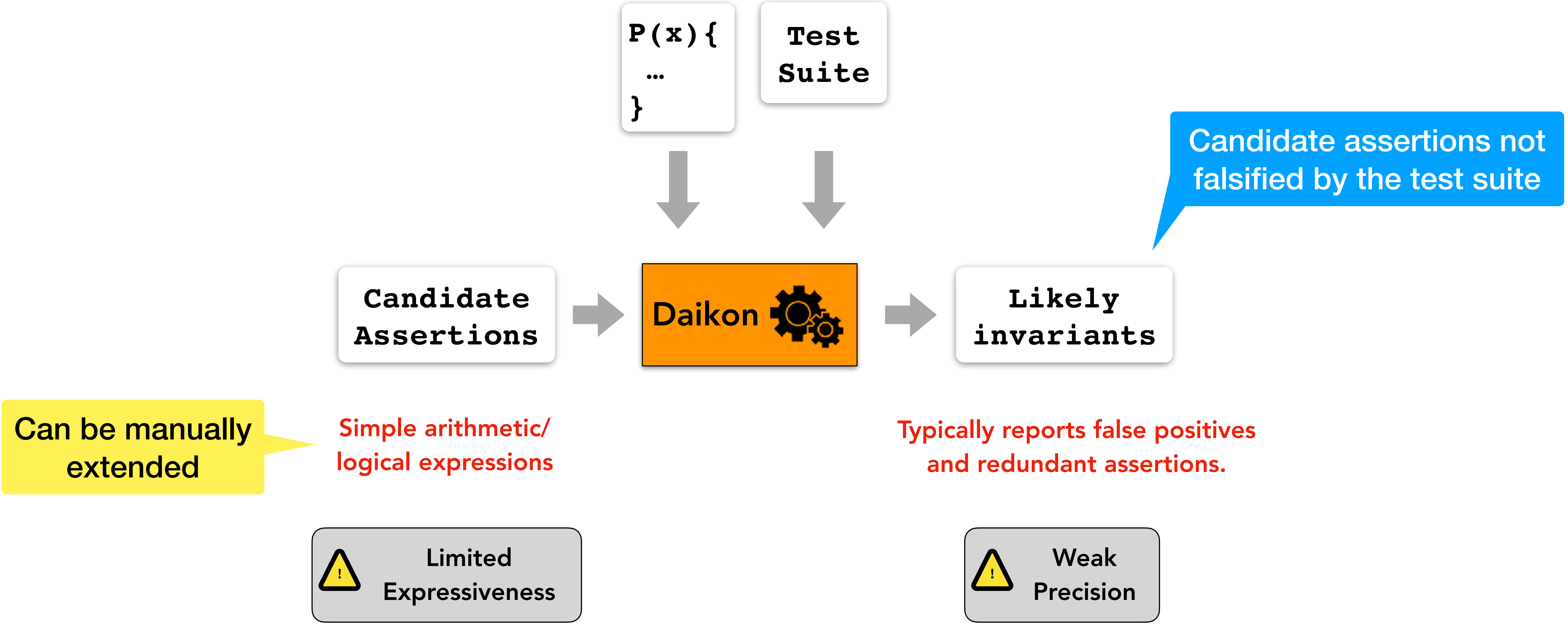
Program Repair

Fault Localization

…

* Michael D. Ernst et al. **The Daikon System for Dynamic Detection of Likely Invariants.** SCP 2007.

Dynamic Invariant Detection (in a nutshell)



P(x){
…
}

Test Suite

Candidate Assertions

Daikon

Likely invariants

Candidate assertions not falsified by the test suite

Can be manually extended

Simple arithmetic/ logical expressions

Typically reports false positives and redundant assertions.

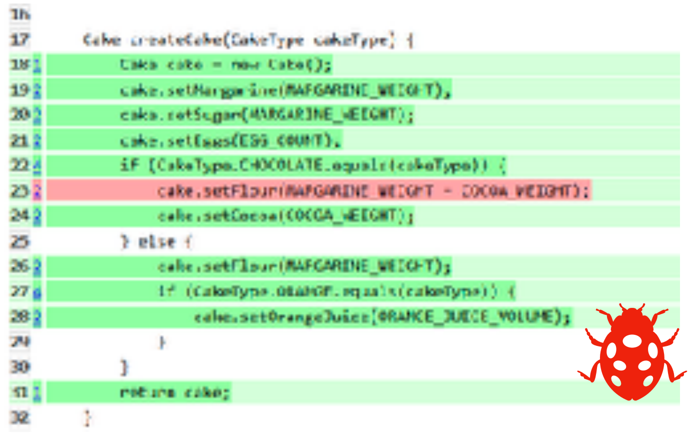
⚠ Limited Expressiveness

⚠ Weak Precision

Michael D. Ernst et al. **The Daikon System for Dynamic Detection of Likely Invariants.** SCP 2007.

Specification Inference Techniques

	Daikon	GAssert	EvoSpex
underlying technique	dynamic analysis	co-evolutionary algorithm	single-objective evolutionary algorithm

assertion strengthening approach	a == b ✓
	a >= b ✗
	ad-hoc template based approach
		
		code mutation
			x = 0, y = 3
			x = 0, y = 2
			state mutation

assertion language	+ - * / < > <= >= != == AND OR IMPLIES EQUIV Standard (extensible)	+ - * / % < > <= >= != == AND OR XOR EXOR IMPLIES EQUIV Arithmetic/Logical	+ - < > <= >= != == AND IMPLIES REACH HAS EXISTS FORALL Navigational/Quantification

```
    Cake createCake(CakeType cakeType) {
        Cake cake = new Cake();
        cake.setMargarine(MARGARINE_WEIGHT);
        cake.setSugar(MARGARINE_WEIGHT);
        cake.setEggs(EGG_COUNT);
        if (CakeType.CHOCOLATE.equals(cakeType)) {
            cake.setFlour(MARGARINE_WEIGHT - COCOA_WEIGHT);
            cake.setCocoa(COCOA_WEIGHT);
        } else {
            cake.setFlour(MARGARINE_WEIGHT);
            if (CakeType.ORANGE.equals(cakeType)) {
                cake.setOrangeJuice(ORANGE_JUICE_VOLUME);
            }
        }
        return cake;
    }
```

Valerio Terragni, Gunel Jahangirova, Paolo Tonella, and Mauro Pezzè. **Evolutionary Improvement of Assertion Oracles.** FSE 2020

Facundo Molina, Pablo Ponzio, Nazareno Aguirre and Marcelo F. Frias. **EvoSpex: An Evolutionary Algorithm for Learning Postconditions.** ICSE 2021

