

Dirk Beyer • Thomas Lemberger

Symbolic Execution with CEGAR

Tackling the Path Explosion Problem of
Symbolic Execution by Borrowing Counterexample-
Guided Abstraction Refinement from Model Checking



Outline

1. Symbolic Execution and Path Explosion
2. Applying Counterexample-Guided Abstraction Refinement
3. Evaluation
4. Conclusion

Symbolic Execution and Path Explosion



Symbolic Execution and Path Explosion

Symbolic Execution is so useful!

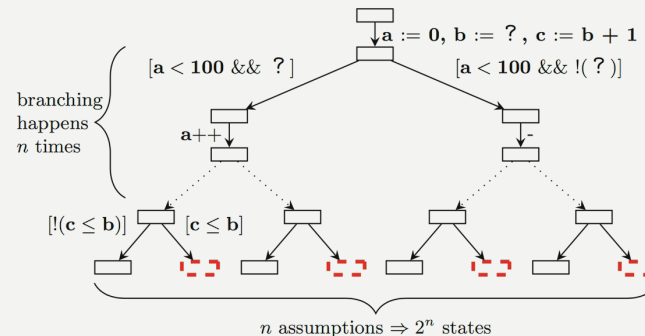
- Tracks explicit/symbolic values and constraints on symbolic values
- Handles unknown and non-deterministic values in dynamic and static analysis (external functions, unavailable libraries, `random()`)
- Test Case Generation, Error Localization, Fault Repair, Verification, Testing, ...

But does not scale well. 🤔

```

1 a := 0;
2 b := ?;
3 c := b+1;
4 while a < 100 do
5   if ? do
6     a++;
7   if c <= b do
8     error();

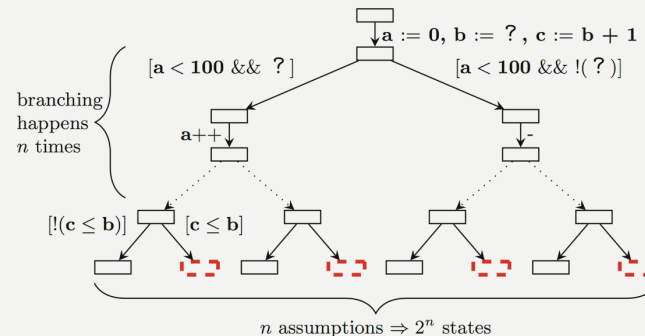
```



Symbolic Execution and Path Explosion

- Path explosion due to amount of tracked information
- But tracked information often unnecessary...
- Use CEGAR to find out what has to be tracked 🤔👉

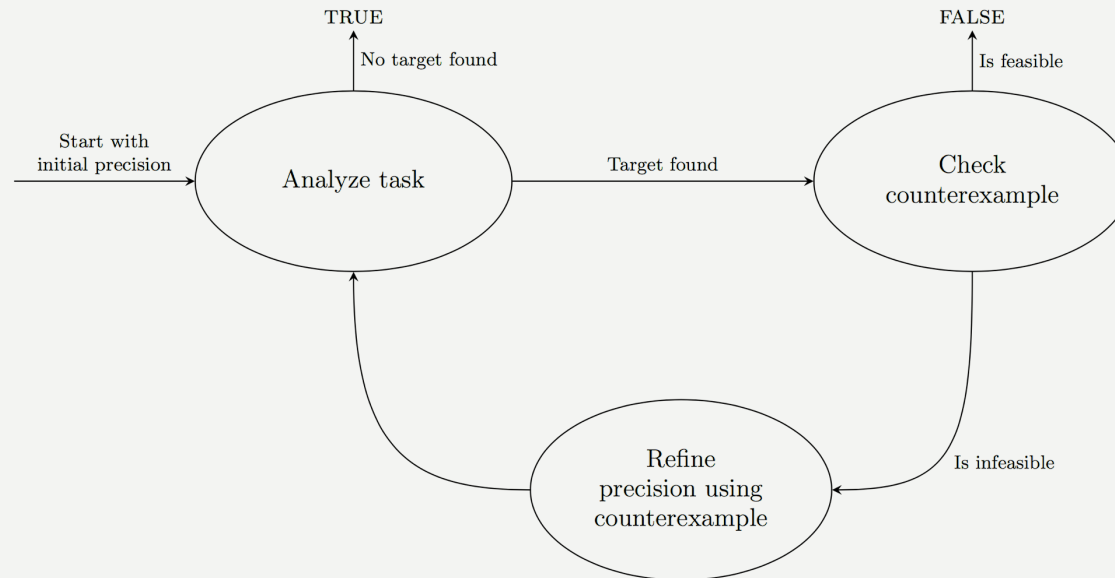
```
1 a := 0;  
2 b := ?;  
3 c := b+1;  
4 while a < 100 do  
5   if ? do  
6     a++;  
7   if c <= b do  
8     error();
```



