CPA-SymExec:
Efficient Symbolic Execution in CPAchecker

Dirk Beyer and **Thomas Lemberger**

LMU Munich, Germany

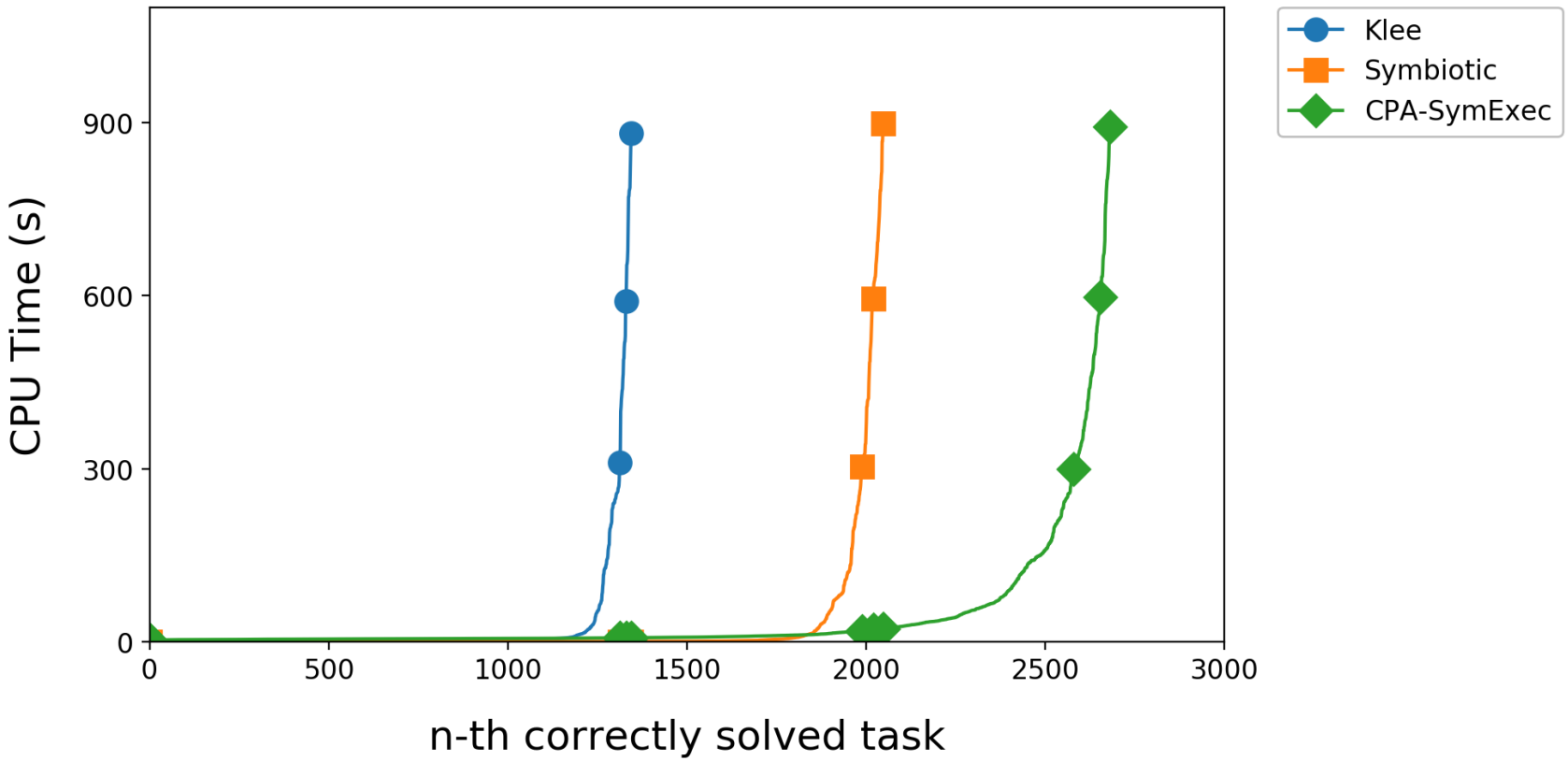
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CPA

SoSy-Lab Software Systems

Why you should listen

Interested in symbolic execution?
(formal verification, test-case generation, program repair, equivalence checking, . . .)
Analyze C programs, implemented in CPAchecker.