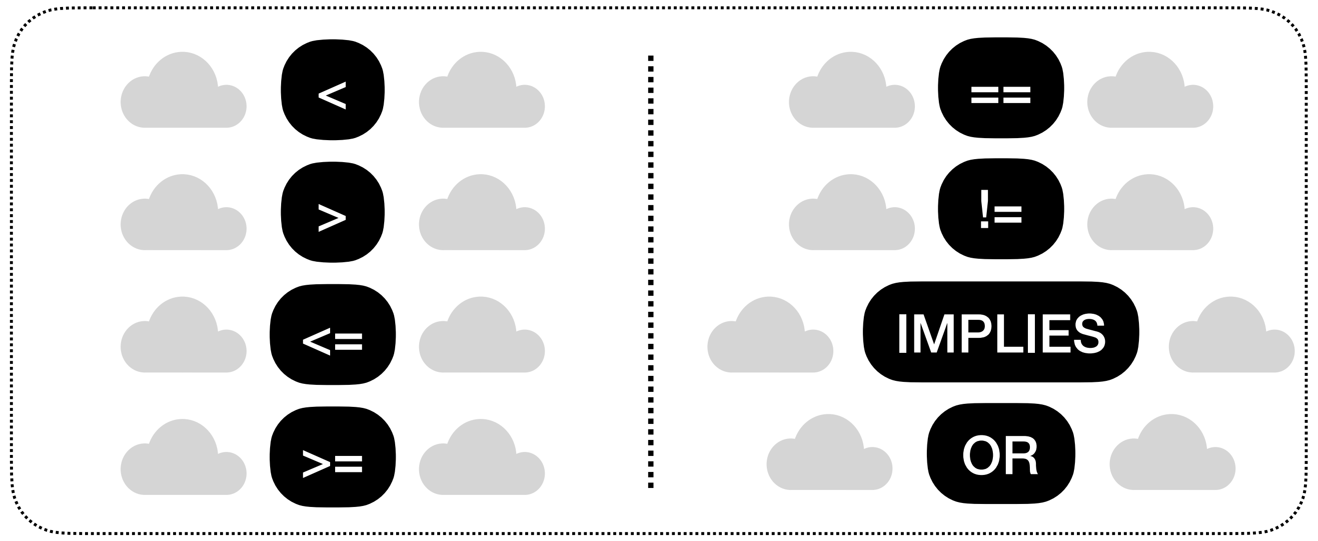
Improvement over Dynamic Invariant Detection

Daikon

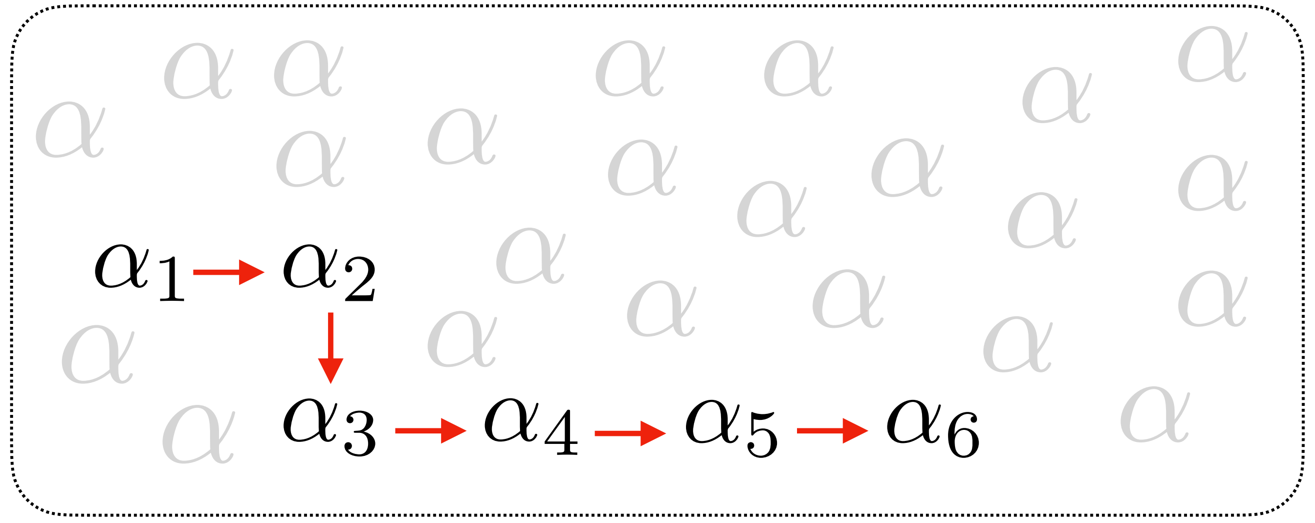
Evolutionary Approaches

Expressiveness

Standard Assertion Templates

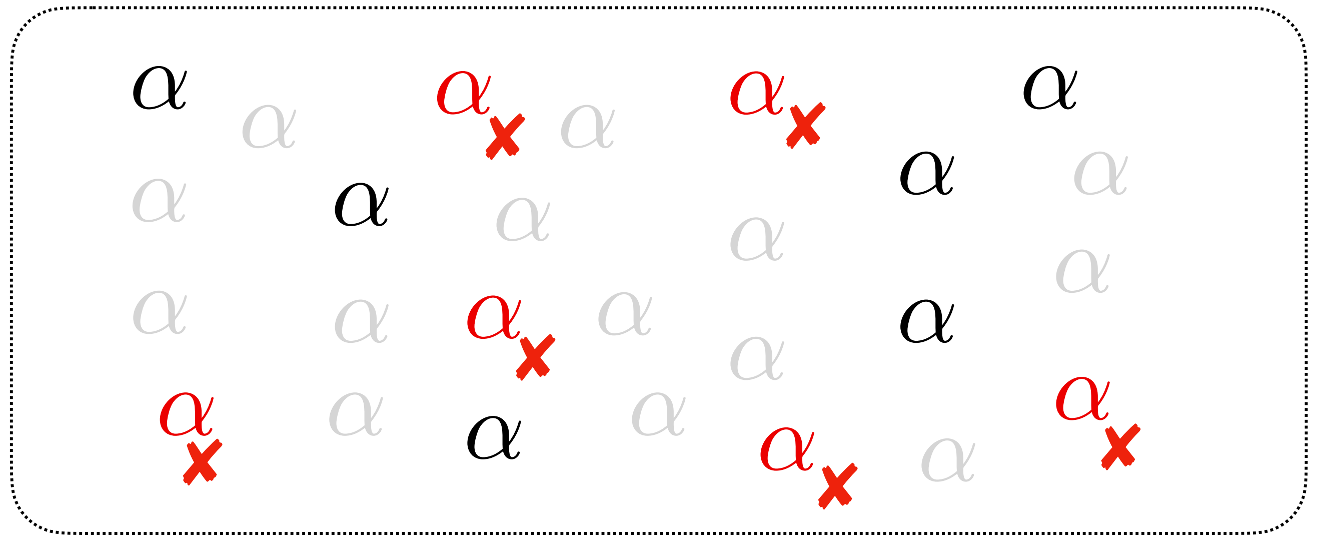


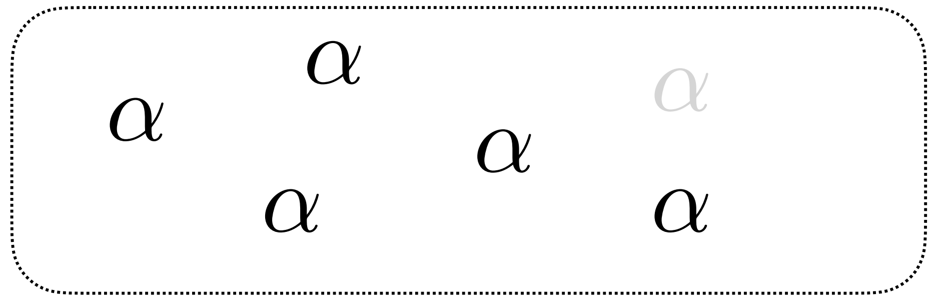
Assertion Search Space


$\alpha_1 \rightarrow \alpha_2$
$\alpha_3 \rightarrow \alpha_4 \rightarrow \alpha_5 \rightarrow \alpha_6$

