Scalable SMT Sampling for Floating-Point Formulas via Coverage-Guided Fuzzing

Manuel Carrasco, Cristian Cadar and Alastair F. Donaldson
Imperial College London

ICST 2025

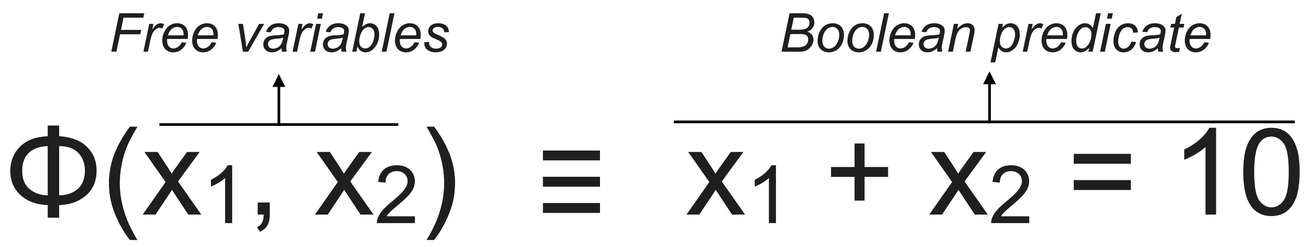
What is SMT Sampling?

What is SMT Sampling?

SMT formula $\Phi(x_1, x_2)$

What is SMT Sampling?

SMT formula

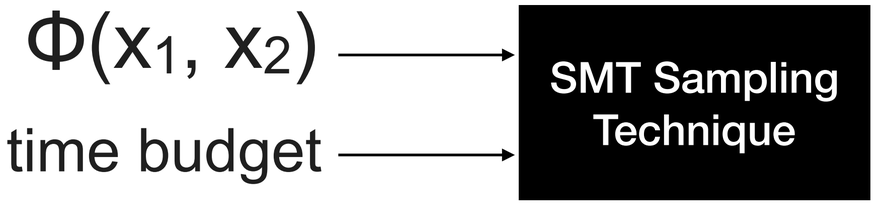


What is SMT Sampling?

SMT formula $\Phi(x_1, x_2) \equiv x_1 + x_2 = 10$

Free variables — x_1, x_2

Boolean predicate — $x_1 + x_2 = 10$



What is SMT Sampling?

SMT formula $\Phi(x_1, x_2) \equiv x_1 + x_2 = 10$

Free variables — x_1, x_2

Boolean predicate — $x_1 + x_2 = 10$

What is SMT Sampling?

SMT formula $\Phi(x_1, x_2) \equiv x_1 + x_2 = 10$

Free variables — x_1, x_2

Boolean predicate — $x_1 + x_2 = 10$

What is SMT Sampling?

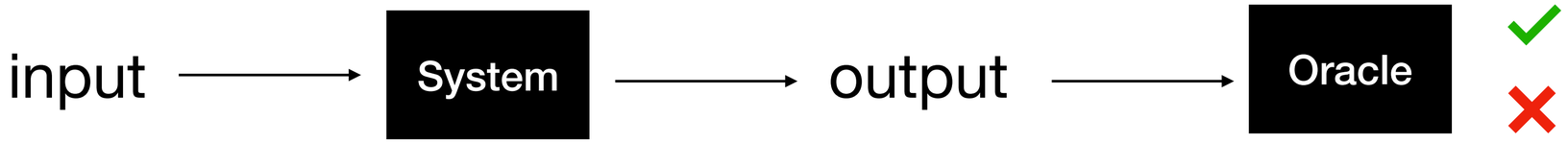
SMT formula $\Phi(x_1, x_2) \equiv x_1 + x_2 = 10$

Free variables — x_1, x_2

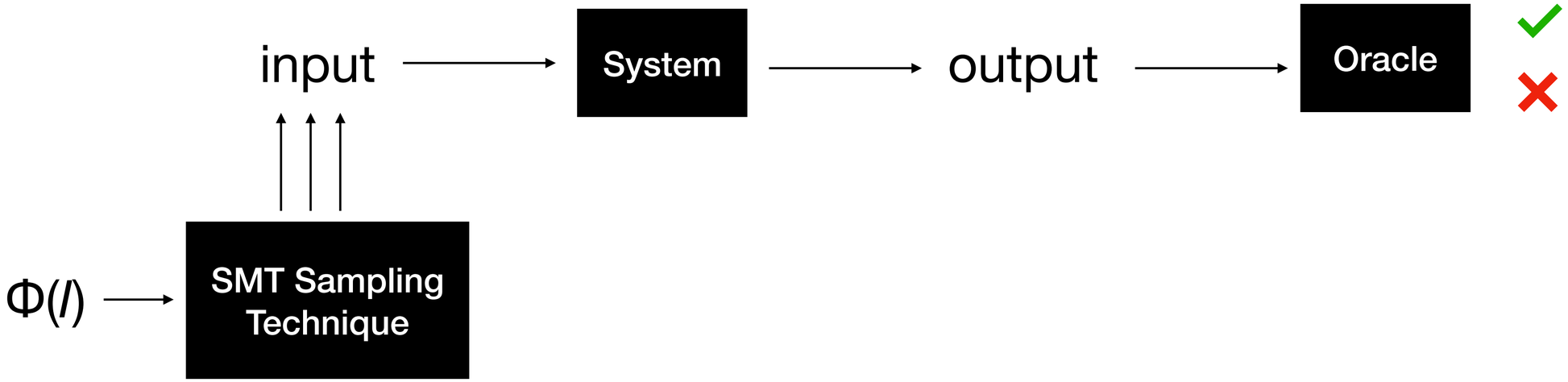
Boolean predicate — $x_1 + x_2 = 10$

Why SMT Sampling?

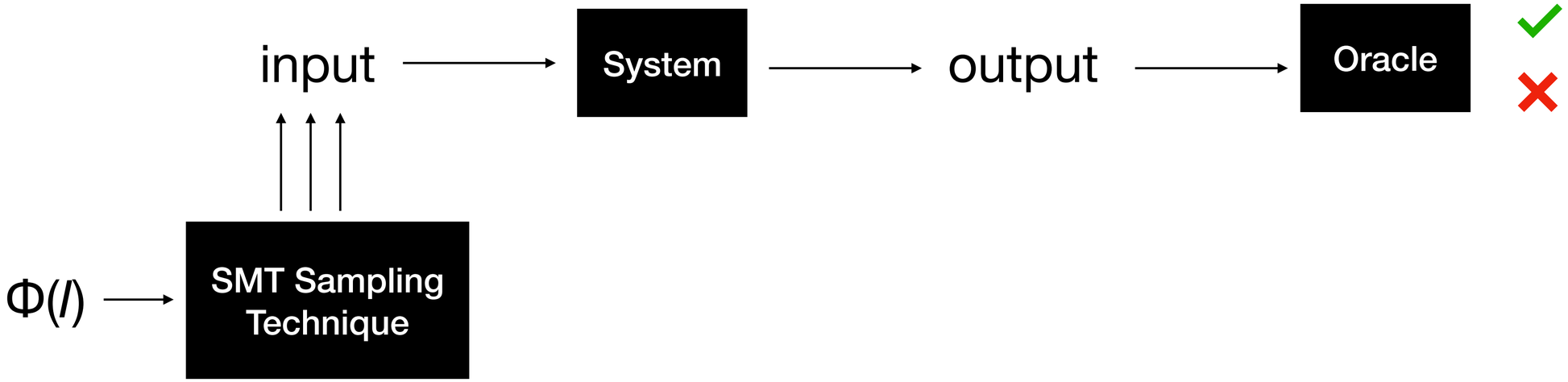
Why SMT Sampling?



Why SMT Sampling?



Why SMT Sampling?



- Φ(*I*) can be an input specification or any other testing property.

SMT Sampling Metrics

SMT Sampling Metrics

- Throughput: # satisfying assignments (samples) found in the time budget.

SMT Sampling Metrics

- Throughput: # satisfying assignments (samples) found in the time budget.
- Diversity: how well the samples represent the solution space.

SMT Sampling Metrics

- Throughput: # satisfying assignments (samples) found in the time budget.
- Diversity: how well the samples represent the solution space.
- These two metrics are in constant tension.

SMT Sampling Metrics

- Throughput: # satisfying assignments (samples) found in the time budget.
- Diversity: how well the samples represent the solution space.
- These two metrics are in constant tension.
- They are influenced by the underlying sampling algorithm!

Related Work: SMTSampler

SMTSampler: Efficient Stimulus Generation from Complex SMT Constraints

Rafael Dutra, Jonathan Bachrach and Koushik Sen
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
{rtd,jrb,ksen}@cs.berkeley.edu

Related Work: SMTSampler

SMTSampler: Efficient Stimulus Generation from Complex SMT Constraints

Rafael Dutra, Jonathan Bachrach and Koushik Sen
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
{rtd,jrb,ksen}@cs.berkeley.edu

- **SMTSampler** is a state-of-the-art SMT sampling technique.