But...

Symbolic Execution can provide:

- Exhaustive (formal) verification
- Automatic test-case generation
- Symbolic and concrete program traces

But: It is not efficient (path explosion, SMT solving)

Existing tools:

- CREST/ CONTEST (dynamic)
- KLEE (heuristics)
- SYMBIOTIC (static slicing)
- ...

Our new tool:

- CPA-SYMEXEC: Abstraction with CEGAR (+ heuristics)

Optimizations in CPA-SymExec

CPA-SymExec uses common optimizations:

- SMT result caching (+ subset caching)
- Model re-use
- Minimal SAT checks

And includes new optimizations:

- Computation of definite assignments
- Simplification of symbolic identifiers
- Counterexample-guided Abstraction Refinement (CEGAR)

Symbolic Execution

- Replace concrete test-values with generic symbolic values
- **Symbolic Memory**: Stores (symbolic) value assignments.
- **Path Constraints**: Constrain symbolic values.

Symbolic Execution

- Replace concrete test-values with generic symbolic values
- **Symbolic Memory**: Stores (symbolic) value assignments.
- **Path Constraints**: Constrain symbolic values.

```
1  unsigned char a = ?;
2  unsigned char b = ?;
3  unsigned char c = b + 1;
4  while (a < 100)
5        a++;
6   if  (c == b)
7      error ();
```

Symbolic Execution

- Replace concrete test-values with generic symbolic values
- **Symbolic Memory**: Stores (symbolic) value assignments.
- **Path Constraints**: Constrain symbolic values.

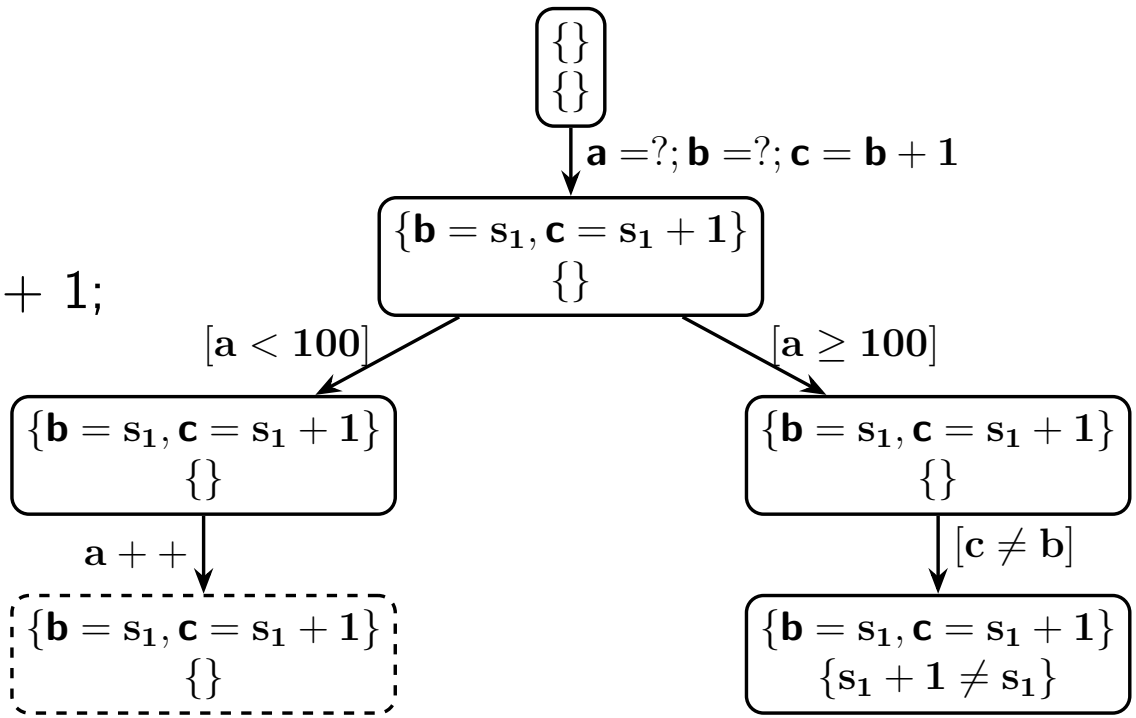
```
1  unsigned char a = ?;
2  unsigned char b = ?;
3  unsigned char c = b + 1;
4  while (a < 100)
5        a++;
6   if  (c == b)
7      error ();
```