Guided exploration of a larger search space

Precision





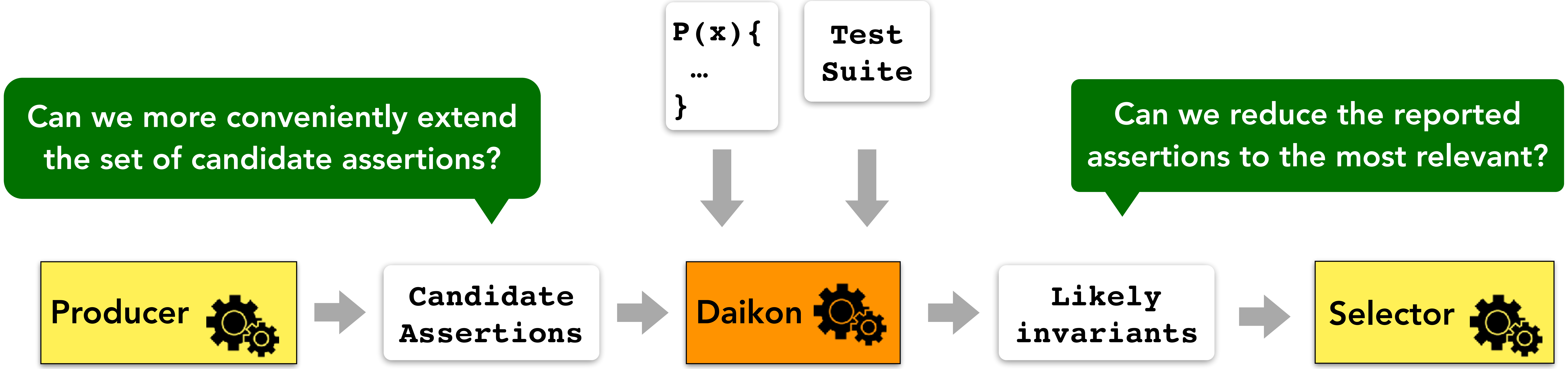
Report less and stronger assertions

Can we improve Daikon to overcome the above limitations?

Evolutionary approaches are expensive

Assertion languages that are difficult to adapt to support further assertions

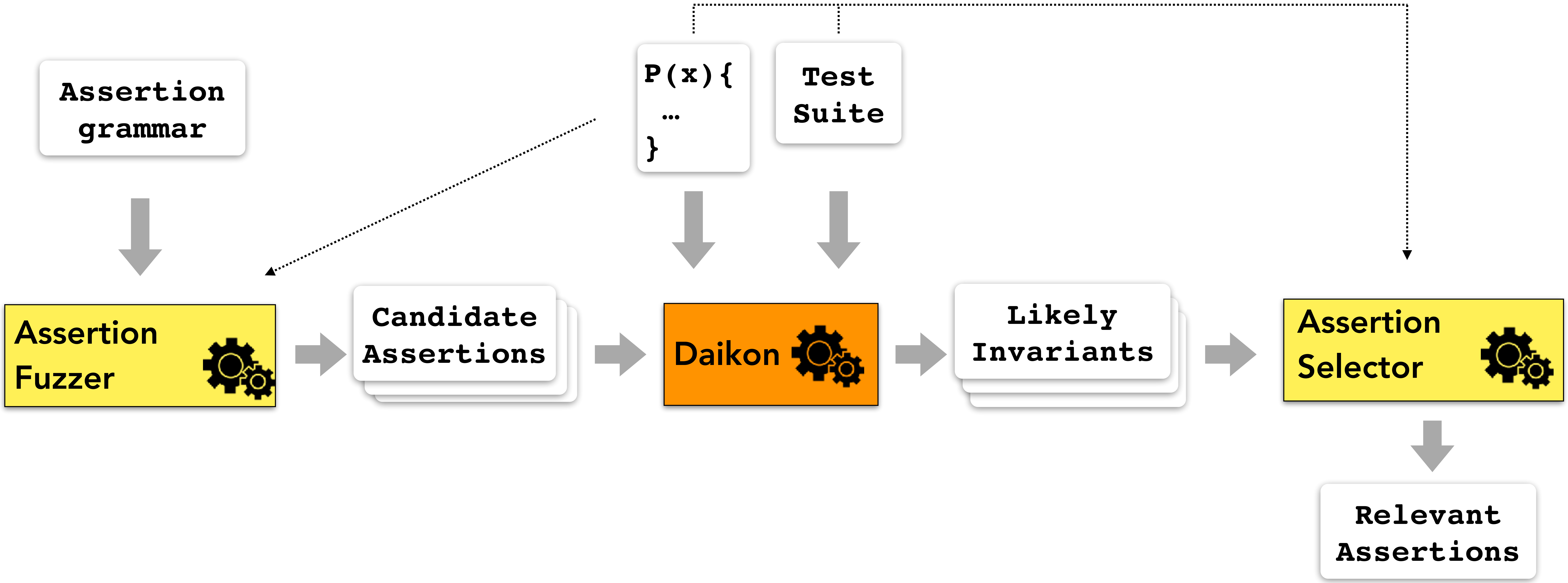
Improving Dynamic Invariant Detection


Can we more conveniently extend the set of candidate assertions?
P(x){
…
}
Test Suite
Can we reduce the reported assertions to the most relevant?
Producer
Candidate Assertions
Daikon
Likely invariants
Selector

Michael D. Ernst et al. **The Daikon System for Dynamic Detection of Likely Invariants.** SCP 2007.