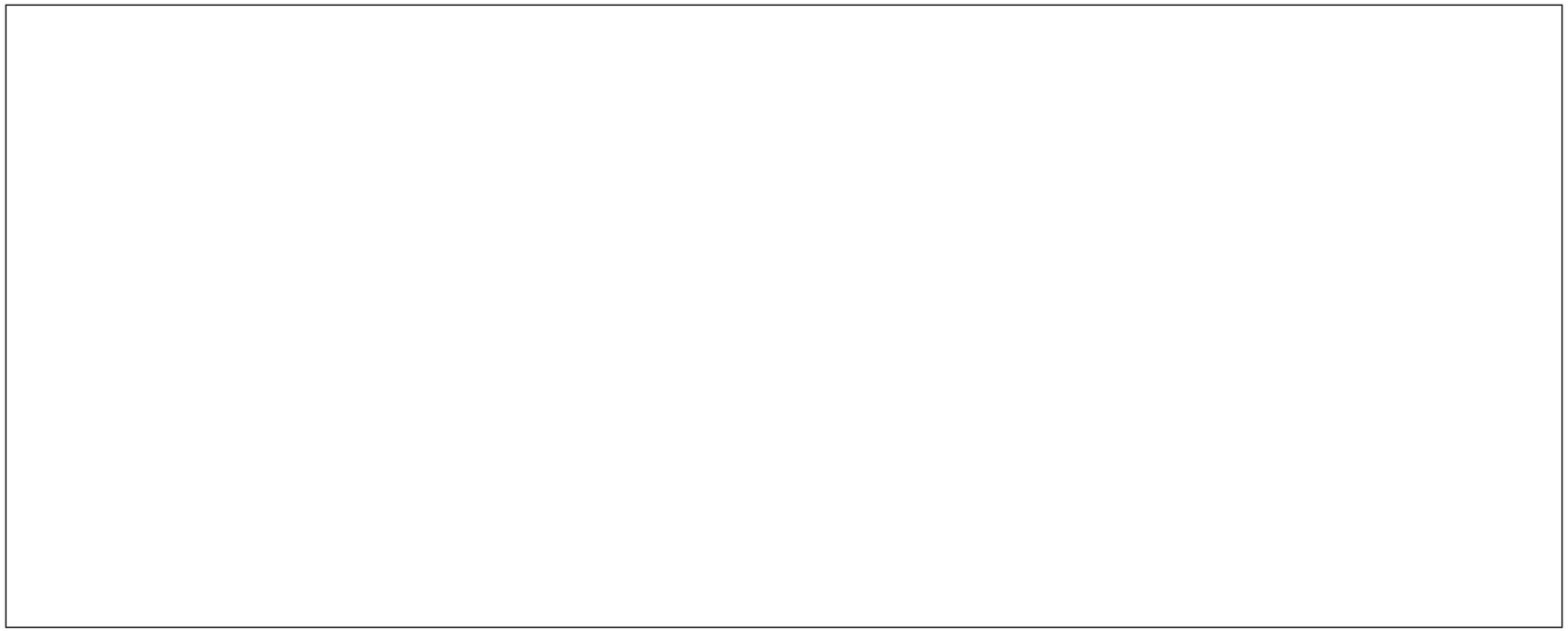
Related Work: SMTSampler

SMTSampler: Efficient Stimulus Generation from Complex SMT Constraints

Rafael Dutra, Jonathan Bachrach and Koushik Sen
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
{rtd,jrb,ksen}@cs.berkeley.edu

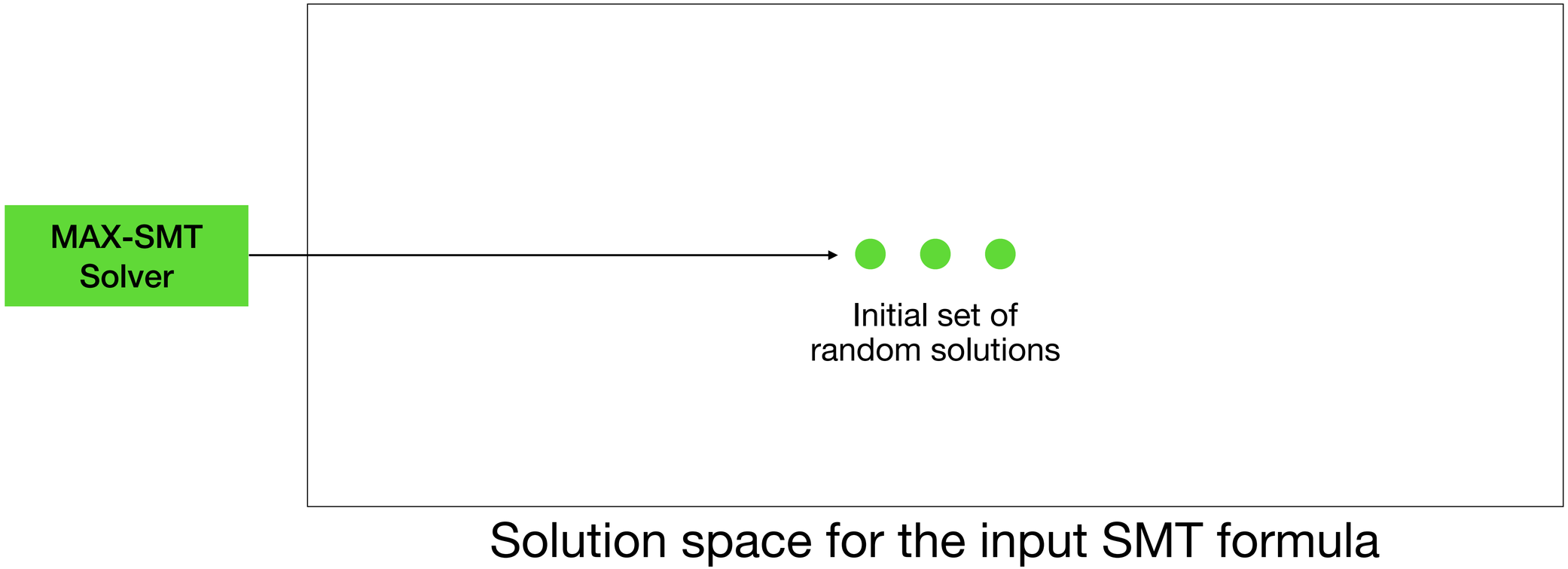
- **SMTSampler** is a state-of-the-art SMT sampling technique.
- It proved to have higher throughput when compared to other samplers.