- Necessary to track (thanks CEGAR!):
  - Symbolic memory of **b** and **c**.
  - Constraint $\mathbf{c} \neq \mathbf{b}$

Symbolic Execution

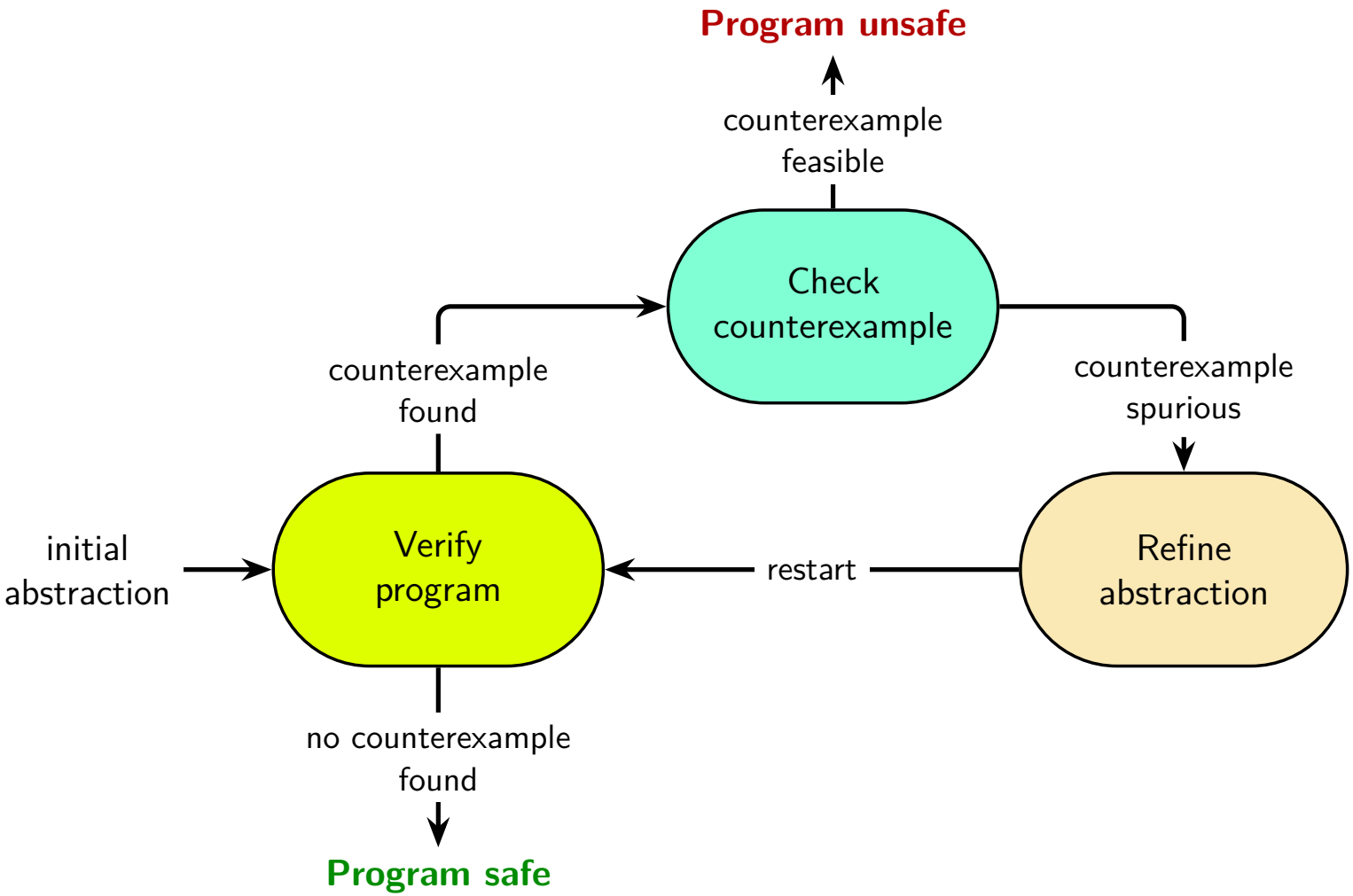
- Replace concrete test-values with generic symbolic values
- **Symbolic Memory**: Stores (symbolic) value assignments.
- **Path Constraints**: Constrain symbolic values.

```
1  unsigned char a = ?;
2  unsigned char b = ?;
3  unsigned char c = b + 1;
4  while (a < 100)
5      a++;
6  if (c == b)
7    error ();
```