Fuzzing Class Specifications

SpecFuzzer

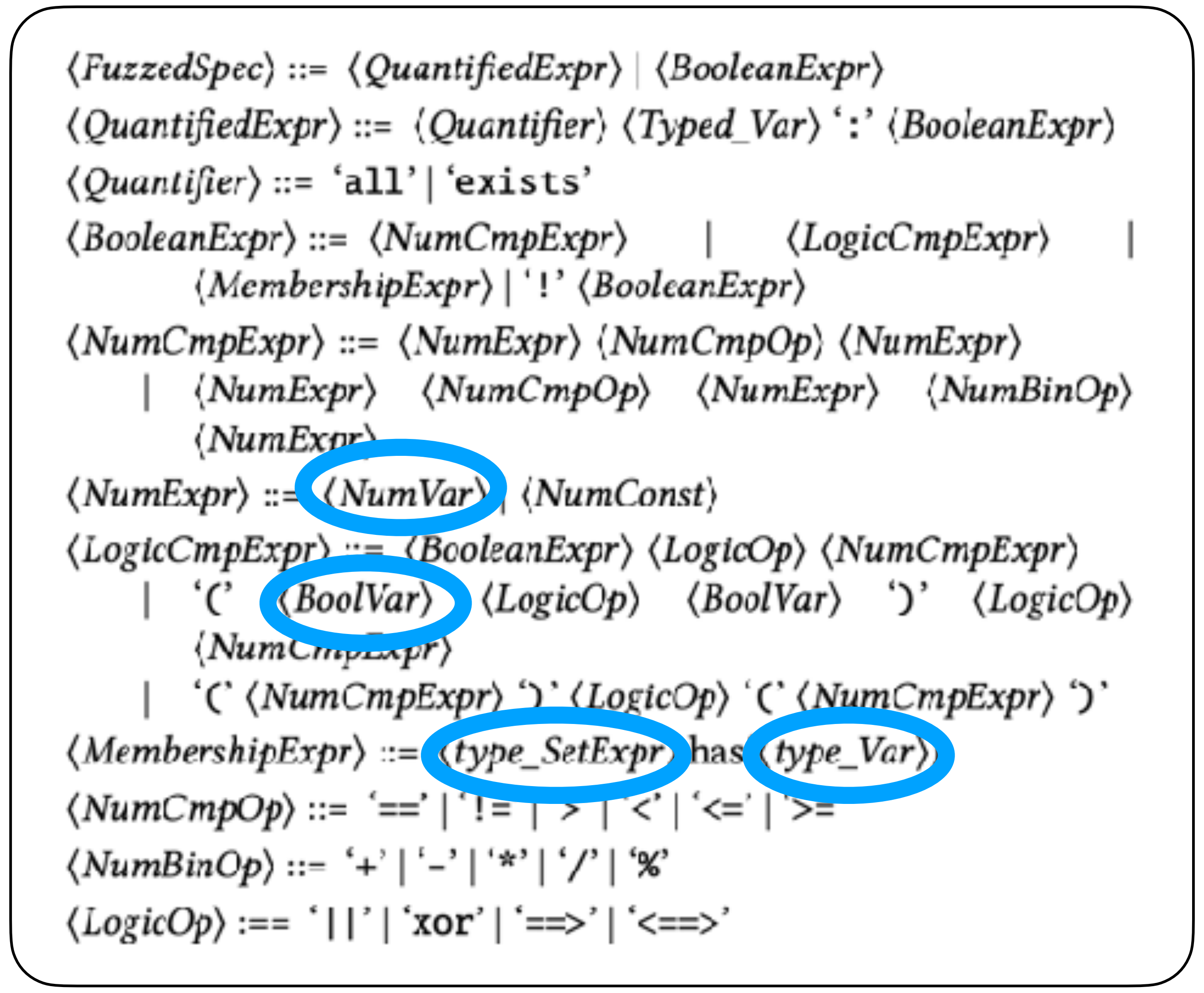


Remove the need of manually specifying new assertions

Report only the assertions that are more relevant

The Assertion Fuzzer

Base Assertion Grammar



```
⟨FuzzedSpec⟩ ::= ⟨QuantifiedExpr⟩ | ⟨BooleanExpr⟩
⟨QuantifiedExpr⟩ ::= ⟨Quantifier⟩ ⟨Typed_Var⟩ ':' ⟨BooleanExpr⟩
⟨Quantifier⟩ ::= 'all' | 'exists'
⟨BooleanExpr⟩ ::= ⟨NumCmpExpr⟩ | ⟨LogicCmpExpr⟩ |
        ⟨MembershipExpr⟩ | '!' ⟨BooleanExpr⟩
⟨NumCmpExpr⟩ ::= ⟨NumExpr⟩ ⟨NumCmpOp⟩ ⟨NumExpr⟩
    | ⟨NumExpr⟩ ⟨NumCmpOp⟩ ⟨NumExpr⟩ ⟨NumBinOp⟩
        ⟨NumExpr⟩
⟨NumExpr⟩ ::= ⟨NumVar⟩ | ⟨NumConst⟩
⟨LogicCmpExpr⟩ ::= ⟨BooleanExpr⟩ ⟨LogicOp⟩ ⟨NumCmpExpr⟩
    | '(' ⟨BoolVar⟩ ⟨LogicOp⟩ ⟨BoolVar⟩ ')' ⟨LogicOp⟩
        ⟨NumCmpExpr⟩
    | '(' ⟨NumCmpExpr⟩ ')' ⟨LogicOp⟩ '(' ⟨NumCmpExpr⟩ ')'
⟨MembershipExpr⟩ ::= ⟨type_SetExpr⟩ has ⟨type_Var⟩
⟨NumCmpOp⟩ ::= '==' | '!=' | '>' | '<' | '<=' | '>='
⟨NumBinOp⟩ ::= '+' | '-' | '*' | '/' | '%'
⟨LogicOp⟩ :== '||' | 'xor' | '==>' | '<==>'
```

Can be straightforwardly adapted

Target Class

```
public class C {
  int x, y;
  boolean b, c;
  Set<Integer> s;
}
```

Assertion Fuzzer

Candidate Assertions

this.x > this.y * -1

this.y >= 0

this.x > 0 -> this.y < 0

this.x > 0

this.x % this.y > 0

this.c <=> this.b

this.x >= this.y

this.b <=> (this.x != this.y)

this.b <=> (this.x == this.y)

this.x + this.y > 1

this.b ll this.x > this.y

this.x > this.y

this.b -> (this.x < this.y)