SMTSampler’s Algorithm

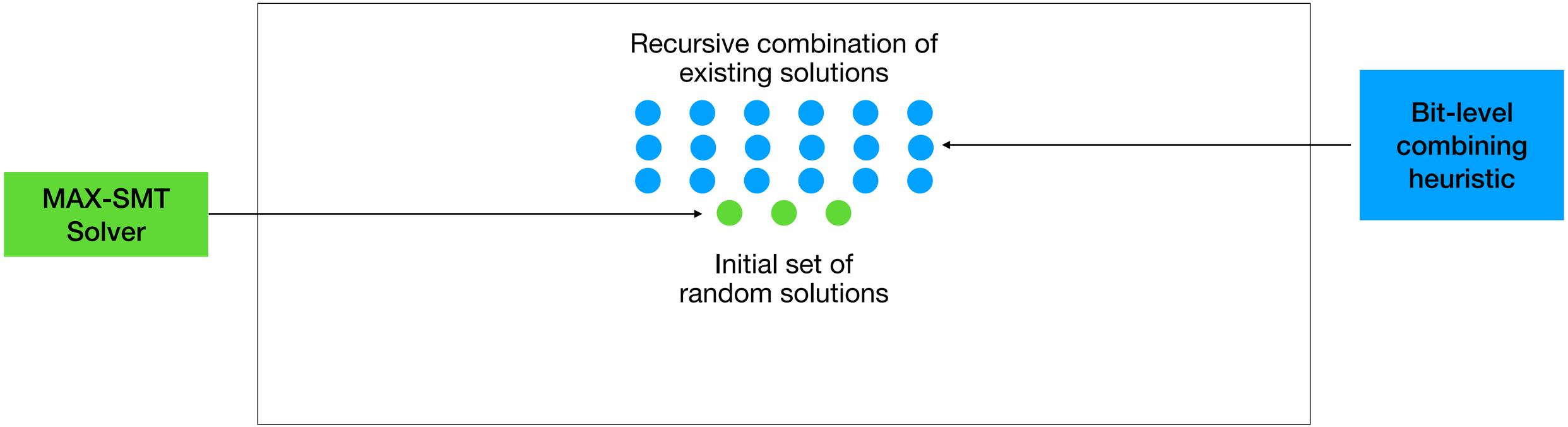


Solution space for the input SMT formula

SMTSampler's Algorithm



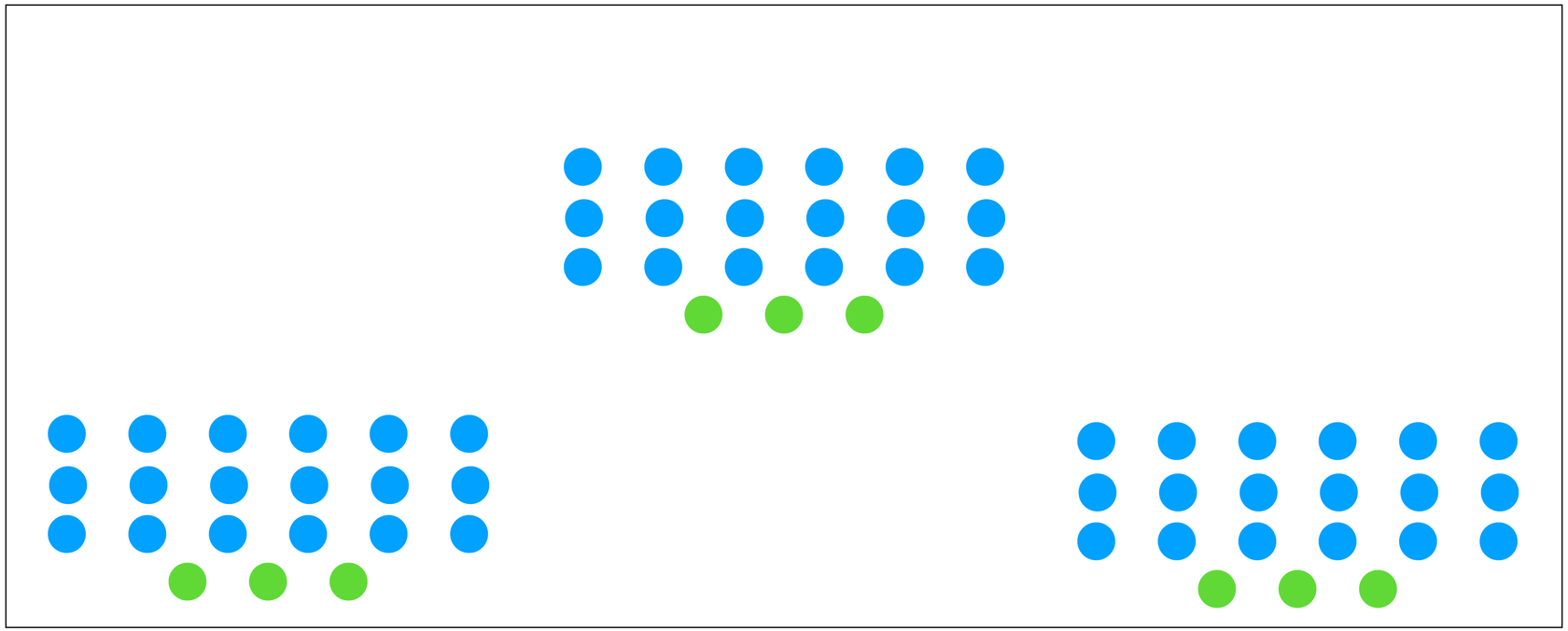
SMTSampler's Algorithm



Solution space for the input SMT formula

SMTSampler's Algorithm

MAX-SMT Solver



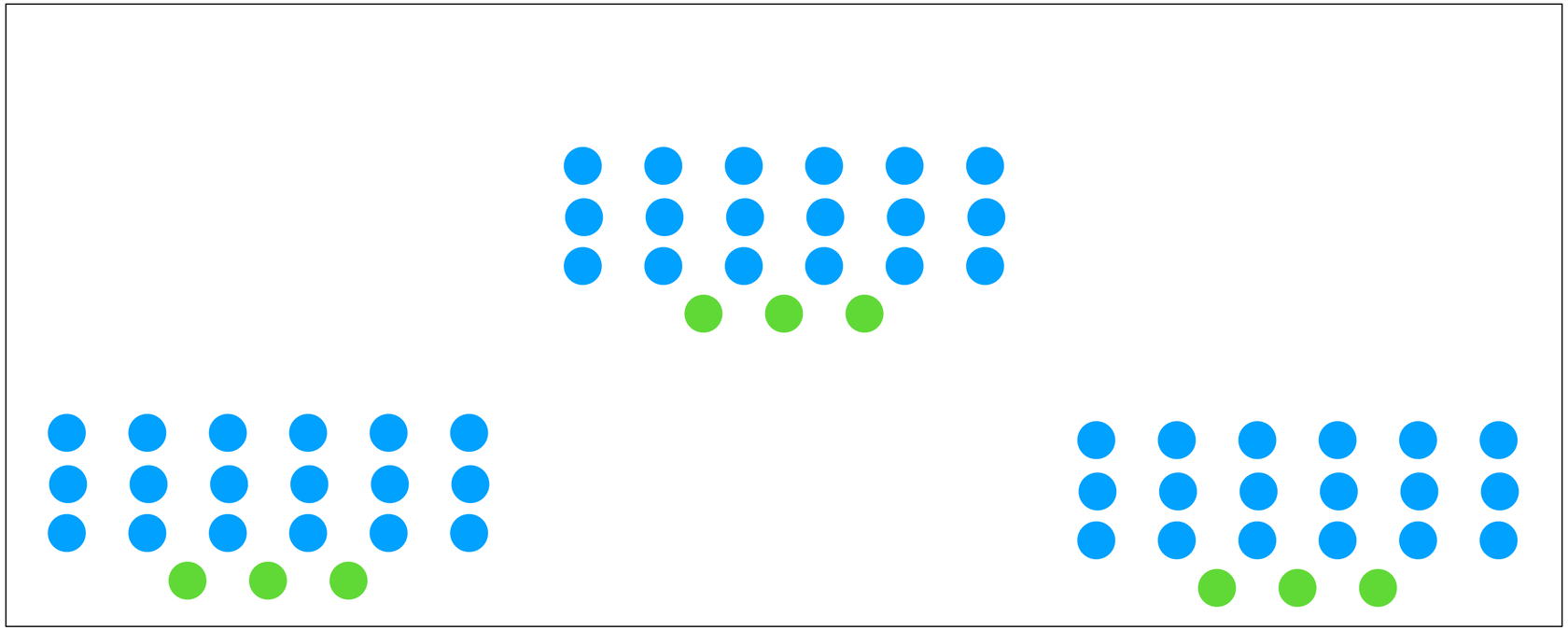
Bit-level combining heuristic

Solution space for the input SMT formula

SMTSampler's Algorithm

MAX-SMT Solver

Slow



Bit-level combining heuristic

Fast

Solution space for the input SMT formula

Floating-Point Formulas

Floating-Point Formulas

- This domain is often challenging for traditional SMT solvers.

Floating-Point Formulas

- This domain is often challenging for traditional SMT solvers.
- By design, **SMTSampler** inherits these scalability limitations.

Floating-Point Formulas

- This domain is often challenging for traditional SMT solvers.
- By design, **SMTSampler** inherits these scalability limitations.
- We want a scalable sampler for this domain.

Floating-Point Formulas

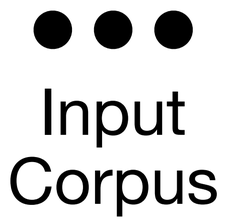
- This domain is often challenging for traditional SMT solvers.
- By design, **SMTSampler** inherits these scalability limitations.
- We want a scalable sampler for this domain.
- We leverage related work that uses coverage-guided fuzzing.

Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.

Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.

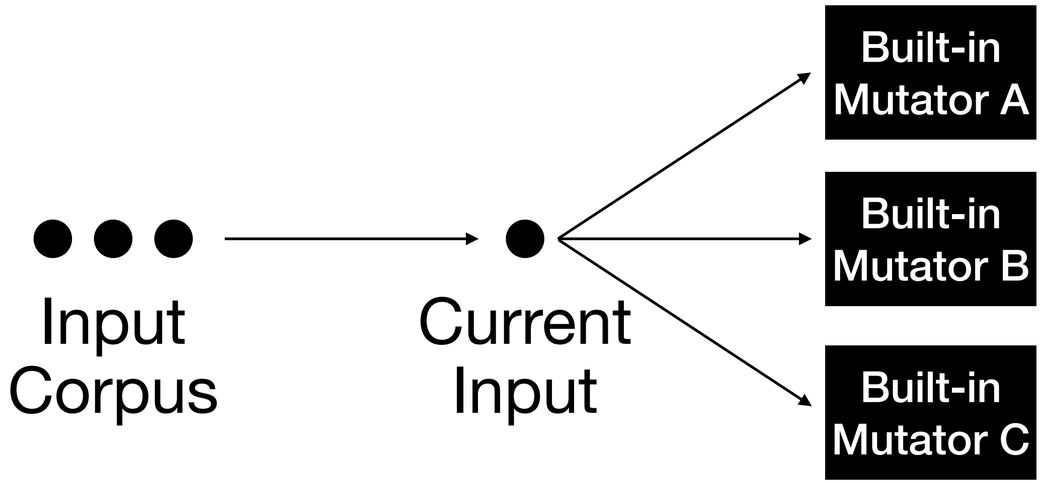


Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.

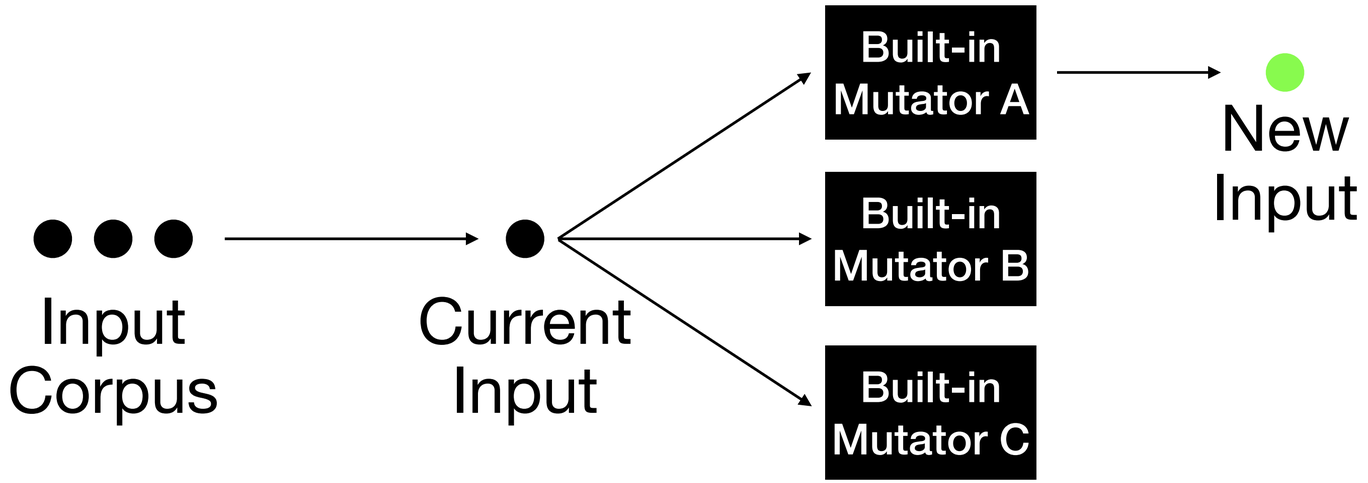
Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.



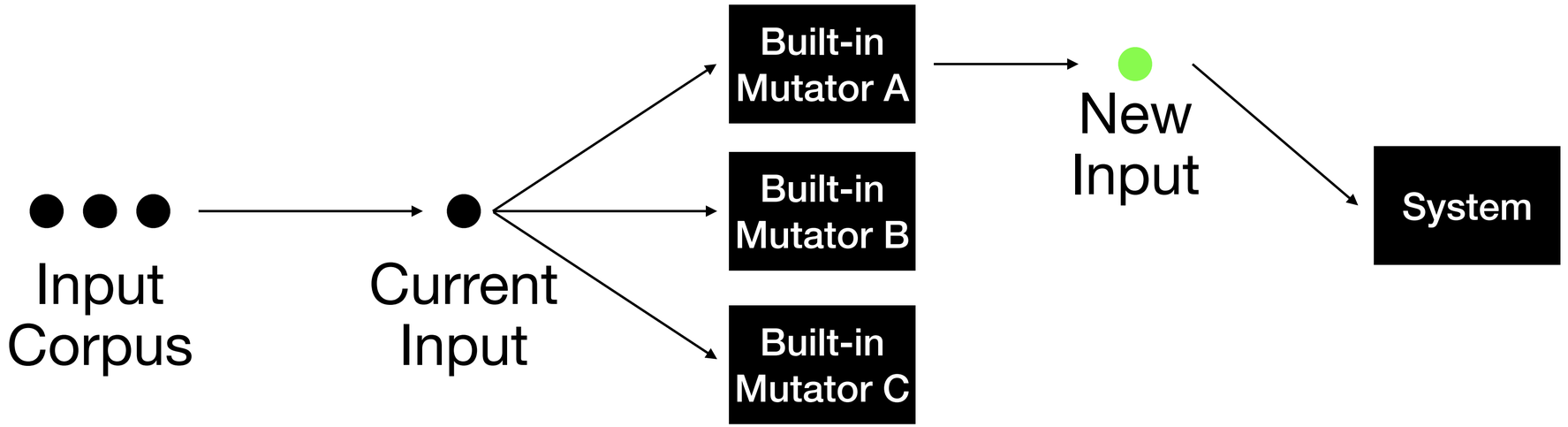
Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.



