Towards Cooperative Software Verification with Test Generation and Formal Verification

December 12, 2022 · PhD Defense · Software and Computational Systems Lab, Fakultät für Mathematik, Informatik und Statistik, LMU Munich
Publications included in the PhD Thesis


T. Lemberger: **Plain random test generation with PRTest**. STTT, 2020.

Publications presented here


Context
Automated Software Verification
Automated Software Verification

Verifier

true
false
unknown

No present is reachable
• Verifiers have different strengths and weaknesses
- Verifiers have different strengths and weaknesses
- Verifiers have different strengths and weaknesses
- **Cooperative Verification** tries to combine the strengths and mitigate the weaknesses
int main(void) {
    unsigned int x = 0;
    unsigned short n = nondet();
    while (x < n) {
        x += 2;
    }
    if (x % 2 == 0) {}
    else
    reach_error();
}
Background
Control-Flow Automaton (CFA)

- CFA represents control flow of program
- We consider intraprocedural, sequential programs

```c
int main(void) {
    unsigned int x = 0;
    unsigned short n = nondet();
    while (x < n) {
        x += 2;
    }
    if (x % 2 == 0) {} else reach_error();
}
```

```
x = 0
n = nondet()

[x < n] [!(x < n)]

x = x + 2 [!(x % 2 = 0)]

[x % 2 = 0] reach_error()
```

Background
Automated Software Verification

- Two approaches:
  - Automated Test Generation
  - Automated Formal Verification
Here, Test = Test Input.

A test \( t = \langle v_0, \ldots, v^n \rangle \) is a sequence of \( n \) input values for a single program execution.
Formal Verification

Common technique:
- Compute reachable (abstract) program state space.
- Any reachable state at call to `reach_error()`?
  → property violation.
Example: Predicate Abstraction

- Program state space potentially infinite
- Abstract the state space with given predicates
- Here: \( x \% 2 = 0 \)
Counterexample-Guided Abstraction Refinement (CEGAR)

- Derive program abstraction as abstract as possible and as precise as necessary
- Start with coarse precision
- Refine precision of abstract-model exploration with found infeasible counterexamples
Verification-Result Witnesses

- Increase trust in formal verification result
- Correctness witness: Description of candidate invariants
- Violation witness: Description of abstract error path

Background

Correctness Witness (Invariant Witness)

- Nodes: States with *candidate invariants*
- Edges: *source-code guards*

- Candidate invariant: Potential invariant at that state
- Source-code guard: Condition on transition

```
int main(void) {
  unsigned int x = 0;
  unsigned short n = nondet();
  while (x < n) {
    x += 2;
  }
  if (x % 2 == 0) {} else reach_error();
}
```
Violation Witness (Path Witness)

- Nodes: States
- Edges: source-code guards and state-space guards
- Accepting state: Violation reached

```c
int main(void) {
    unsigned int x = 0;
    unsigned short n = nondet();
    while (x < n) {
        x += 2;
    }
    if (x % 2 == 0) {} 
    else
        reach_error();
}
```

o/w: otherwise
Witness Validation

Witness validators use information in witness to recompute the verification result.

Success $\rightarrow$ Verification result confirmed

A condition automaton describes the already-explored state-space with source-code guards (and state-space guards).

A condition covers a program execution if its run leads to an accepting state.

```c
int main() {
    int out;
    int val = nondet();
    if (val >= 0) {
        out = val%2 * val%3;
    } else {
        out = -val;
    }
    if (out < 0) {
        reach_error();
    }
}
```

---

Conditional Verification

\[ P \models \phi \]
\[ P \not\models \phi \]

\[ P \models \phi \text{ if } \Psi \]

D. Beyer, T. A. Henzinger, M. E. Keremoglu, and P. Wendler:
Conditional Verification


Thomas Lemberger · December 12, 2022 · PhD Defense “Towards Cooperative Software Verification”
Cooperative Software Verification with Condition Automata

D. Beyer, M.-C. Jakobs, T. Lemberger, and H. Wehrheim: 
Reducer-Based Construction of Conditional Verifiers

- Conditional Verification is great!
- But only one conditional verifier: CPAchecker.

Create providers of conditions?

