Converting Test Goals To Condition Automata

FREDERIC SCHÖNBERGER/BACHELOR’S THESIS
Introduction

- Software correctness is important
- Testing is expensive, up to 50% of all development costs go into testing
- Hence, automatic test generators

- Test generation is hard
  - We could combine the strengths of each generator
Conditional Testing

Program \[\rightarrow\] Tester 1 \[\rightarrow\] Some goals are covered \[\rightarrow\] Pruner \[\rightarrow\] Tester 2

Reduced program
Conditions

- Automaton that describes which paths have been verified
- Assumptions: Conditions under which a path has been explored

A condition covers a path iff there is a run s.t.
- the run ends in an accepting state
- all assumptions are satisfied
Some goals are covered

Pruner

Reduced program
Some goals are covered

List of covered goals

CPAchecker’s reducer

Reduced program
Converting Test Goals to Condition Automata
Example CFA of a program with two branching if/else blocks
Example CFA of a program with two branching if/else blocks
Example CFA of a program with two branching if/else blocks
Example CFA of a program with two branching if/else blocks
Our Approach

1. Identify our goals
2. Generate a condition from these goals
3. Prune!
Phase 1: BFS

- We want to find leaf goals
- Partition them into covered/ not covered

- We can at most remove covered leaf goals
Phase 2: Condition generation

- **Covered leaf goals**: True assumption
- **Uncovered goals**: False assumption
  - To avoid issues with non-linear program flows we use all nodes
- **Everything else**: True assumption

- This condition satisfies our requirements:
  - All paths that only contain covered goals are pruned
  - Others are kept
Optimization: Propagation

**NAÏVE APPROACH**
We just identify leaf goals.

**APPROACH W/ PROPAGATION**
We merge nodes whose ancestors are all either covered or uncovered.
Evaluation
Evaluation

- Branch coverage
- Resource consumption
  - CPU
  - RAM
- Number of tasks that were successfully completed
- Benchexec as benchmarking tool, orchestrated by CoVeriTeam
- Testers participants of Test-Comp 2020
Setup

Instrumenter

Tester à 7min

Extractor

Pruner

Tester à 8min
One Tester

- One Tester, applied sequentially
- Baseline: Tester à 15min
- CondTest: Using CondTest’s reducer
Naïve vs optimized version
Two Testers

- Two combinations:
  - PRTest/ CoVeriTest
  - PRTest/ HybridTiger

- Idea: PRTest “dumb” random tester, eliminates the easy paths
Conclusion
Conclusion

- We have shown two approaches that generate condition automata from test goals.
- The approaches work well for a single tester (comparable to both baseline and CondTest).
- They suffer when being used with pairs of different testers.
- There is some evidence that there is a bug in the implementation:
  - Resource usage indicates most of the time only one tester is running.

- What are “good” combinations for testers? How to find them?
- Play around with the time limits.
- What happens if we use other testers?