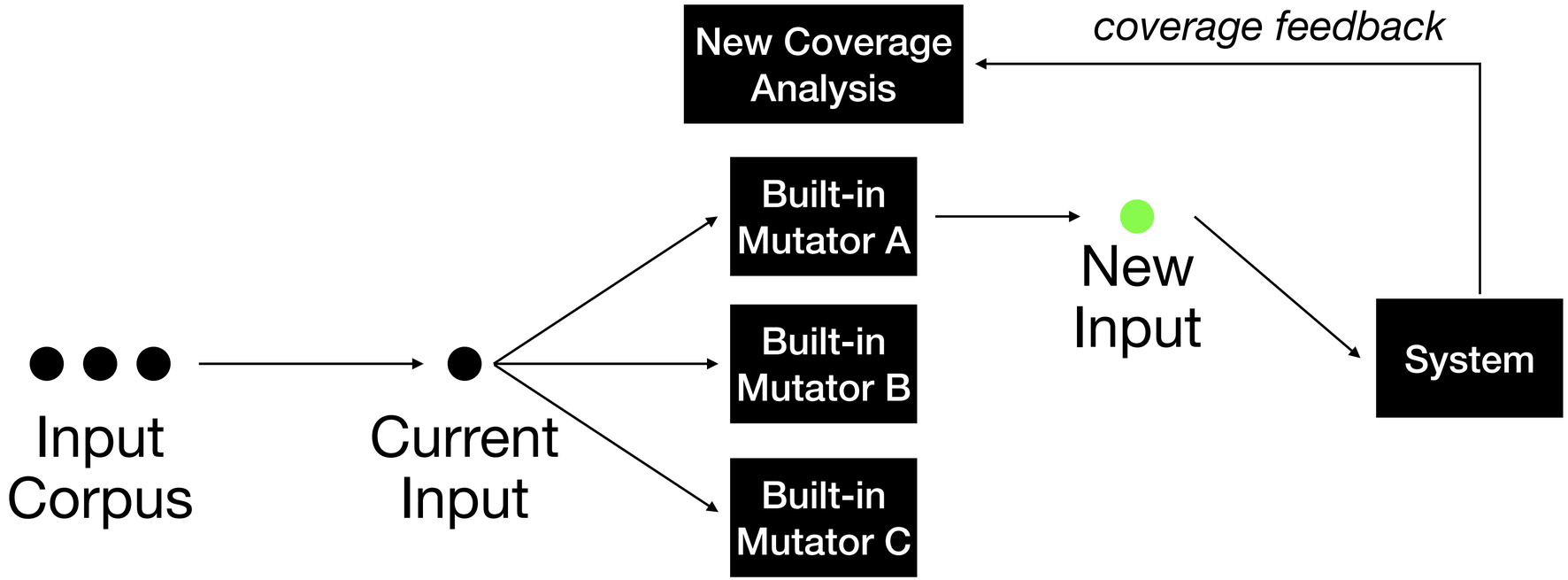
Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.



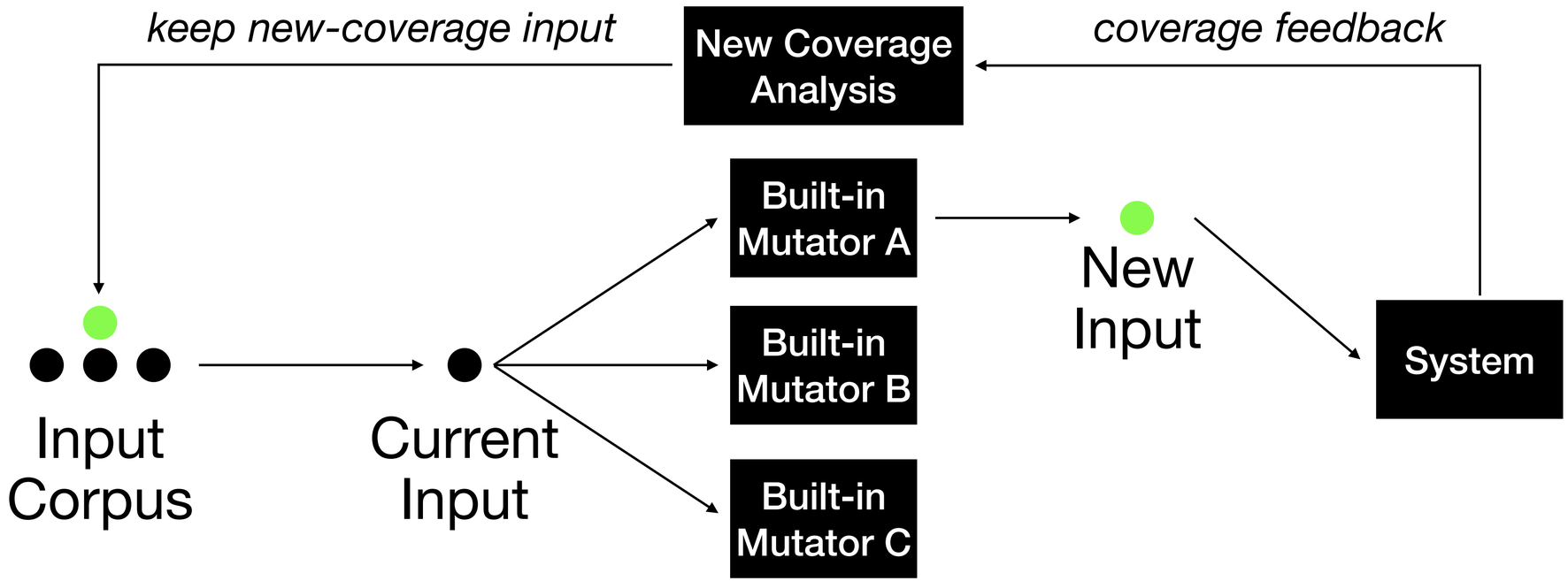
Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.



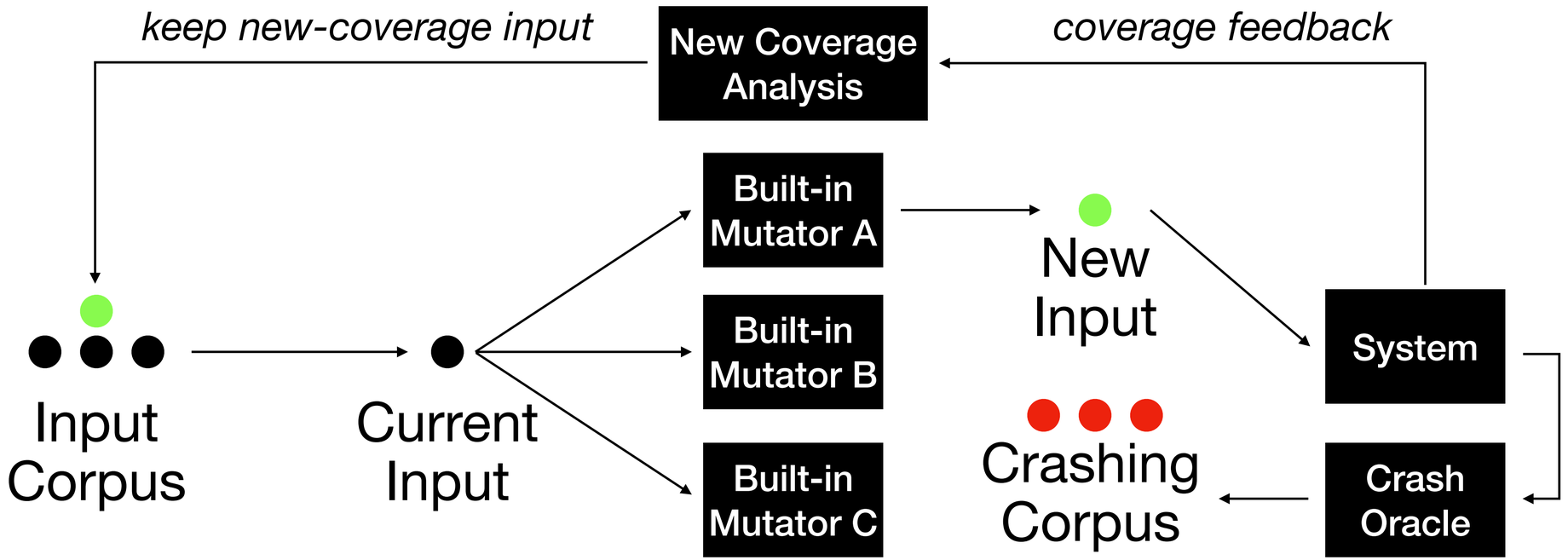
Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.



Background: Coverage-Guided Fuzzing

- Coverage-guided fuzzing is a simple but powerful program testing technique.



Related Work: JFS (Just Fuzz-It Solver)

Just Fuzz It: Solving Floating-Point Constraints using Coverage-Guided Fuzzing

Daniel Liew
dan@su-root.co.uk
Imperial College London
United Kingdom

Cristian Cadar
c.cadar@imperial.ac.uk
Imperial College London
United Kingdom

Alastair F. Donaldson
afd@imperial.ac.uk
Imperial College London
United Kingdom

J. Ryan Stinnett
jryans@gmail.com
Mozilla
United States

Related Work: JFS (Just Fuzz-It Solver)

Just Fuzz It: Solving Floating-Point Constraints using Coverage-Guided Fuzzing

Daniel Liew
dan@su-root.co.uk
Imperial College London
United Kingdom

Cristian Cadar
c.cadar@imperial.ac.uk
Imperial College London
United Kingdom

Alastair F. Donaldson
afd@imperial.ac.uk
Imperial College London
United Kingdom

J. Ryan Stinnett
jryans@gmail.com
Mozilla
United States

- **JFS** is an incomplete SMT solver.

Related Work: JFS (Just Fuzz-It Solver)

Just Fuzz It: Solving Floating-Point Constraints using Coverage-Guided Fuzzing

Daniel Liew
dan@su-root.co.uk
Imperial College London
United Kingdom

Cristian Cadar
c.cadar@imperial.ac.uk
Imperial College London
United Kingdom

Alastair F. Donaldson
afd@imperial.ac.uk
Imperial College London
United Kingdom

J. Ryan Stinnett
jryans@gmail.com
Mozilla
United States

- **JFS** is an incomplete SMT solver.
  - It may only prove that a formula is satisfiable, but never unsatisfiable.

Related Work: JFS (Just Fuzz-It Solver)

Just Fuzz It: Solving Floating-Point Constraints using Coverage-Guided Fuzzing

Daniel Liew
dan@su-root.co.uk
Imperial College London
United Kingdom

Cristian Cadar
c.cadar@imperial.ac.uk
Imperial College London
United Kingdom

Alastair F. Donaldson
afd@imperial.ac.uk
Imperial College London
United Kingdom

J. Ryan Stinnett
jryans@gmail.com
Mozilla
United States

- **JFS** is an incomplete SMT solver.
  - It may only prove that a formula is satisfiable, but never unsatisfiable.
- Coverage-guided fuzzing is used to find **just one satisfying assignment**.

