UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

AutoStub: Genetic Programming-Based Stub Creation for Symbolic Execution

Felix Mächtle, **Nils Loose**, Jan-Niclas Serr, Jonas Sander, Thomas Eisenbarth

July 17, 2025

Humies, GECCO 2025

Motivation: Symbolic Execution

```
function main(user_age):
 if verify_input(user_age) == True:
  display("Access Granted")


function verify_input(user_age):
 if user_age >= 16:
  return True
 return False
```

Motivation: Symbolic Execution

```
function main(15):
 if verify_input(x) == True:
  display("Access Granted")


function verify_input(x):
 if x >= 16:
  return True
 return False
```

SMT Constraints

Motivation: Symbolic Execution

```
function main(15):
 if verify_input(x) == True:
  display("Access Granted")


function verify_input(x):
 if x >= 16:
  return True
 return False
```

SMT Constraints

$(declare\text{-}fun\ s_1\ ()\ int)$

Motivation: Symbolic Execution

```
function main(15):
 if verify_input(15) == True:
  display("Access Granted")


function verify_input(x):
 if x >= 16:
  return True
 return False
```

SMT Constraints

```
(declare-fun s1 () int)
(assert (= ? True)
```

Motivation: Symbolic Execution

```
function main(x):
 if verify_input(15) == True:
  display("Access Granted")


function verify_input(15):
 if 15 >= 16:
  return True
 return False
```

SMT Constraints

```
(declare-fun s1 ()  int)
(assert (= ? True)

(assert (>= s1 16))
```

Motivation: Symbolic Execution

```
function main(15):
 if verify_input(15) == True:
  display("Access Granted")


function verify_input(15):
 if 15 >= 16:
  return True
 return False
```

SMT Constraints

```
(declare-fun s1 () int)
(assert (= False True)

(assert (>= s1 16))
```

Motivation: Symbolic Execution

```
function main(15):
 if verify_input(15) == True:
  display("Access Granted")
```

SMT Constraints

```
(declare-fun s1 () int)
(assert (= False True)
```

Out-of-instrumentation

Out-of-scope

Motivation: Symbolic Execution

```
function main(15):
 if verify_input(15) == True:
  display("Access Granted")
```

SMT Constraints

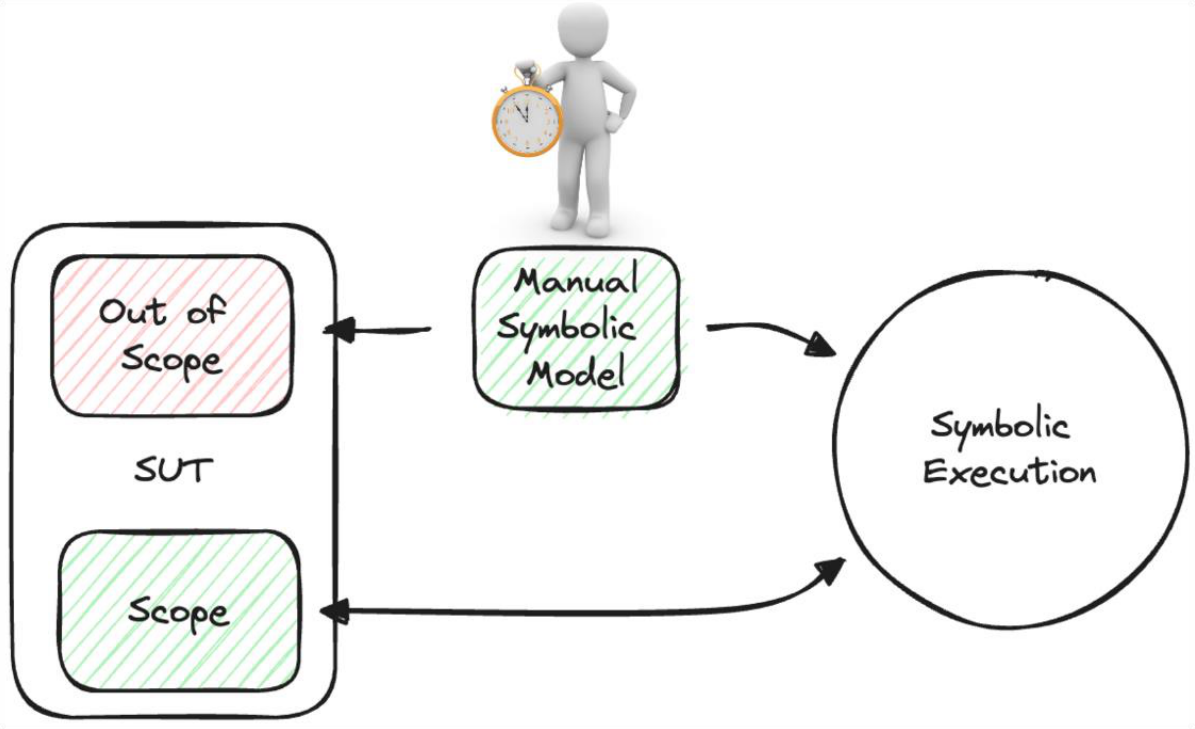
```
(declare-fun s1 () int)
(assert (= False True)
```