Applying Counterexample-Guided Abstraction Refinement (CEGAR)



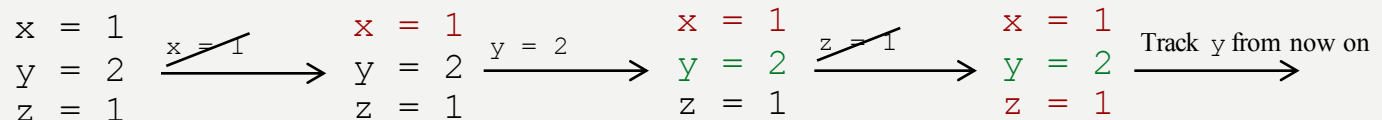
Applying CEGAR to Symbolic Execution



Applying CEGAR to Symbolic Execution

- Initially applied to model checking, already applied to explicit value analysis
- Precision refinement based on Craig interpolants
 - Start at location 0 with initial interpolant
 - Compute next value assignment and constraints based on previous interpolant
 - Filter values needed to proof trace infeasible
 - Filter constraints needed to proof trace infeasible
 - Combine values and constraints to interpolant
 - If not at last location on error trace, go to next location and continue at 2.
 - Based on interpolants, adjust precision at every location for both value and constraints tracking

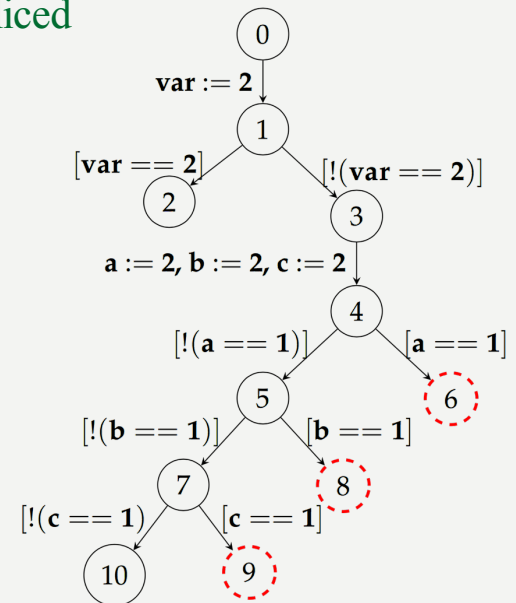
Example: Adjusting precision for value tracking



Applying CEGAR to Symbolic Execution

Further optimization: **Refinement selection**

- Choose good interpolants by computing them on sliced prefixes
- Different heuristics for prefix selection
- Influence behavior and performance significantly
- Example heuristics:
 - Variable domains
 - Interpolant width
 - Number of assumptions in prefix