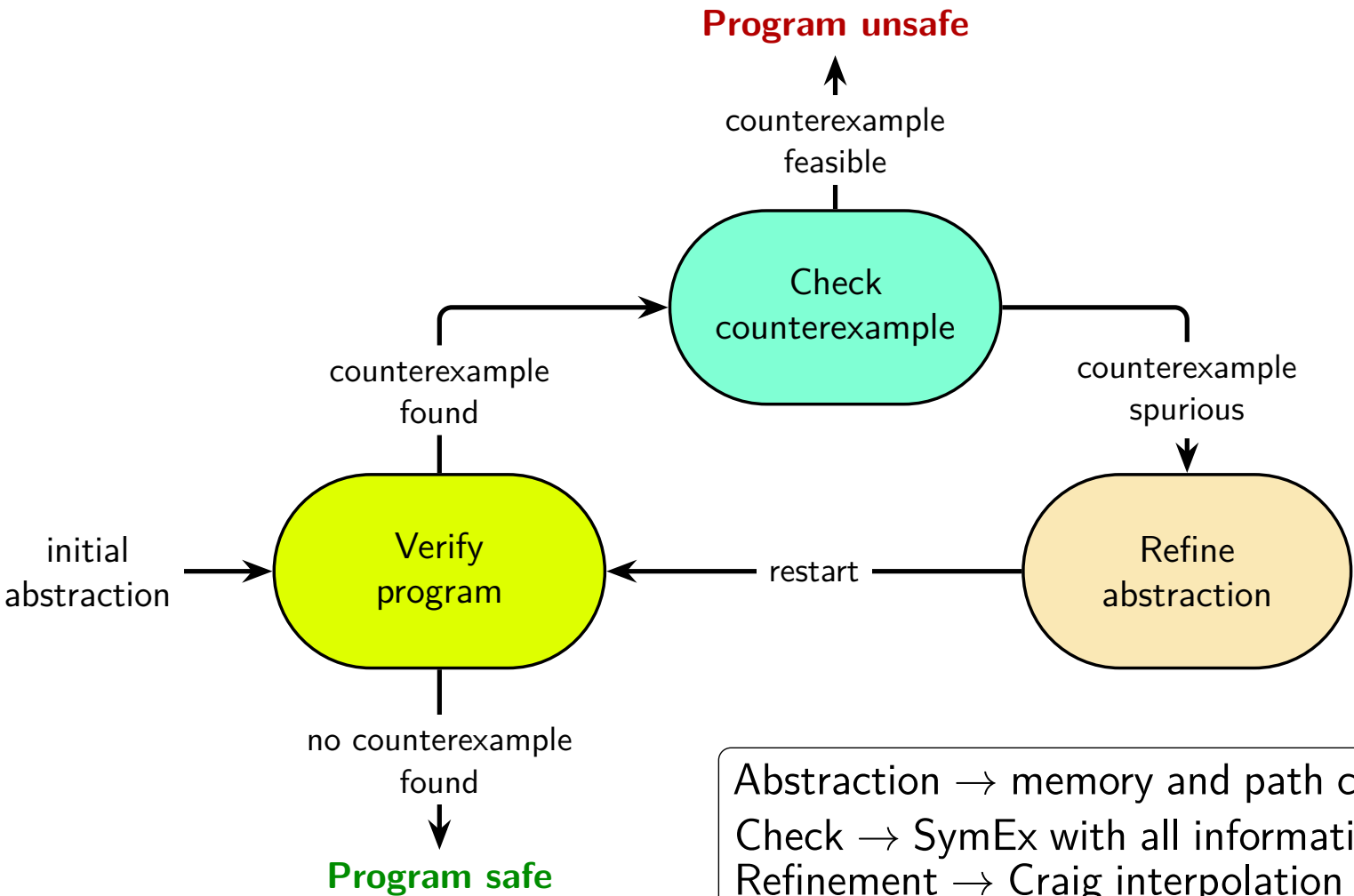


- Necessary to track (thanks CEGAR!):
  - Symbolic memory of **b** and **c**.
  - Constraint $\mathbf{c} \neq \mathbf{b}$

Counterexample-guided Abstraction Refinement



Counterexample-guided Abstraction Refinement



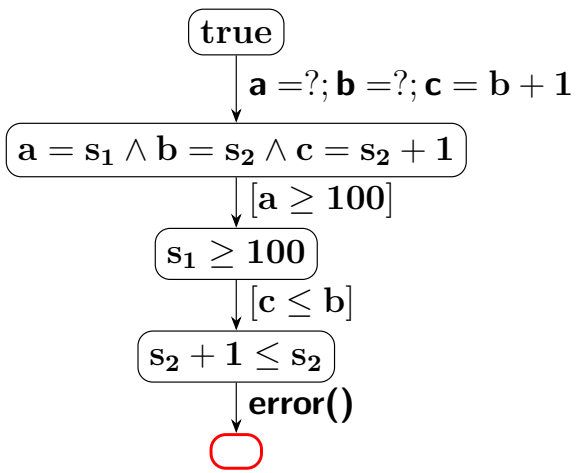
Abstraction → memory and path constraints
Check → SymEx with all information