Out-of-instrumentation

- Native functions
- Third-Party Libraries
- Un-instrumented functions

Out-of-scope

Motivation: Best Solution



Motivation: Symbolic Execution

```
function main(15):
 if verify_input(15) == True:
  display("Access Granted")
```

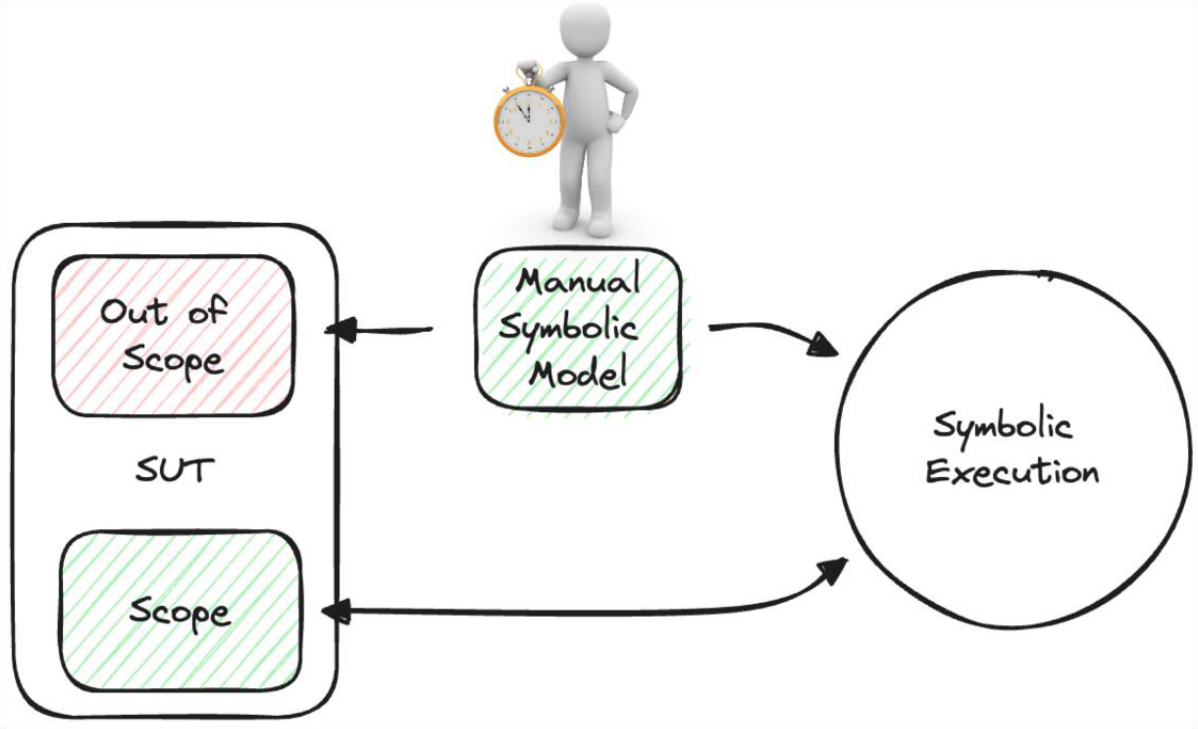
Out-of-instrumentation
▶ Manual-model

SMT Constraints

```
(declare-fun s1 () int)
(assert (= (f1 s1) True))
```

```
(define-fun f1((x Int))
 Bool
  (>= x 16))
```