Related Work: JFS (Just Fuzz-It Solver)

Related Work: JFS (Just Fuzz-It Solver)

$$x + y = 10$$

$x, y \in \text{FloatingPoint<32>}$

Input SMT Formula

Related Work: JFS (Just Fuzz-It Solver)

$$x + y = 10$$

$x, y \in \text{FloatingPoint<32>}$

Input SMT Formula

SMT to C++ Transpiler

Related Work: JFS (Just Fuzz-It Solver)

x + y = 10

x, y ∈ FloatingPoint<32>

Input SMT Formula

SMT to C++ Transpiler


```
1   int smt_formula(float x, float y) {
2     float sum = x + y;
3     bool sat = sum == 10.0f;
4     if (!sat) {
5       // UNSAT assignment
6       return 0;
7     }
8     // SAT assignment
9     abort();
10  }
```

Related Work: JFS (Just Fuzz-It Solver)

x + y = 10

x, y ∈ FloatingPoint<32>

Input SMT Formula

SMT to C++ Transpiler


```
1   int smt_formula(float x, float y) {
2     float sum = x + y;
3     bool sat = sum == 10.0f;
4     if (!sat) {
5       // UNSAT assignment
6       return 0;
7     }
8     // SAT assignment
9     abort();
10  }
```

Related Work: JFS (Just Fuzz-It Solver)

x + y = 10

x, y ∈ FloatingPoint<32>

Input SMT Formula

SMT to C++ Transpiler


```
1    int smt_formula(float x, float y) {
2      float sum = x + y;
3      bool sat = sum == 10.0f;
4      if (!sat) {
5       // UNSAT assignment
6       return 0;
7      }
8      // SAT assignment
9      abort();
10   }
```

Related Work: JFS (Just Fuzz-It Solver)

x + y = 10

x, y ∈ FloatingPoint<32>

Input SMT Formula

SMT to C++ Transpiler


```
1    int smt_formula(float x, float y) {
2      float sum = x + y;
3      bool sat = sum == 10.0f;
4      if (!sat) {
5        // UNSAT assignment
6        return 0;
7      }
8      // SAT assignment
9      abort();
10   }
```

Related Work: JFS (Just Fuzz-It Solver)

x + y = 10

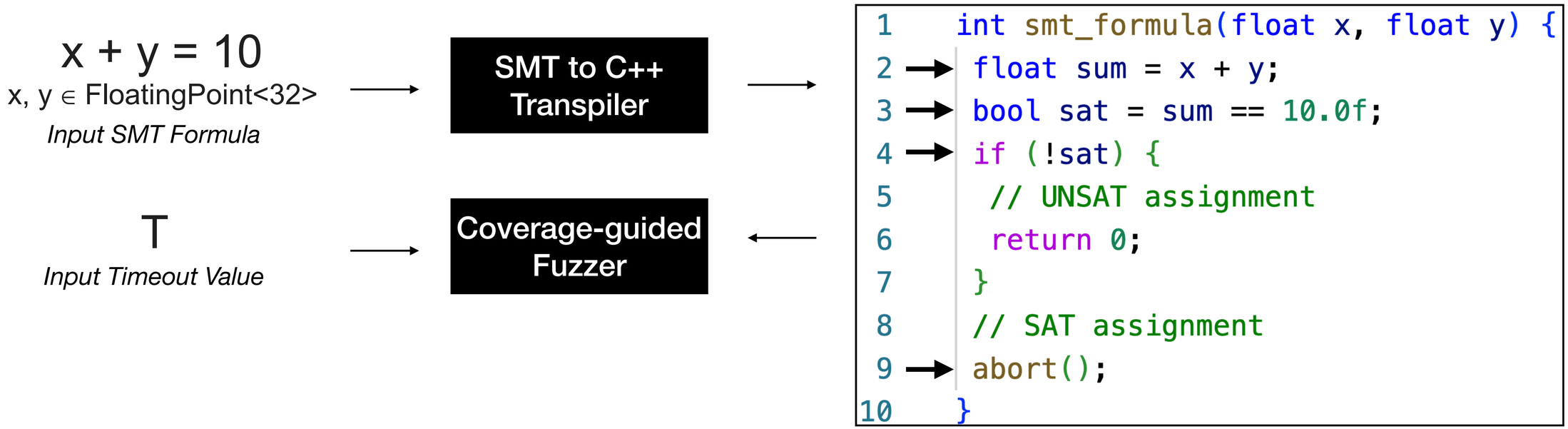
x, y ∈ FloatingPoint<32>

Input SMT Formula

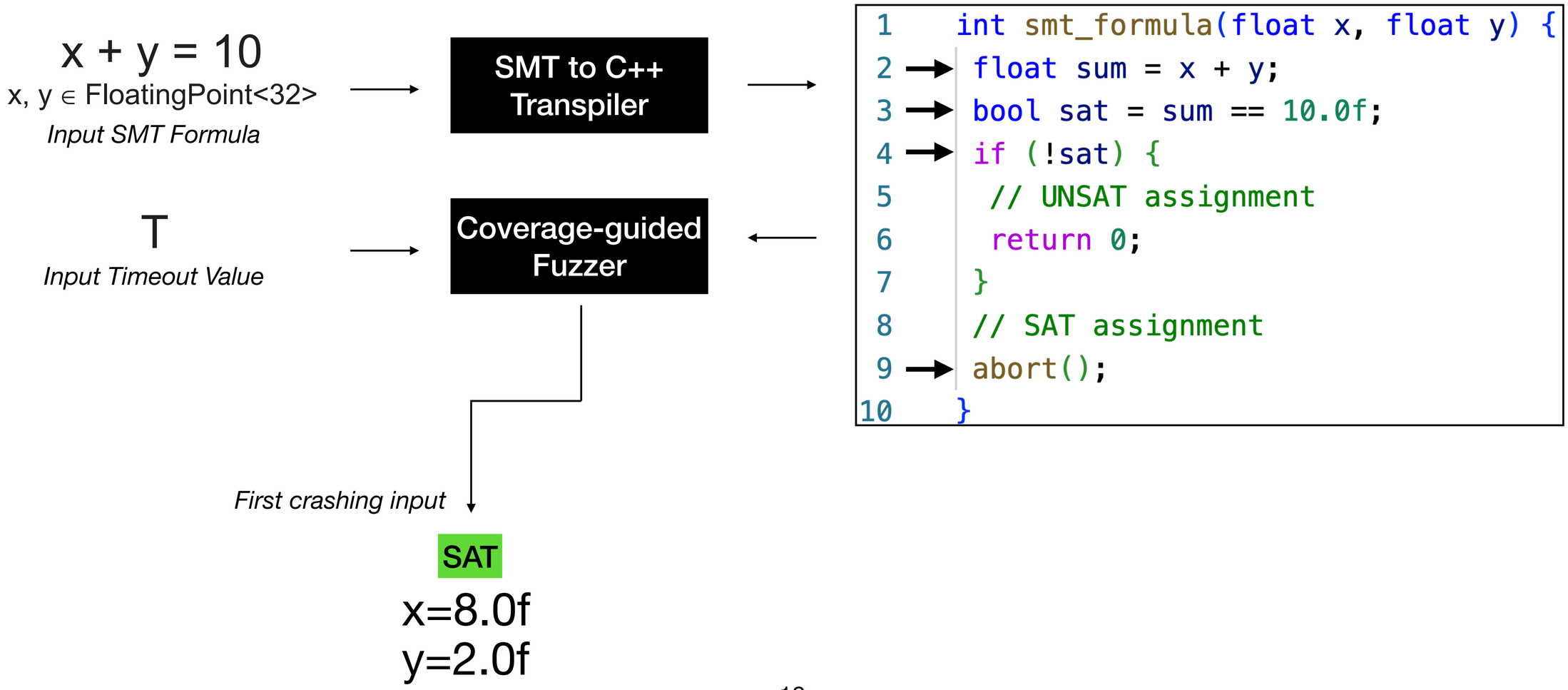
SMT to C++ Transpiler


```
1    int smt_formula(float x, float y) {
2      float sum = x + y;
3      bool sat = sum == 10.0f;
4      if (!sat) {
5        // UNSAT assignment
6        return 0;
7      }
8      // SAT assignment
9      abort();
10   }
```

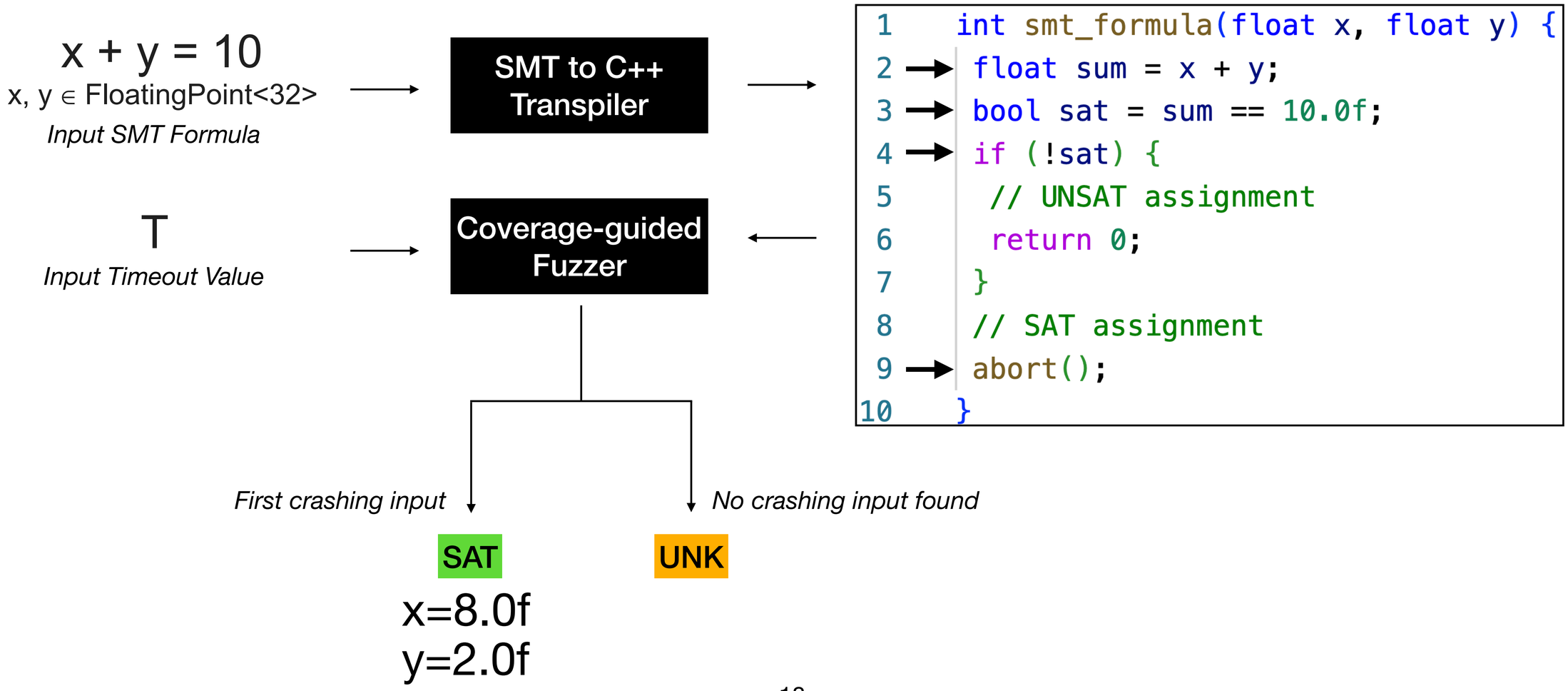
Related Work: JFS (Just Fuzz-It Solver)