Refinement → Craig interpolation

CPA-SymExec Output

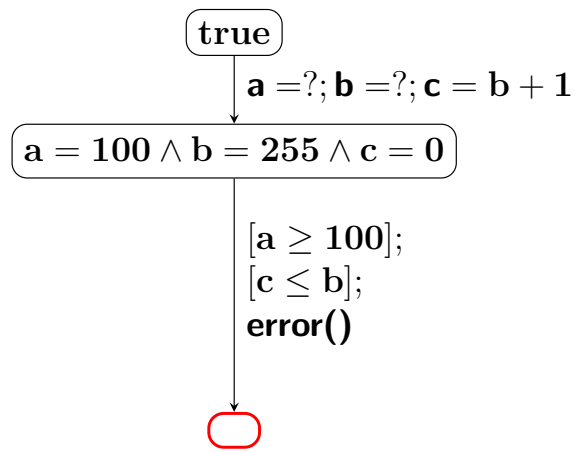
CPA-SymExec can create:

- Concrete and symbolic target paths
- Executable test cases (condition coverage)
- Interactive, visual reports

```
1   unsigned char a = ?;
2   unsigned char b = ?;
3   unsigned char c = b + 1;
4   while (a < 100)
5       a++;
6   if (c <= b)
7       error ();
```