this.x < this.y + 1

exists n : reach(this.header, Node, next) : n.value > 0

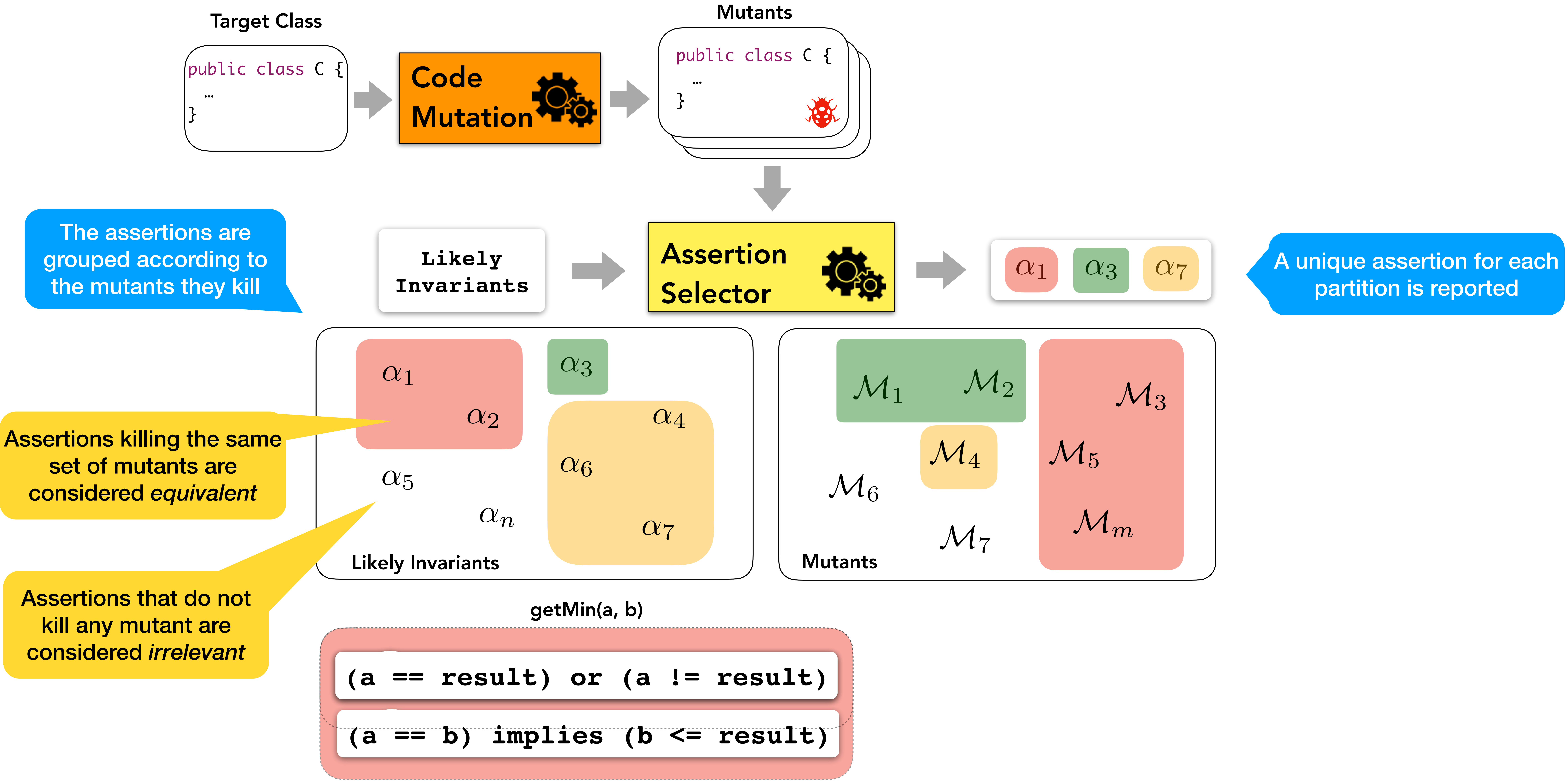
forall n : reach(this.header, Node, next) : n.value < n.next.value

this.x < this.y

exists n : reach(this.header, Node, next) : b -> n.value < 0

Assertions fuzzing

The Assertion Selector



Target Class

```
public class C {
  …
}
```

Code Mutation

Mutants

```
public class C {
  …
}
```

The assertions are grouped according to the mutants they kill

Likely Invariants

Assertion Selector

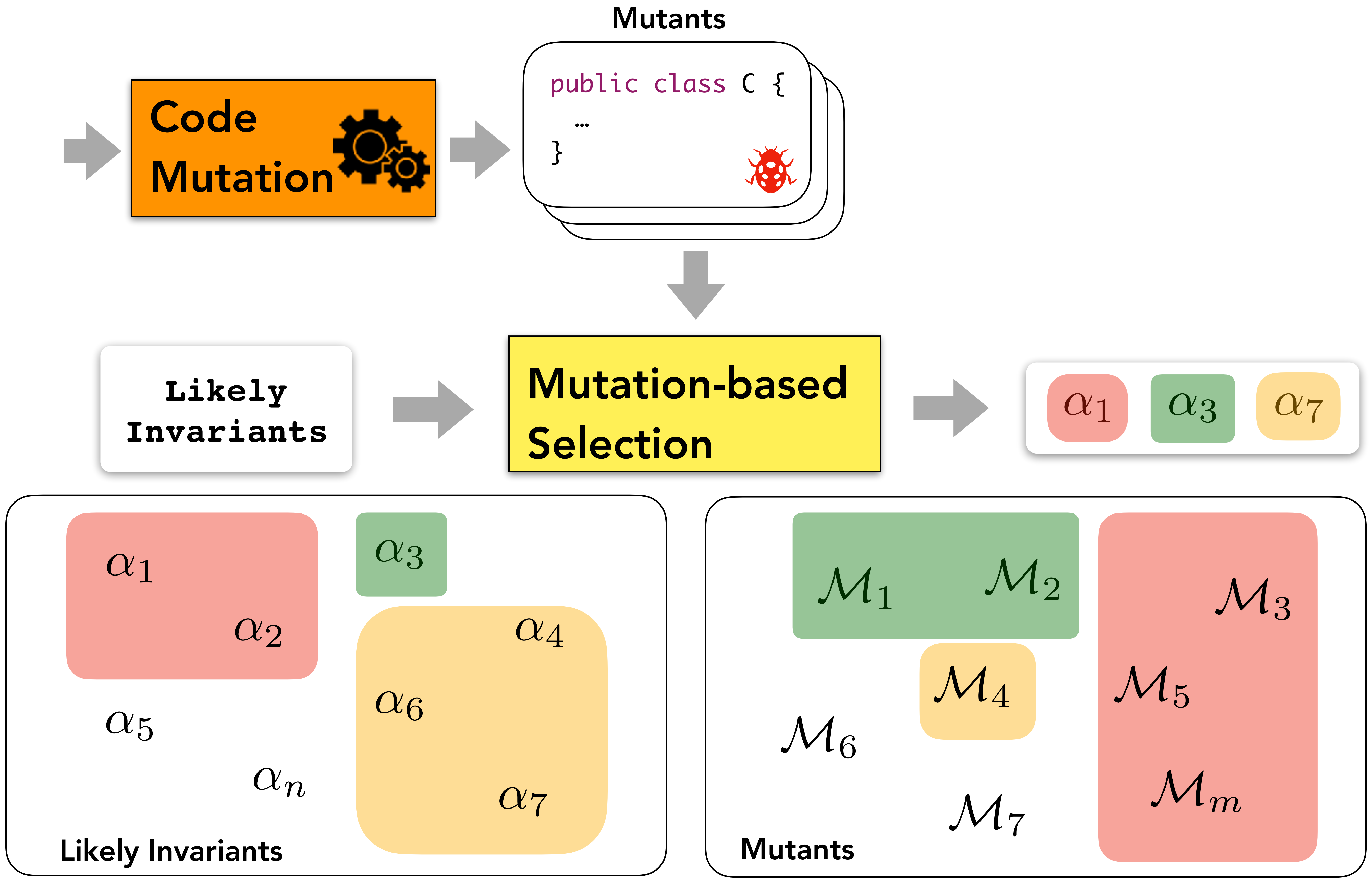
A unique assertion for each partition is reported