Related Work: JFS (Just Fuzz-It Solver)



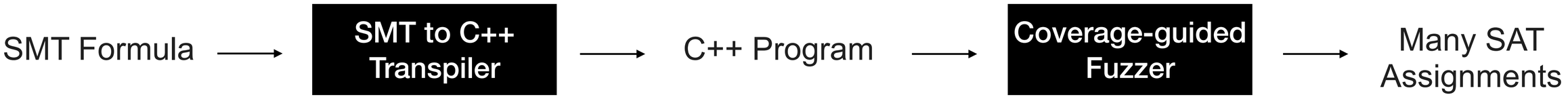
Related Work: JFS (Just Fuzz-It Solver)



Our Work: JFSampler (Just Fuzz-it Sampler)

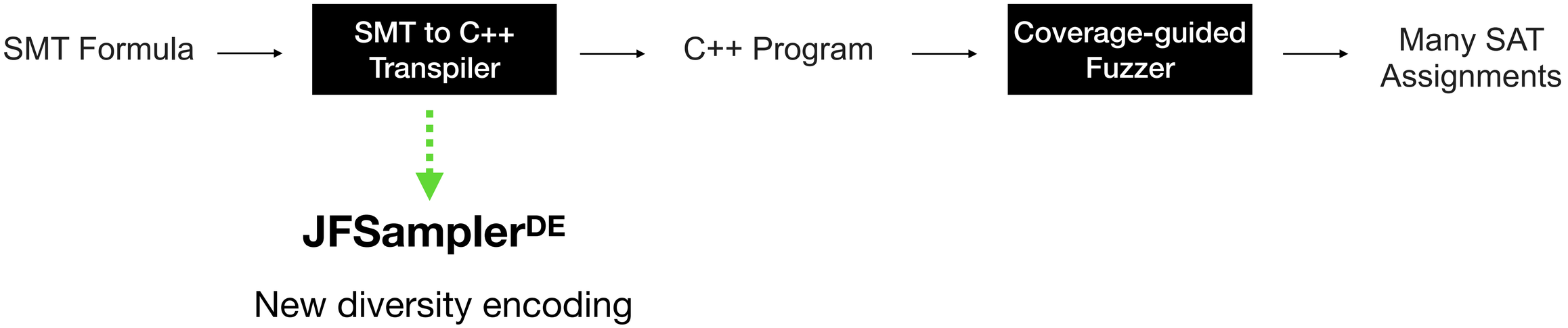
Our Work: JFSampler (Just Fuzz-it Sampler)

- **JFSampler$^{\text{Naive}}$** is **JFS** but it continues fuzzing after the first crashing input.



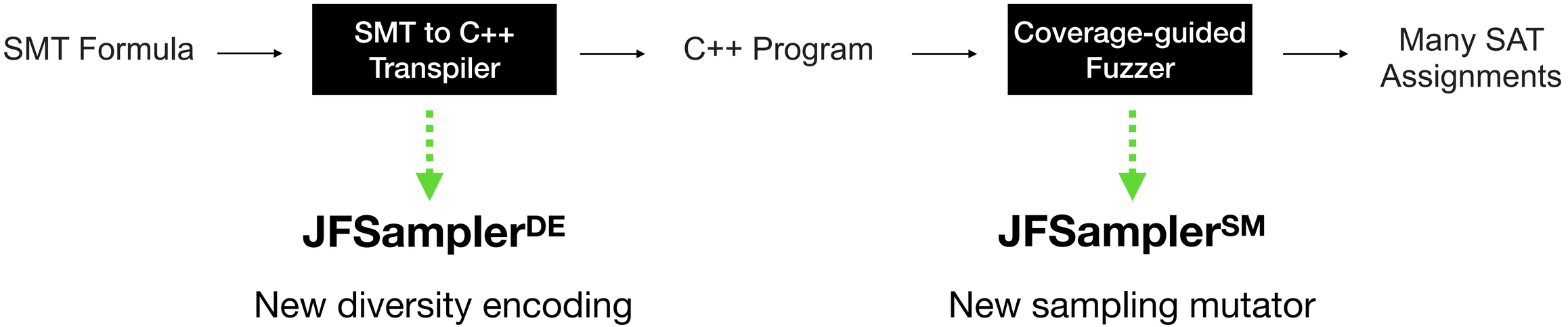
Our Work: JFSampler (Just Fuzz-it Sampler)

- **JFSampler^{Naive}** is **JFS** but it continues fuzzing after the first crashing input.



Our Work: JFSampler (Just Fuzz-it Sampler)

- **JFSampler^{Naive}** is **JFS** but it continues fuzzing after the first crashing input.



Coverage Saturation in JFSampler$^{\text{Naive}}$

Coverage Saturation in JFSampler$^{\text{Naive}}$

- The code coverage in the C++ function for satisfying assignments (samples) is very narrow.

Coverage Saturation in JFSampler$^{\text{Naive}}$

- The code coverage in the C++ function for satisfying assignments (samples) is very narrow.
- In our example, all satisfying inputs will take the same path.

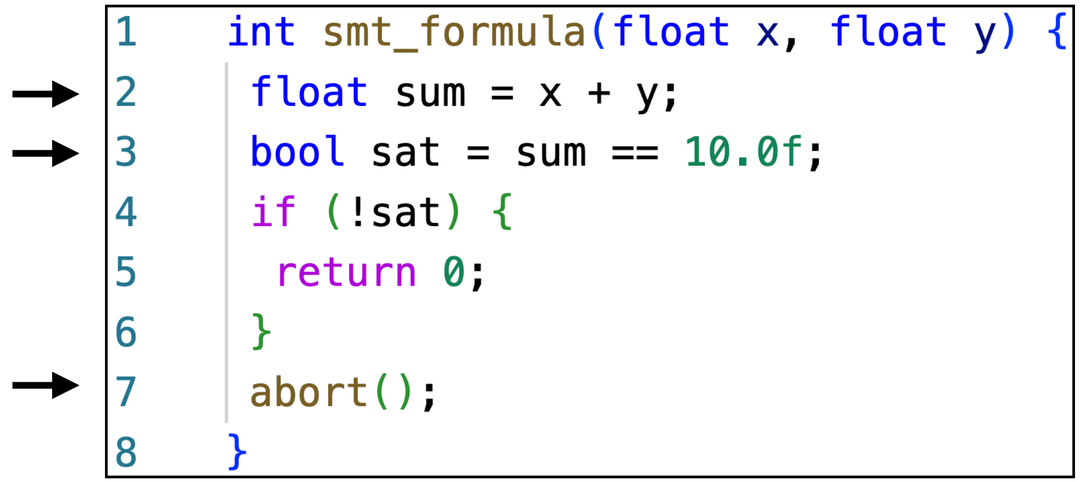
Coverage Saturation in JFSampler$^{\text{Naive}}$

- The code coverage in the C++ function for satisfying assignments (samples) is very narrow.
- In our example, all satisfying inputs will take the same path.

```
1    int smt_formula(float x, float y) {
2      float sum = x + y;
3      bool sat = sum == 10.0f;
4      if (!sat) {
5        return 0;
6      }
7      abort();
8    }
```

Coverage Saturation in JFSampler$^{\text{Naive}}$

- The code coverage in the C++ function for satisfying assignments (samples) is very narrow.
- In our example, all satisfying inputs will take the same path.



- If coverage saturates for satisfying assignments, the fuzzer cannot distinguish them.

SMT Coverage Metric

SMT Coverage Metric

- At a high-level, it captures how much of the formula’s logic has been exercised by all the samples.

SMT Coverage Metric

- At a high-level, it captures how much of the formula's logic has been exercised by all the samples.
- It keeps track of the subexpressions' values across all samples.

SMT Coverage Metric

- At a high-level, it captures how much of the formula’s logic has been exercised by all the samples.
- It keeps track of the subexpressions’ values across all samples.
- **SMTSampler** defined it to measure sample diversity, used for evaluation but not part of the algorithm.

JFSampler^{DE} (Diversity Encoding)

JFSampler^{DE} (Diversity Encoding)

- **JFSampler^{DE}** makes the coverage-guided fuzzer aware of the SMT coverage metric.

JFSampler^{DE} (Diversity Encoding)

- **JFSampler^{DE}** makes the coverage-guided fuzzer aware of the SMT coverage metric.
- Diverse satisfying assignments will now take different paths in the code.

JFSampler^{DE} (Diversity Encoding)

- **JFSampler^{DE}** makes the coverage-guided fuzzer aware of the SMT coverage metric.
- Diverse satisfying assignments will now take different paths in the code.

```
 1  int smt_formula(float x, float y) {
 2    float sum = x + y;
 3    bool sat = sum == 10.0f;
 4    if (!sat) {
 5     return 0;
 6    }
 7    // New code coverage for SAT assignments
 8    SMT_COVERAGE(sum);
 9    abort();
10  }
```

JFSampler^{DE} (Diversity Encoding)

- **JFSampler^{DE}** makes the coverage-guided fuzzer aware of the SMT coverage metric.
- Diverse satisfying assignments will now take different paths in the code.

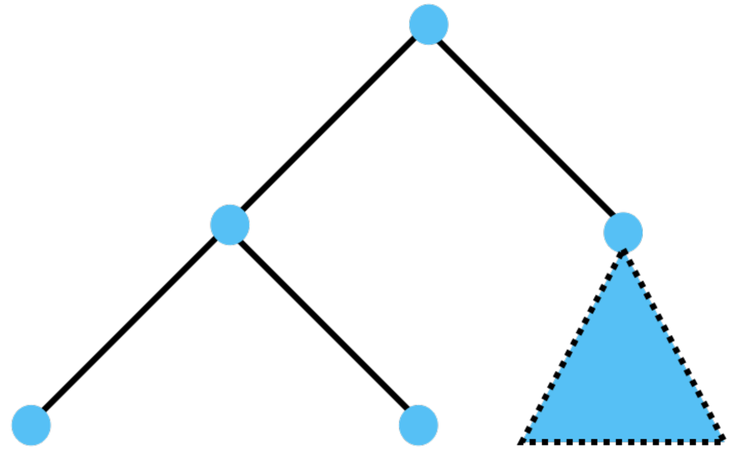
```
1   int smt_formula(float x, float y) {
2     float sum = x + y;
3     bool sat = sum == 10.0f;
4     if (!sat) {
5       return 0;
6     }
7     // New code coverage for SAT assignments
8     SMT_COVERAGE(sum);
9     abort();
10  }
```

JFSampler^{DE} (Diversity Encoding)

- **JFSampler^{DE}** makes the coverage-guided fuzzer aware of the SMT coverage metric.
- Diverse satisfying assignments will now take different paths in the code.

```
1   int smt_formula(float x, float y) {
2     float sum = x + y;
3     bool sat = sum == 10.0f;
4     if (!sat) {
5       return 0;
6     }
7     // New code coverage for SAT assignments
8     SMT_COVERAGE(sum);
9     abort();
10  }
```

```
11  #define SMT_COVERAGE(F) {
12    // New C++ code that, if covered,
13    // means an increase in diversity
14
15
16
17
18
19
20
21
22  }
```



Bespoke Mutator for SMT Sampling

Bespoke Mutator for SMT Sampling

- **JFSampler^{Naive}** and **JFSampler^{DE}** both rely on the set of byte-level mutators that the underlying coverage-guided fuzzer has.