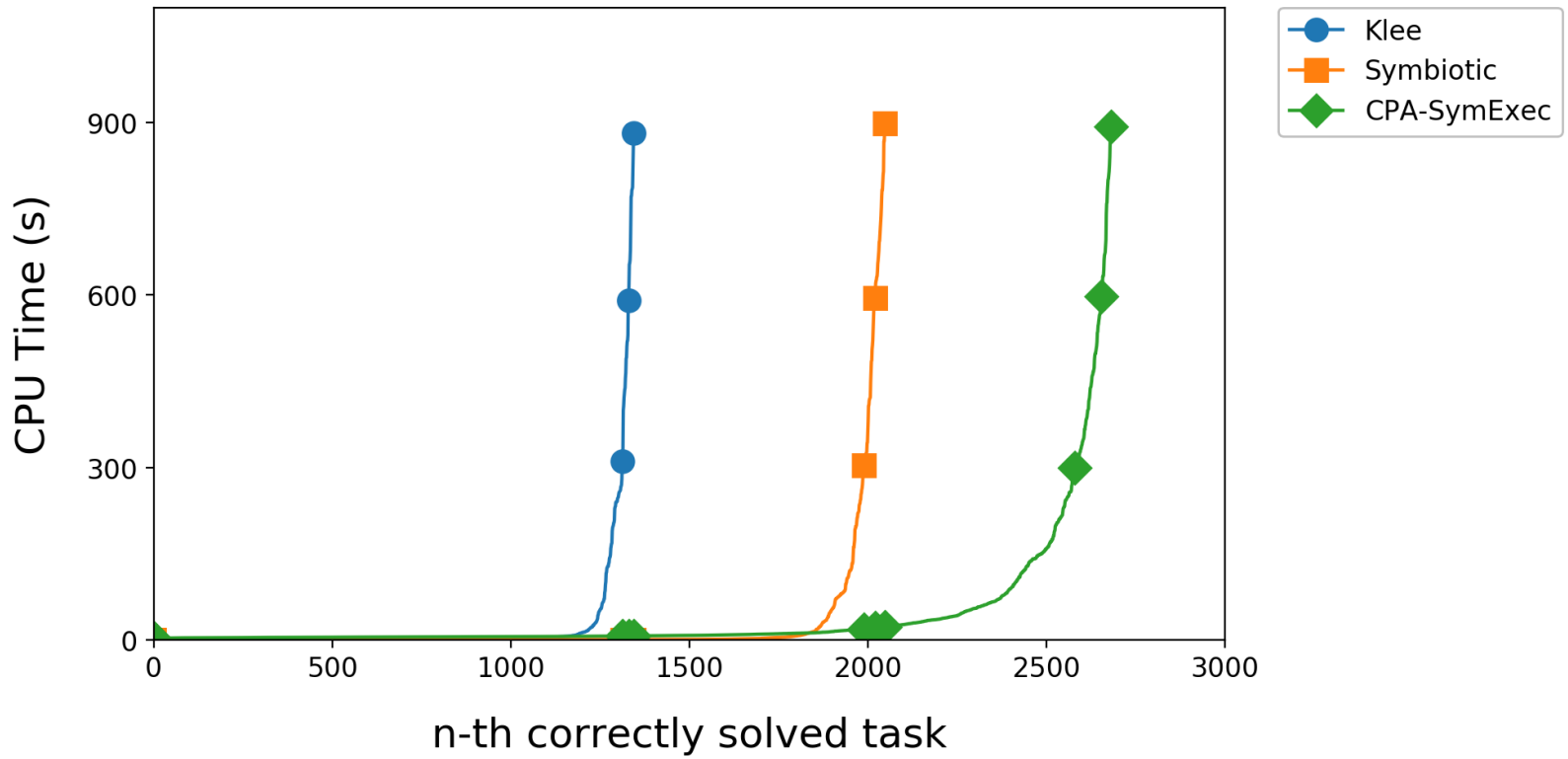


Symbolic target path



Concrete target path

Experimental Comparison



Tool	correct TRUE	correct FALSE	incorrect TRUE	incorrect FALSE
CPA-SymExec	**2137**	545	**0**	22
Klee	446	**899**	6	33
Symbiotic	1 201	848	3	**9**

- Find target path to function call/show none exists
- 5590 sequential programs from SV-COMP