Evaluation





Evaluation of Symbolic Execution with CEGAR

- Setup:
 - Cluster of Intel Xeon E5-2650 v2 CPUs at 2.60 GHz and 135 GB of memory
 - 2 CPU cores and 15 GB of memory for each verification task
 - 900s time limit
 - SV-COMP'16 task set
- Experiments:
 - Comparison of different refinement heuristics
 - Comparison of Symbolic Execution with and without CEGAR and Symbiotic 3 (based on KLEE)

Evaluation of Symbolic Execution with CEGAR

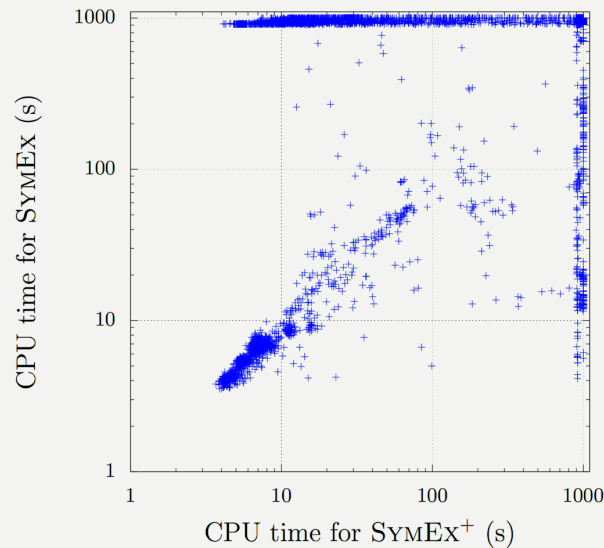
Table 1: Comparison of different refinement-selection heuristics in SYMEX⁺

Verdict	unsolved	solved	correct	correct	incorrect	incorrect
			TRUE	FALSE	TRUE	FALSE
No preference	4341	2336	1737	443	0	156
Domain good – width narrow	4444	2233	1702	531	0	171
Domain good – short	3906	2771	2042	567	0	162
Assumptions most – short	4028	2491	1892	599	0	158

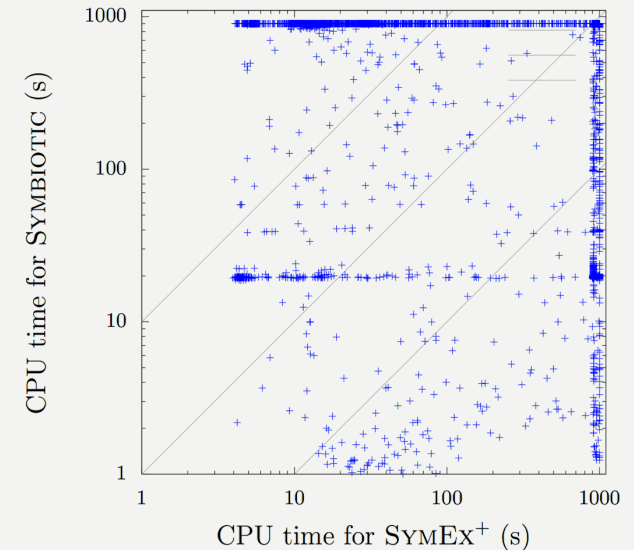
Table 2: Comparison of classical symbolic execution (SYMEX) to SYMEX⁺ (both implemented in CPACHECKER) and SYMBIOTIC (an external tool)

Verdict	unsolved	solved	correct	correct	incorrect	incorrect
			TRUE	FALSE	TRUE	FALSE
SYMEX	5756	921	171	634	1	115
SYMEX ⁺	3906	2771	2042	567	0	162
SYMBIOTIC	5388	1289	769	503	2	15