Assertions killing the same set of mutants are considered *equivalent*

Assertions that do not kill any mutant are considered *irrelevant*

getMin(a, b)

```
(a == result) or (a != result)
(a == b) implies (b <= result)
```



Mutants

```
public class C {
  …
}
```

Code Mutation

Likely Invariants

Mutation-based Selection

α_1 α_3 α_7

α_1 α_2 α_3 α_4 α_5 α_6 α_n α_7

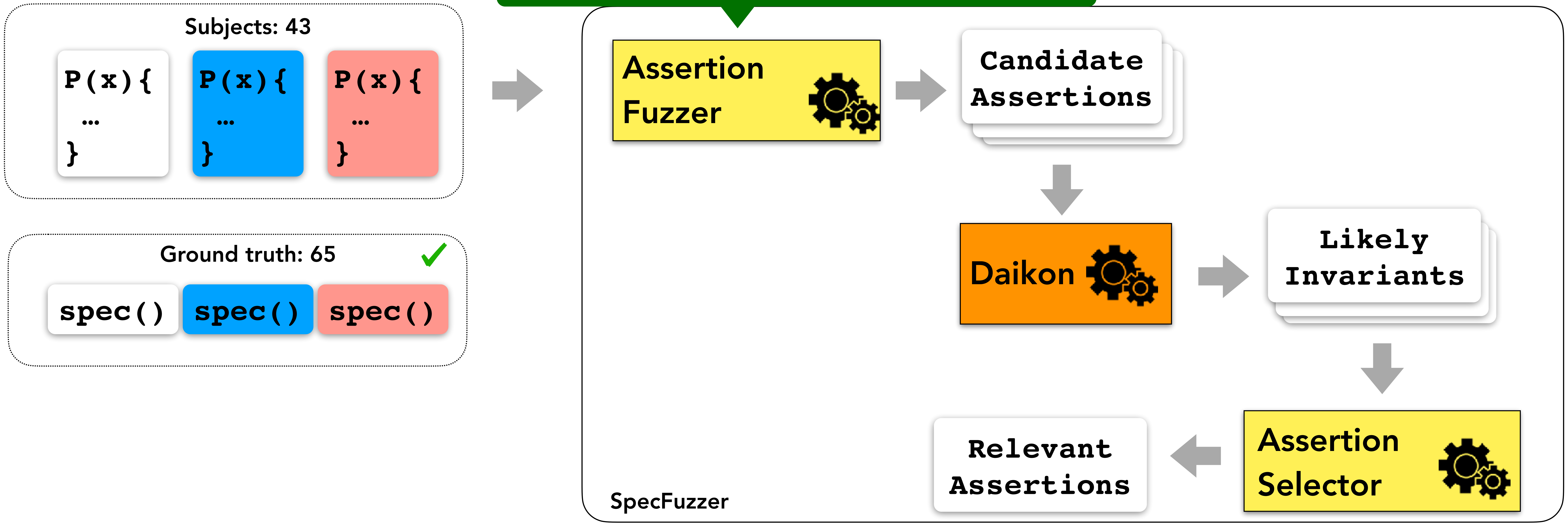
Likely Invariants

$\mathcal{M}_1$ $\mathcal{M}_2$ $\mathcal{M}_3$ $\mathcal{M}_4$ $\mathcal{M}_5$ $\mathcal{M}_6$ $\mathcal{M}_7$ $\mathcal{M}_m$

Mutants

Experimental Setup

Is grammar-based fuzzing effective at generating relevant assertions?



Subjects: 43

P(x){ … }

P(x){ … }

P(x){ … }

Ground truth: 65 ✓

spec()

spec()

spec()

Assertion Fuzzer

Candidate Assertions

Daikon

Likely Invariants

Assertion Selector

Relevant Assertions

SpecFuzzer

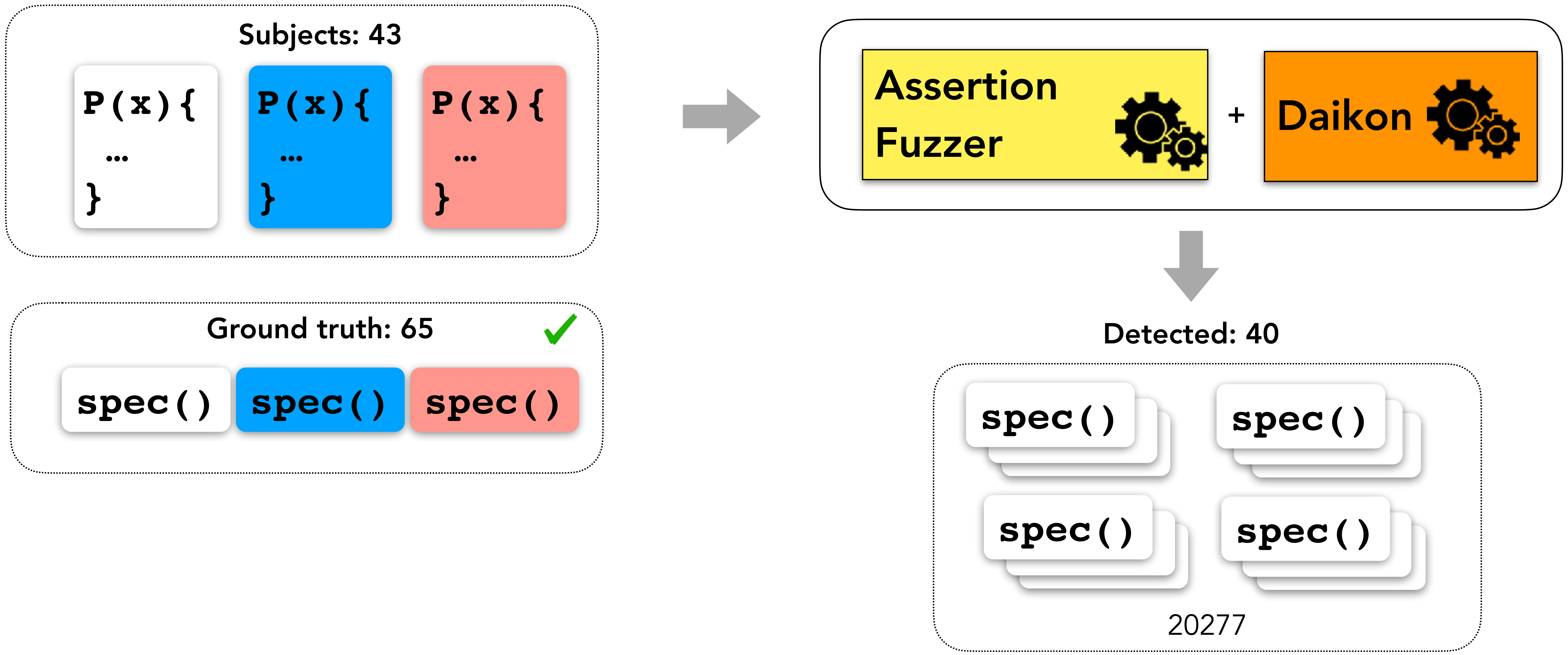
How does SpecFuzzer compare with alternative techniques?