Motivation: Best Solution



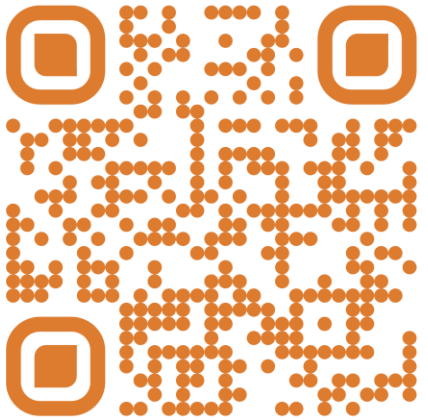
Drawbacks:

- Time-intensive, e.g., even simple functions must be considered
- Risk of omissions: every new target requires new symbolic stubs

SWAT: DSE for JVM

- SWAT is an open source instrumentation-based dynamic symbolic execution engine.
  - Currently all symbolic models are engineered by hand.
- Participates each year in the SV-Comp software verification competition (670 tasks in the Java track).
  - 36% of all tasks have some symbolic context loss for SWAT caused by native functions
  - Reduces final score by > 10% for safe cases
  - Reduces final score by > 15% for violation cases

SWAT Repository



Workflow AutoStub

Details in our paper!

Example Stub: Double.isNaN(double)

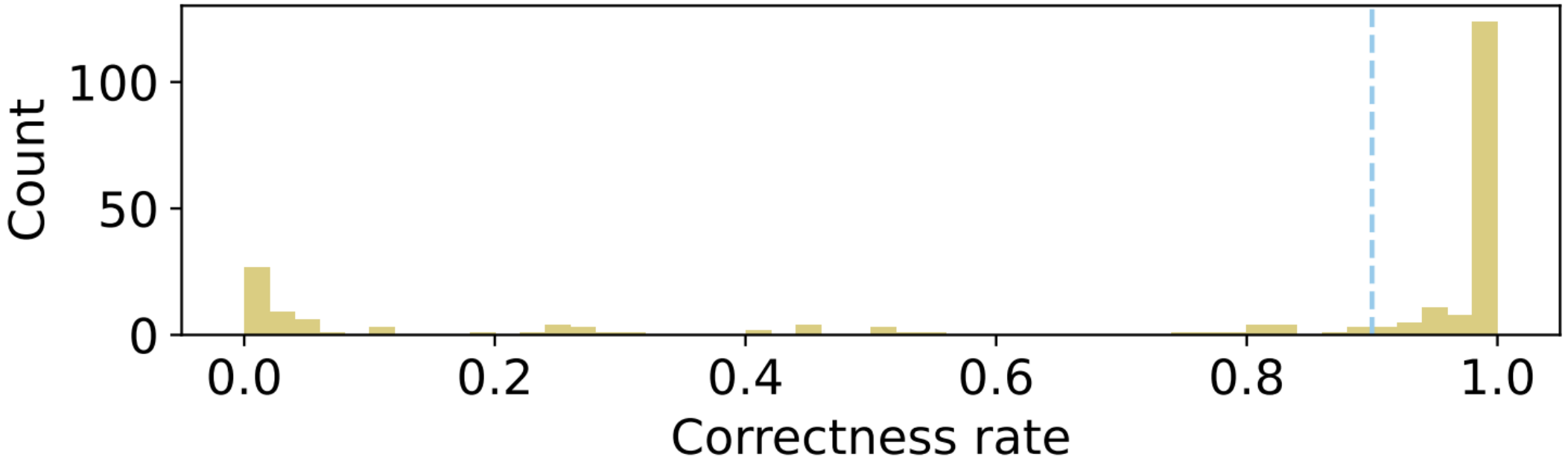
$$!(-1 < |x|)$$

Criterion B: Symbolic-Execution Breakthrough

- On-the-fly stub generation
- Fully automated
- Opens unreachable paths

Evaluation: Correctness of Generated Expressions

- Benchmark: Input → Predict output
- Dataset: stateless functions from `java.util.*`, `java.lang.Math`, `java.lang.StrictMath`
- Currently: only primitive types and strings



Evaluation: Symbolic Execution

```
public static boolean test(String[] args) {

 //  fetch input
 Boolean input_1_0 = Boolean.valueOf(args[0]);
 Boolean input_2_0 = Boolean.valueOf(args[1]);

 //  Perform computation
 Boolean output_1 = input_1_0.booleanValue();
 Boolean output_2 = input_2_0.booleanValue();

 //  Compare output
 return (output_1 == false && output_2 == true);
}
```