Bespoke Mutator for SMT Sampling

- **JFSampler^{Naive}** and **JFSampler^{DE}** both rely on the set of byte-level mutators that the underlying coverage-guided fuzzer has.
- **SMTSampler** defined a bit-level combining heuristic that can likely merge three satisfying assignments into a new one.

Bespoke Mutator for SMT Sampling

- **JFSampler$^{\text{Naive}}$** and **JFSampler$^{\text{DE}}$** both rely on the set of byte-level mutators that the underlying coverage-guided fuzzer has.
- **SMTSampler** defined a bit-level combining heuristic that can likely merge three satisfying assignments into a new one.

$$COMBINE(A_1, A_2, A_3) = A_1 \oplus ((A_1 \oplus A_2) \vee (A_1 \oplus A_3))$$

Bespoke Mutator for SMT Sampling

- **JFSampler$^{\text{Naive}}$** and **JFSampler$^{\text{DE}}$** both rely on the set of byte-level mutators that the underlying coverage-guided fuzzer has.
- **SMTSampler** defined a bit-level combining heuristic that can likely merge three satisfying assignments into a new one.

$$COMBINE(A_1, A_2, A_3) = A_1 \oplus ((A_1 \oplus A_2) \vee (A_1 \oplus A_3))$$

- **SMTSampler** blindly applies the heuristic without any feedback; it could combine uninteresting satisfying assignments.

JFSampler^{SM} (Sampling Mutator)

JFSampler^{SM} (Sampling Mutator)

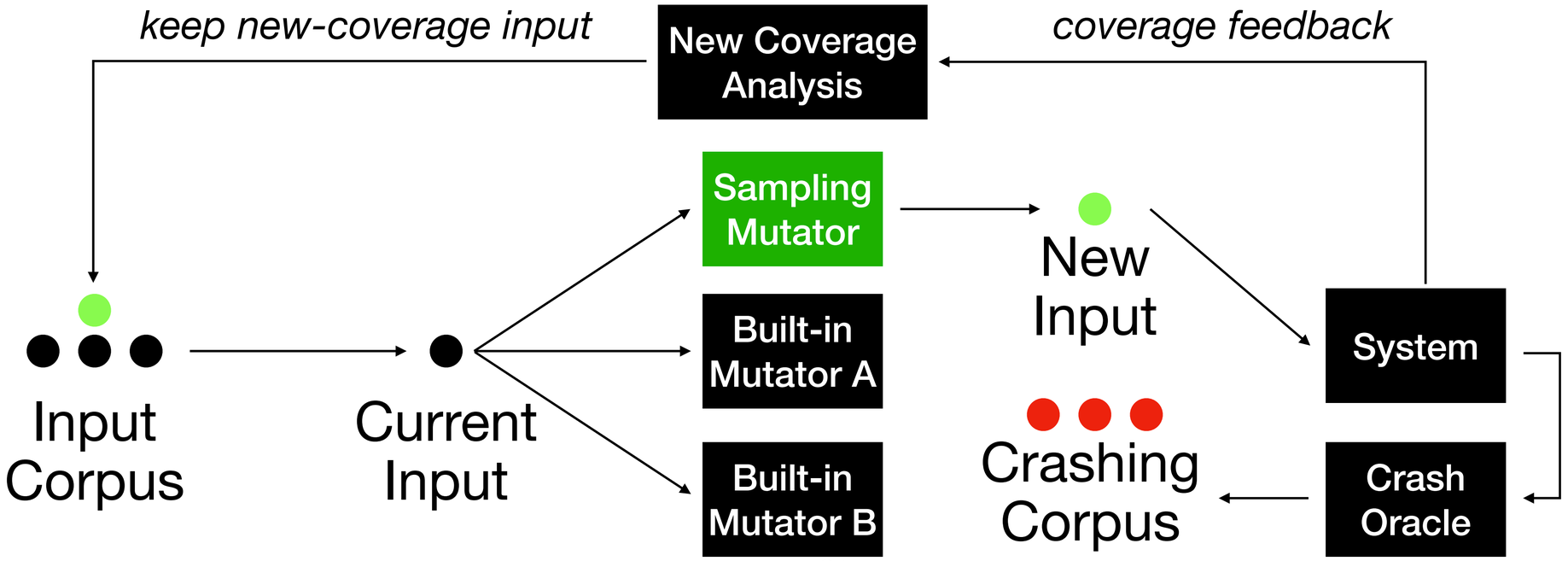
- **JFSampler^{SM}** incorporates the **SMTSampler’**s heuristic as a new *mutator* in the fuzzer.

JFSampler^{SM} (Sampling Mutator)

- **JFSampler^{SM}** incorporates the **SMTSampler’**s heuristic as a new *mutator* in the fuzzer.
- The sampling mutator benefits from the code-coverage feedback and is applied to test cases deemed interesting by the fuzzer.

JFSampler^{SM} (Sampling Mutator)

- **JFSampler^{SM}** incorporates the **SMTSampler'**s heuristic as a new *mutator* in the fuzzer.
- The sampling mutator benefits from the code-coverage feedback and is applied to test cases deemed interesting by the fuzzer.



JFSampler^{SM+DE}

JFSampler^{SM+DE}

- **JFSampler^{SM+DE}** combines all of our features:

JFSampler^{SM+DE}

- **JFSampler^{SM+DE}** combines all of our features:
  - the sampling mutator for the coverage-guided fuzzer

JFSampler^{SM+DE}

- **JFSampler^{SM+DE}** combines all of our features:
  - the sampling mutator for the coverage-guided fuzzer
  - the new C++ encoding for the SMT coverage metric

JFSampler^{SM+DE}

- **JFSampler^{SM+DE}** combines all of our features:
  - the sampling mutator for the coverage-guided fuzzer
  - the new C++ encoding for the SMT coverage metric
- We expect this mode to perform the best in terms of diversity and throughput.

Evaluation

Evaluation

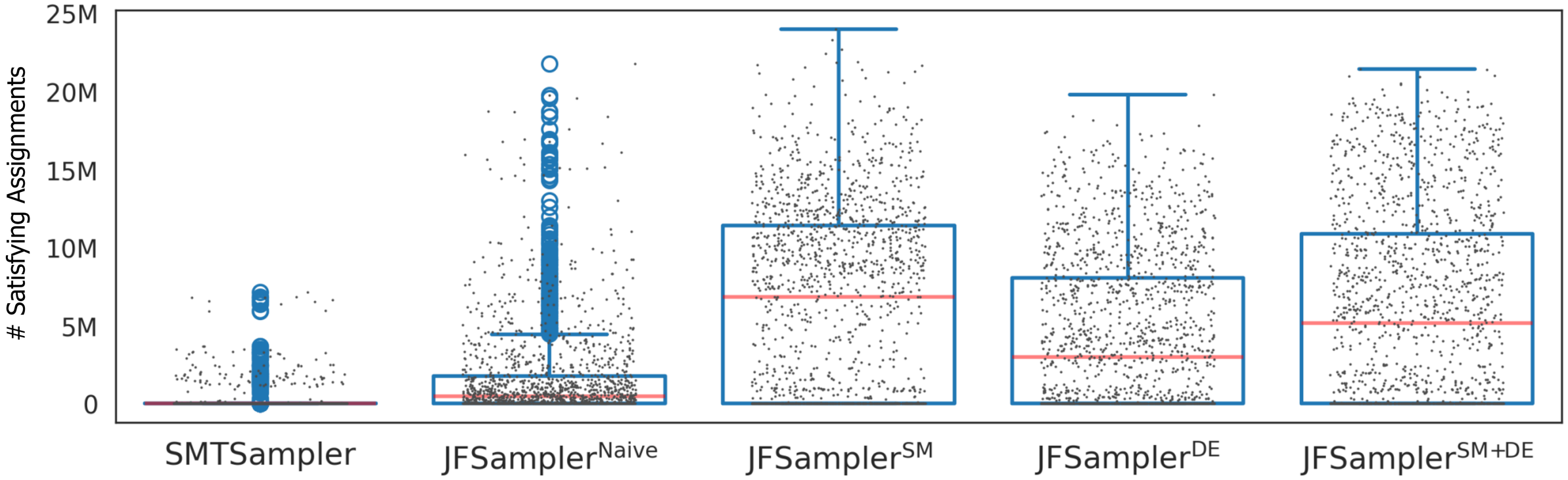
- Our evaluation was conducted in the SMT-LIB benchmark.

Evaluation

- Our evaluation was conducted in the SMT-LIB benchmark.
- We evaluated the FP and FP+BV suites (total of 862 formulas).