Is the mutation-based selector successful for removing redundant/irrelevant assertions?

Valerio Terragni, Gunel Jahangirova, Paolo Tonella, and Mauro Pezzè. **Evolutionary Improvement of Assertion Oracles.** FSE 2020

Facundo Molina, Pablo Ponzio, Nazareno Aguirre, and Marcelo Frias. **EvoSpex: An Evolutionary Algorithm for Learning Postconditions.** ICSE 2021.

Effectiveness of Grammar-based Fuzzing



Subjects: 43

P(x){ … }

P(x){ … }

P(x){ … }

Ground truth: 65 ✓

spec() spec() spec()

Assertion Fuzzer + Daikon

Detected: 40

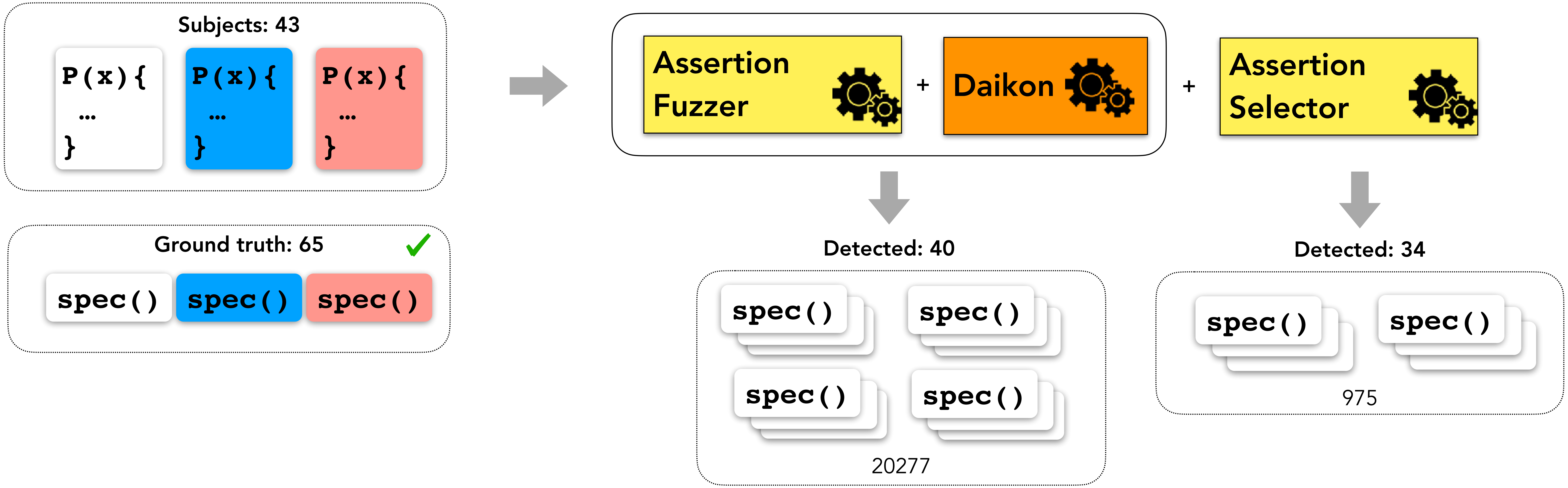
spec() spec()

spec() spec()

20277

The assertion fuzzer allowed us to detect 61% of the ground truth assertions.

Performance of the Assertion Selector



Subjects: 43

P(x){ … }

P(x){ … }

P(x){ … }

Ground truth: 65 ✓

spec() spec() spec()

Assertion Fuzzer + Daikon + Assertion Selector

Detected: 40

spec() spec() spec() spec()

20277

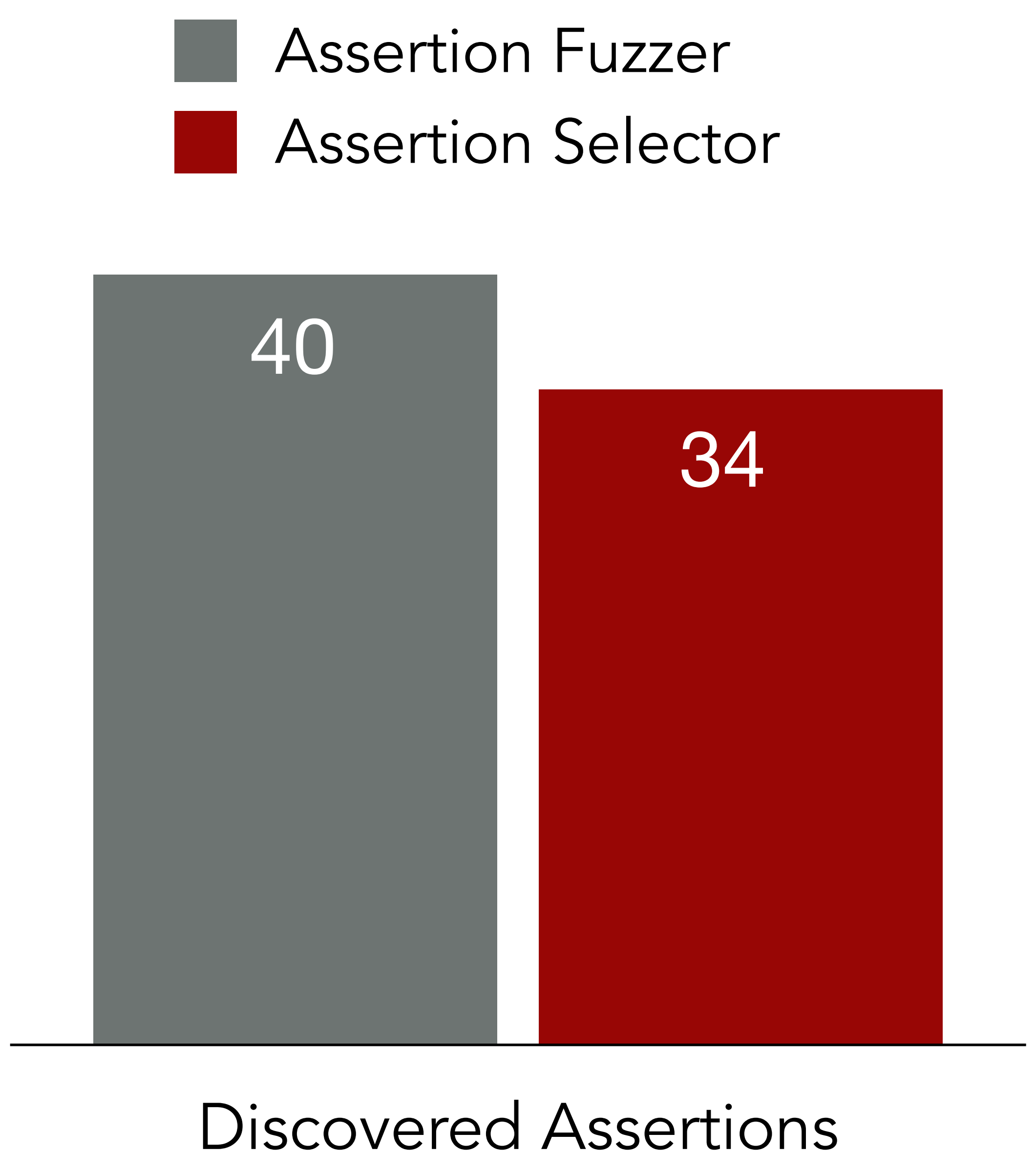
Detected: 34

spec() spec()

