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 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, \texttt{random()})
- Test Case Generation, Error Localization, Fault Repair, Verification, Testing, …

But does not scale well. 😢

```plaintext
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

TRUE
No target found

FALSE
Is feasible

Start with initial precision

Analyze task

Target found

Check counterexample

Refine precision using counterexample

Is infeasible
Applying CEGAR to Symbolic Execution

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

Example: Adjusting precision for value tracking

\[
\begin{align*}
x &= 1 & x &= 1 & x &= 1 & x &= 1 \\
y &= 2 & y &= 2 & y &= 2 & y &= 2 \\
z &= 1 & z &= 1 & z &= 1 & z &= 1
\end{align*}
\]

Track \( y \) from now on
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^+

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

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

<table>
<thead>
<tr>
<th>Verdict</th>
<th>unsolved</th>
<th>solved</th>
<th>correct TRUE</th>
<th>correct FALSE</th>
<th>incorrect TRUE</th>
<th>incorrect FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SymEx</td>
<td>5756</td>
<td>921</td>
<td>171</td>
<td>634</td>
<td>1</td>
<td>115</td>
</tr>
<tr>
<td>SymEx^+</td>
<td>3906</td>
<td>2771</td>
<td>2042</td>
<td>567</td>
<td>0</td>
<td>162</td>
</tr>
<tr>
<td>Symbiotic</td>
<td>5388</td>
<td>1289</td>
<td>769</td>
<td>503</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>
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
  o 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! 😊

```plaintext
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?