Evaluation: Symbolic Execution

```
public static boolean test(String[] args) {

 //  fetch input
 Boolean input_1_0 = Boolean.valueOf(args[0]);
 Boolean input_2_0 = Boolean.valueOf(args[1]);

 //  Perform computation
 Boolean output_1 = input_1_0.booleanValue();
 Boolean output_2 = input_2_0.booleanValue();

 //  Compare output
 return (output_1 == false && output_2 == true);
}
```

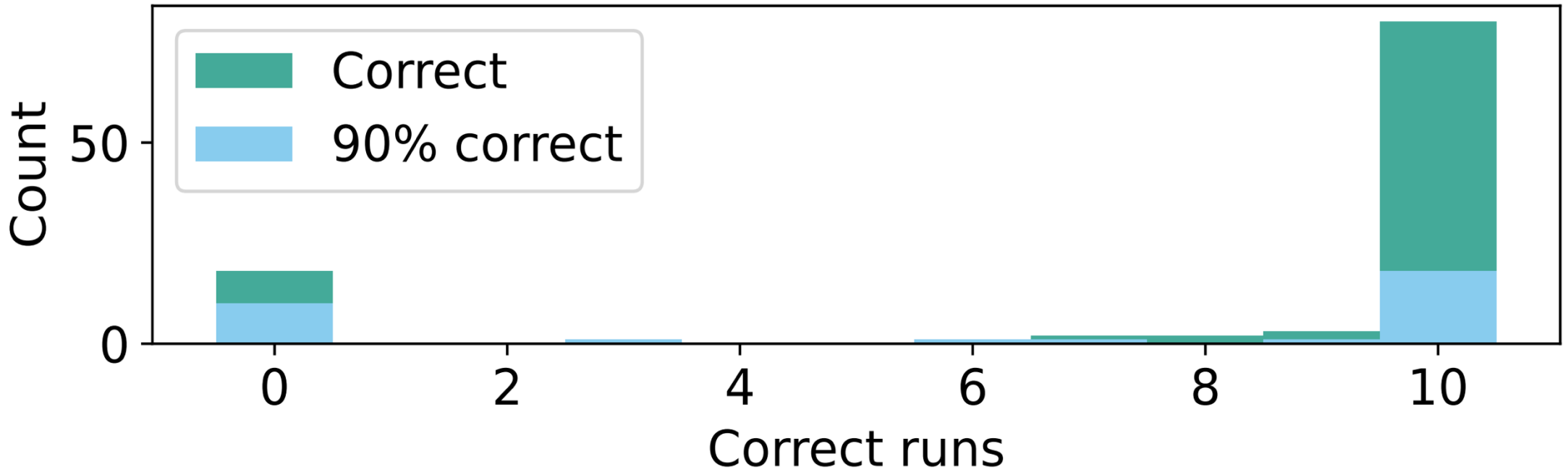
Evaluation: Symbolic Execution

```
public static boolean test(String[] args) {
  // fetch input
  Boolean input_1_0 = Boolean.valueOf(args[0]);
  Boolean input_2_0 = Boolean.valueOf(args[1]);

  // Perform computation
  Boolean output_1 = input_1_0.booleanValue();
  Boolean output_2 = input_2_0.booleanValue();

  // Compare output
  return (output_1 == false && output_2 == true);
}
```

Evaluation: Symbolic Execution



Criterion D: Immediately-Useful Artifact

- Evolution produces a library of ready to use Java stubs
- Drop-in usable today
- Open-Source

Criterion G: Long-Standing Difficulty Solved

- AutoStub enables automatic exploration of out-of-context functions
- Exemplarily shown for real Java standard-library calls
- No human effort required

Why AutoStub is #1

#1

- Puts evolutionary computation into everyday security workflows
- Replaces manual models with automated GP stubs
- Opens unreachable paths
- Able to find edge cases
- Helps in making software more secure
- Shown for real-world Java code
- Open-source

AutoStub: Genetic Programming-Based Stub Creation for Symbolic Execution

Contact

AutoStub Repo

Contact:

- Felix Mächtle
- f.maechtle@uni-luebeck.de
- https://mchtl.de

SWAT Repo

Contact:

Nils Loose

n.loose@uni-luebeck.de

https://bio.nils-loose.com