Evaluation of Symbolic Execution with CEGAR



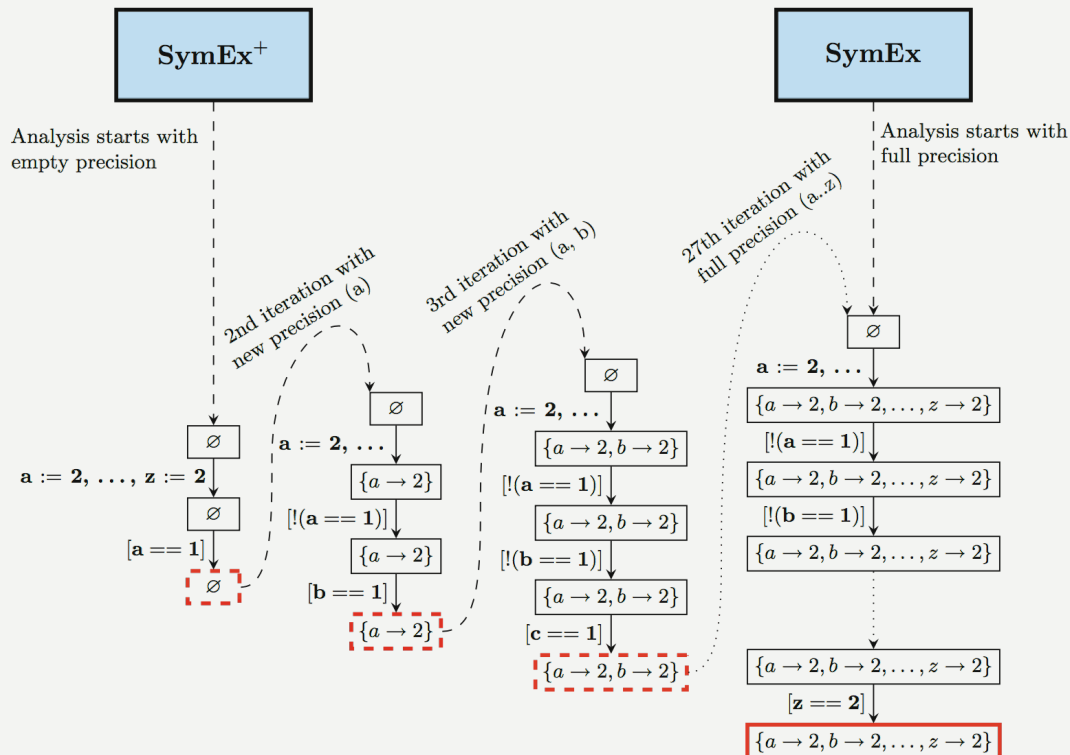
Comparison between Symbolic Execution
with CEGAR and without CEGAR



Comparison between Symbolic Execution
with CEGAR and Symbiotic 3

Evaluation of Symbolic Execution with CEGAR

Using CEGAR is not always better.



Conclusion





Conclusion

- CEGAR changes behavior of Symbolic Execution significantly
- Tracks only information really necessary for the analysis
- We choose which characteristics this information is supposed to have, using refinement selection
- ✓ Mitigates problem of path explosion
- ✓ Provides major performance boost for a significant amount of tasks
- Challenge: Existence of many error paths with different error causes

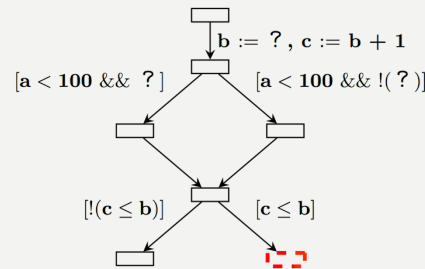
Conclusion

Symbolic Execution is so useful!

- Tracks explicit and symbolic values of execution
- Handles unknown and non-deterministic values in dynamic and static analysis (external functions, unavailable libraries, `random()`)
- Test Case Generation, Error Localization, Fault Repair, Verification, Testing, ...

And it scales! 😊

```
1 a := 0;  
2 b := ?;  
3 c := b+1;  
4 while a < 100 do  
5   if ? do  
6     a++;  
7   if c <= b do  
8     error();
```



Thank you! Questions?