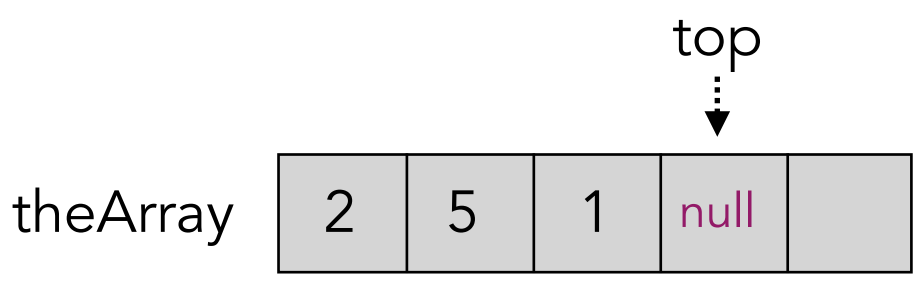
975

The Assertion Selector reduced the reported assertions by 95%, allowing us to discover 52% of the ground truth assertions.

Performance of the Assertion Selector



StackAr.pop



```
assert(theArray[old(top)] == null)
```

No mutant modifies the null value after the pop array update

Angle.getTurn

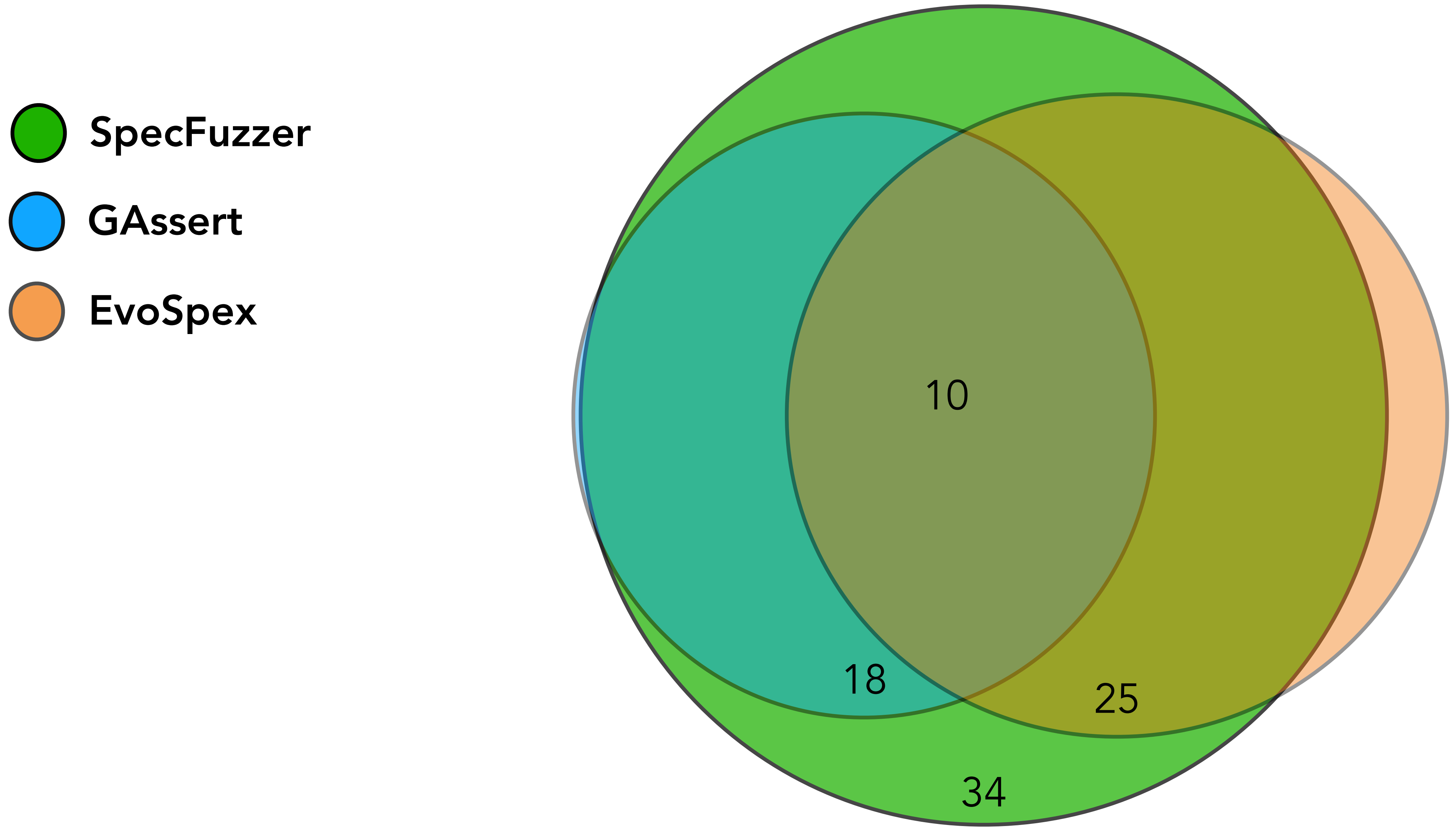
```
crossproduct = Math.sin(ang2 - ang1);
if (crossproduct > 0)
   return 1;
if (crossproduct < 0)
   return -1;

return 0;
```

```
assert(abs(result) <= 1)
```

No mutant makes the method return a value other than 1, -1 or 0

The effectiveness of the Assertion Selector may be improved considering further mutation operators

SpecFuzzer vs Evolutionary Approaches



SpecFuzzer missed 1 assertion produced by GAssert

SpecFuzzer missed 8 assertions produced by EvoSpex

GAssert - 27%

EvoSpex - 38%

SpecFuzzer - 52%

Valerio Terragni, Gunel Jahangirova, Paolo Tonella, and Mauro Pezzè. **Evolutionary Improvement of Assertion Oracles.** FSE 2020

Facundo Molina, Pablo Ponzio, Nazareno Aguirre, and Marcelo Frias. **EvoSpex: An Evolutionary Algorithm for Learning Postconditions.** ICSE 2021.

Remarks

- The Oracle Problem is a relevant problem in Software Engineering.

- SpecFuzzer uses grammar-based fuzzing and mutation-based selection to effectively improve dynamic invariant detection.

- Specification Inference can still be improved.

Scalability **Expressiveness** **Precision**

Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. **The Oracle Problem in Software Testing: A Survey**. TSE 2015