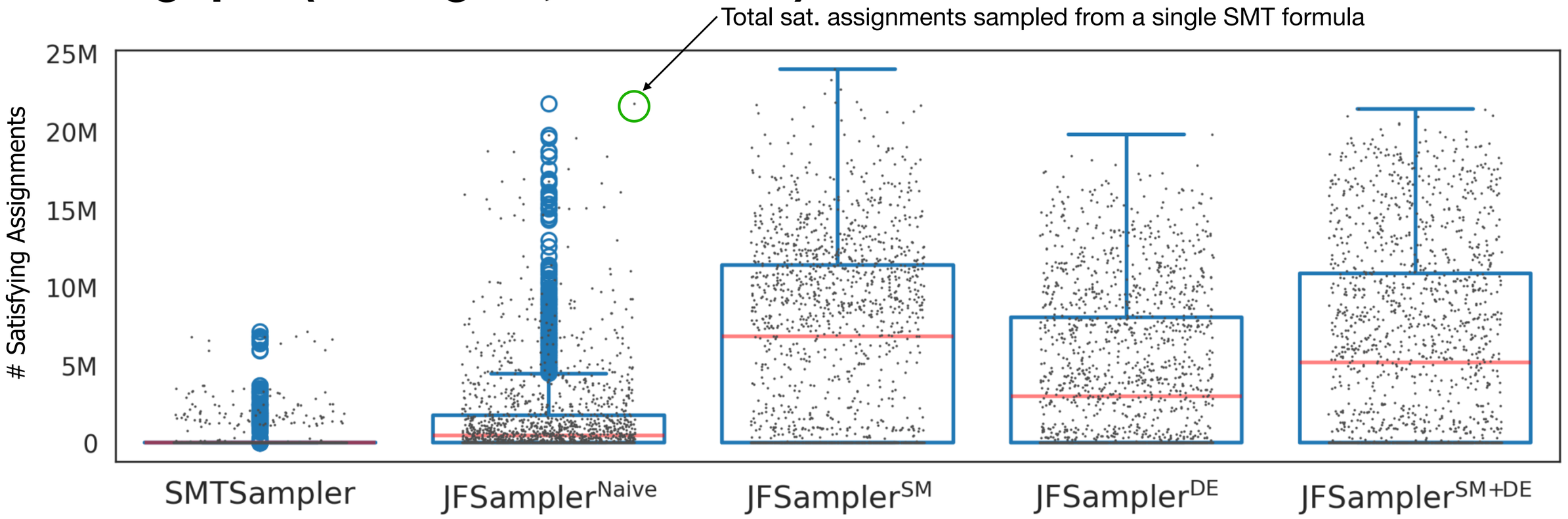
FP Suite

Throughput (the higher, the better)



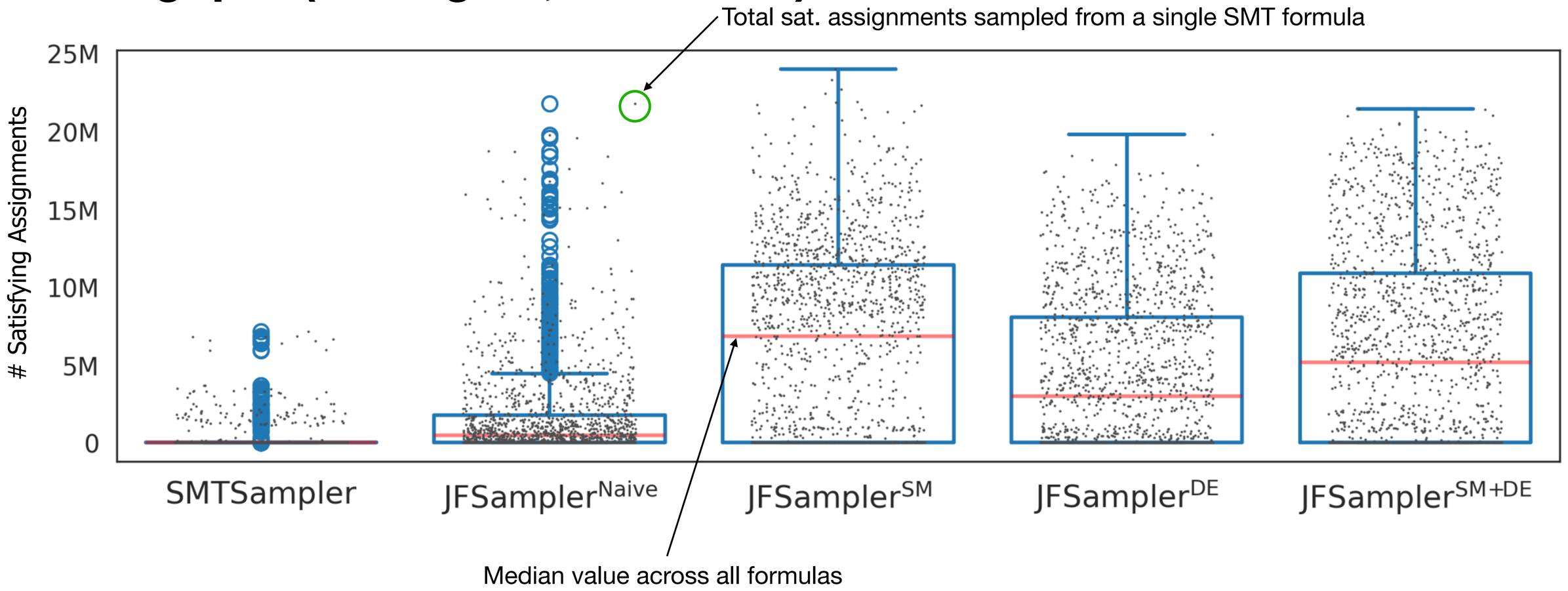
FP Suite

Throughput (the higher, the better)



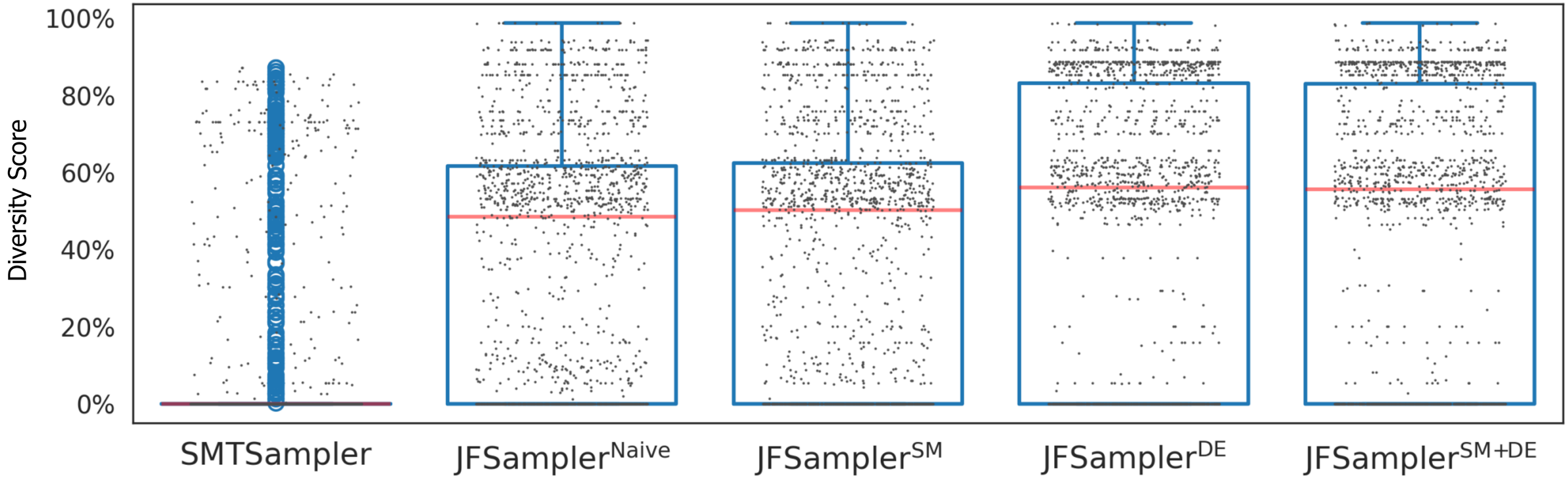
FP Suite

Throughput (the higher, the better)



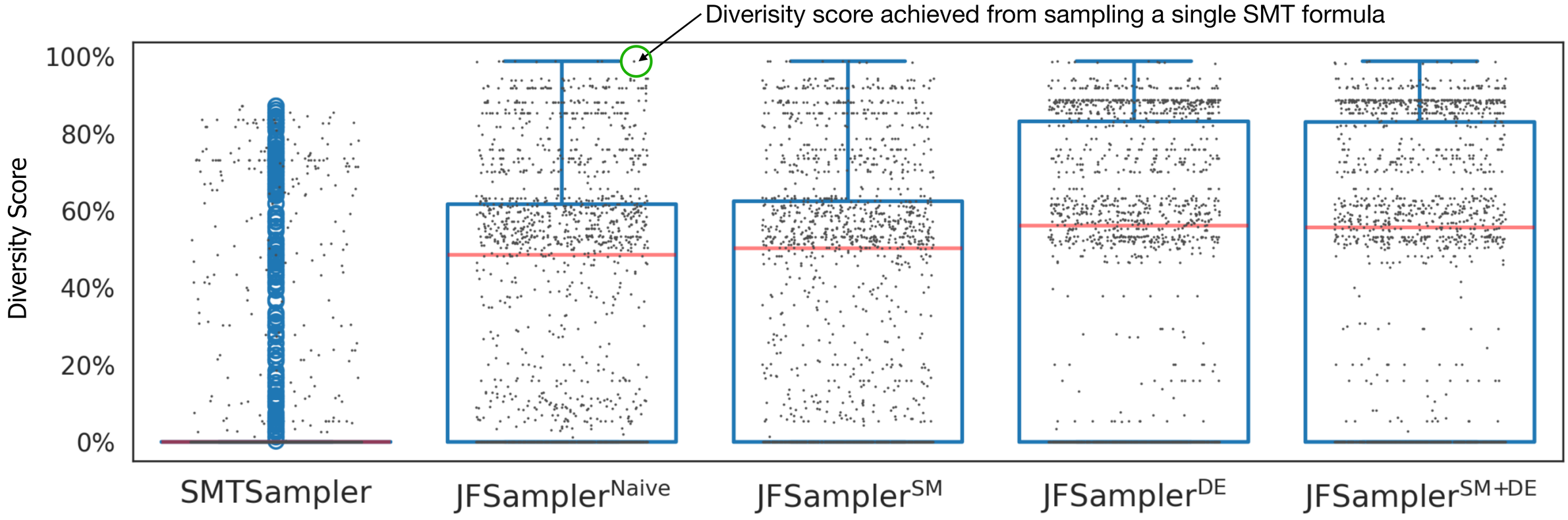
FP Suite

Diversity (the higher, the better)



FP Suite

Diversity (the higher, the better)



Conclusions

Conclusions

- We designed and implemented, **JFSampler**, the first SMT sampling technique using coverage-guided fuzzing.

Conclusions

- We designed and implemented, **JFSampler**, the first SMT sampling technique using coverage-guided fuzzing.
- **JFSampler^{SM+DE}** outperforms the **SMTSampler** in the FP domain.

Conclusions

- We designed and implemented, **JFSampler**, the first SMT sampling technique using coverage-guided fuzzing.
- **JFSampler^{SM+DE}** outperforms the **SMTSampler** in the FP domain.
- We hope our results can help in the adoption of SMT sampling for testing techniques.

Conclusions

- We designed and implemented, **JFSampler**, the first SMT sampling technique using coverage-guided fuzzing.
- **JFSampler^{SM+DE}** outperforms the **SMTSampler** in the FP domain.
- We hope our results can help in the adoption of SMT sampling for testing techniques.
- Our tool: https://srg.doc.ic.ac.uk/projects/jfs/