Demo

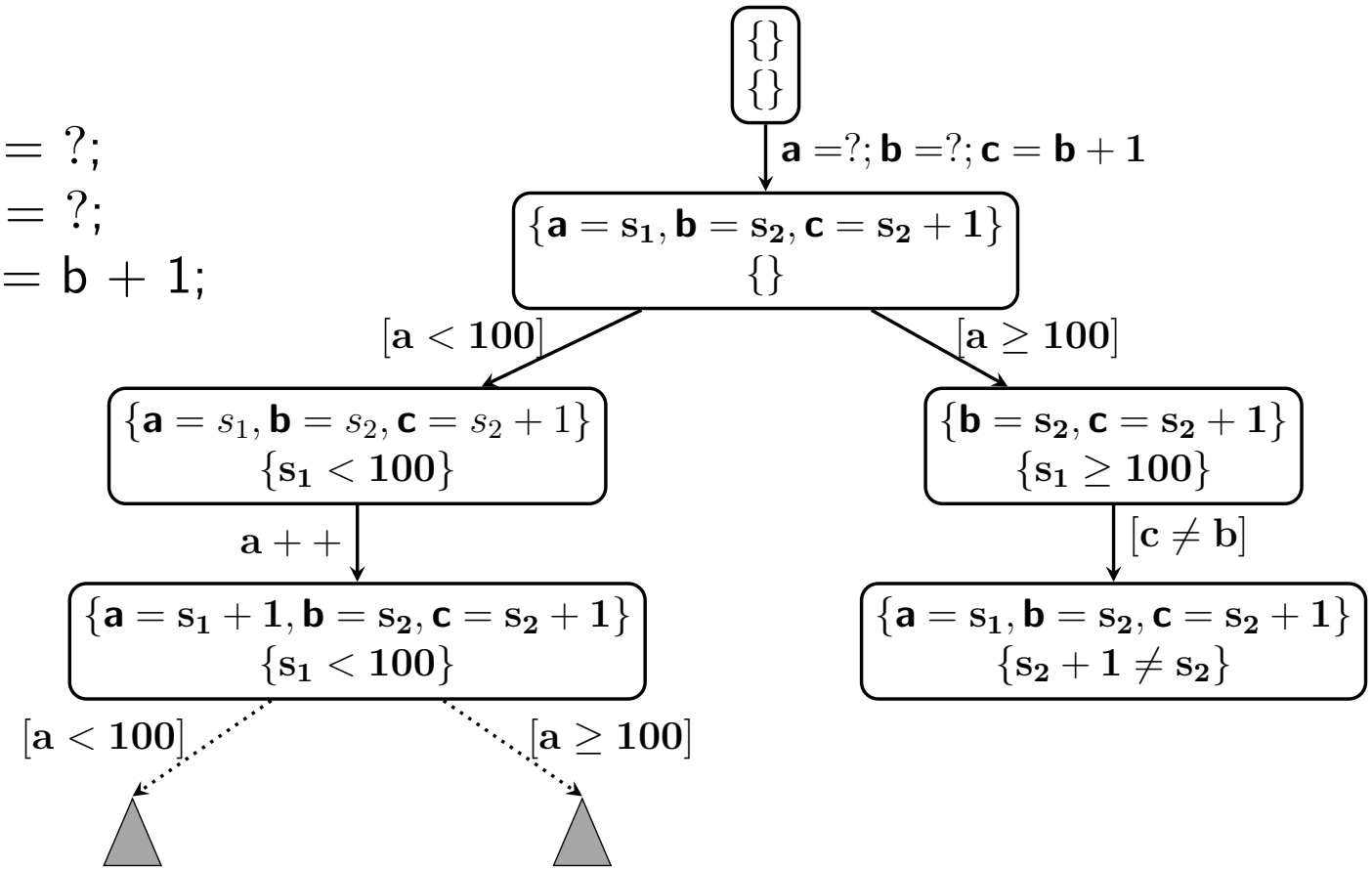
More Information

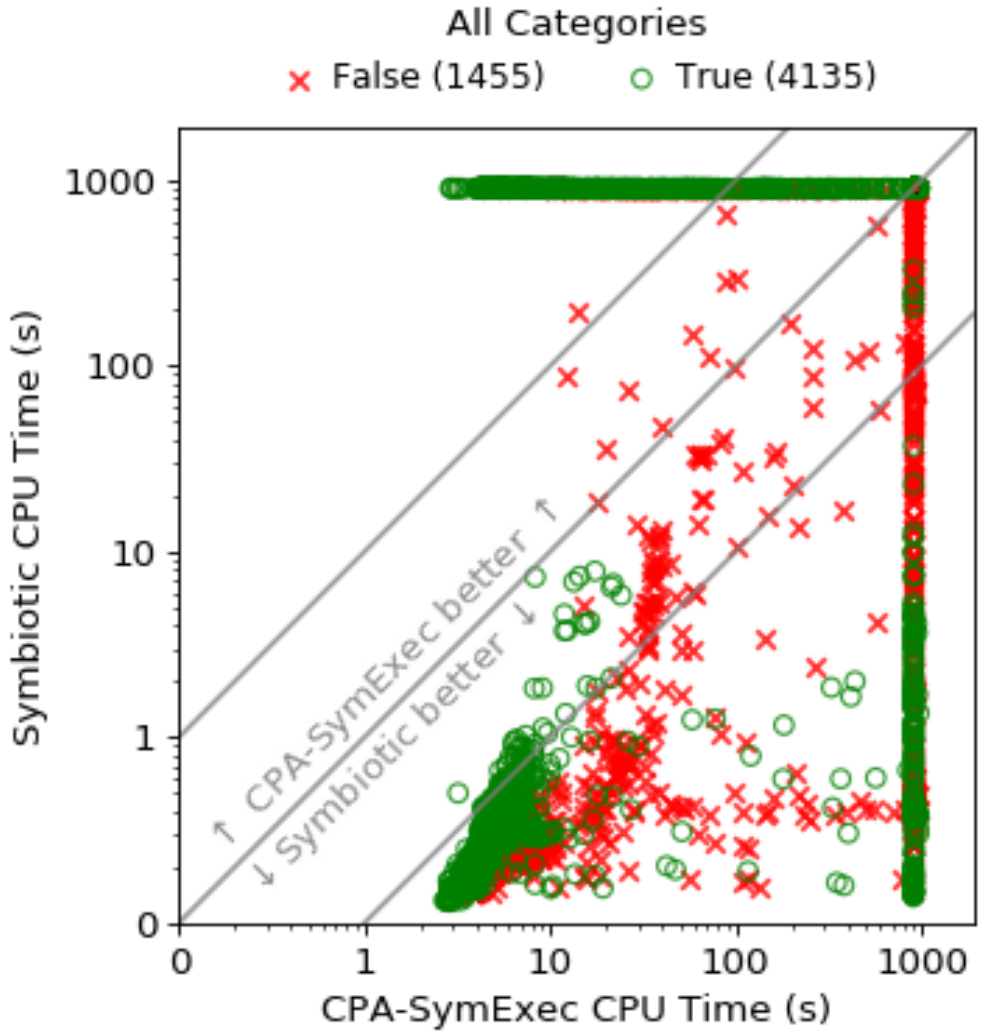
- Video: https://youtu.be/qoBHtvPKtnw
- Artifact (tool and data): https://zenodo.org/record/1321181

- Technical Session: Tomorrow, Joffre CD, 13:30–15:00

Traditional Symbolic Execution Example

```
1   unsigned char a = ?;
2   unsigned char b = ?;
3   unsigned char c = b + 1;
4   while (a < 100)
5       a++;
6   if (c == b)
7       error ();
```




All Categories
False (1455)
True (4135)
Symbiotic CPU Time (s)
CPA-SymExec CPU Time (s)
0
1
10
100
1000
↑ CPA-SymExec better ↑
↓ Symbiotic better ↓