Create consumers of conditions?
Reducer-Based Construction of Conditional Verifiers

- $P$
- $P_r$
- $\Psi$
- $\phi$
- $P \models \phi$
- $P \not\models \phi$
- Unknown

Reducer-Based Construction of Conditional Verifiers

A mapping from program and condition to residual program is a reducer, iff:

The state space of the residual program is a superset of the original program’s state space that is not covered by the condition.

Reducers:

- Identity
- Parallel Composition
Reducer: Parallel Composition

```c
int main() {
    int out;
    int val = nondet();
    if (val >= 0) {
        out = val % 2 * val % 3;
    } else {
        out = -val;
    }
    if (out < 0) {
        reach_error();
    }
}
```
Reducer: Parallel Composition

\[
\begin{align*}
\text{l1} & : \text{val = nondet()} \\
\text{l2} & : \text{[val >= 0]} \\
\text{l3} & : \text{[! (val >= 0)]} \\
\text{l4} & : \text{out = val} \times \text{val3} \\
\text{l5} & : \text{[out < 0]} \\
\text{l6} & : \text{[! (out < 0)]} \\
\text{l7} & : \text{reach_error()} \\
\text{q0} & : \text{(l1, l2)} \\
\text{q1} & : \text{(l3, l2)} \\
\text{q2} & : \text{(l3, l4)} \\
\text{qf} & : \text{(l2, l4)} \\
\end{align*}
\]
Reducer: Parallel Composition

```c
int main() {
    int out;
    int val = nondet();
    if (val >= 0) {
        out = val%2 * val%3;
    } else {
        out = -val;
    }
    if (out < 0) {
        reach_error();
    }
}
```

Evaluation

- Reducers *Identity* and *Parallel Composition*, implemented in CPAchecker [https://gitlab.com/sosy-lab/software/cpachecker/](https://gitlab.com/sosy-lab/software/cpachecker/)
- Combinations: CPAchecker predicate abstraction + Parallel Composition + SV-COMP 2017 Overall medalists:
  - CPA-seq
  - Smack
  - Ultimate Automizer
- Tasks: 5687 ReachSafety tasks @ SV-COMP 2017
- Limits:
  - 15GB memory
  - 100s predicate analysis + 900s CPA-seq/Smack/Ultimate Automizer
- Reproduction package: [https://doi.org/10.5281/zenodo.1172228](https://doi.org/10.5281/zenodo.1172228)
Evaluation

- 820 additional tasks solved
- Each combination contributes!
Insights

- Effectiveness increases through combinations
- We need many combinations. Integrating condition format into a single verifier is not flexible enough
- Encoding in program allows to apply tools without explicit condition support
Cooperative Software Verification with Condition Automata

D. Beyer and T. Lemberger:
Conditional Testing: Off-the-Shelf Combination of Test-Case Generators.
Cooperation between Test Generators

```c
int main() {
    int i = nondet();
    if (i != 1017) {
        while (i > 1017) {
            // branch 1.1
            i--;
        }
        // branch 1.2
        // .. snip ..
    } else {
        // branch 2
        // .. snip ..
    }
}
```

- Goal: Create test suite that reaches all branches
- Random tester: unlikely to enter `else`-branch
- Symbolic execution: may hang in `while`-loop
Cooperation between Test Generators

```c
int main() {
    int i = nondet();
    if (i != 1017) {
        while (i > 1017) {
            // branch 1.1
            i--;
        }
        // branch 1.2
        // .. snip ..
    } else {
        // branch 2
        // .. snip ..
    }
}
```

Cooperation between Test Generators

```c
int main() { 
    int i = nondet(); 
    if (i != 1017) {
        while (i > 1017) {
            // branch 1.1
            i--;
        }
        // branch 1.2
        // .. snip ..
    } else {
        // branch 2
        // .. snip ..
    }
}
```
Cooperation between Test Generators

```c
int main() {
    int i = nondet();
    if (i != 1017) {
        while (i > 1017) {
            // branch 1.1
            i--;
        }
        // branch 1.2
        // .. snip ..
    } else {
        // branch 2
        // .. snip ..
    }
}
```
Conditional Testing

Program

\[ \Psi_0 \] Covered Test Goals

\[ \Psi_1 \] Covered Test Goals (accumulated)

Test Suite

Condition = Covered Test Goals

Coverage Criterion

\[ \varphi \]

\begin{verbatim}
int main() {
    int i = nondet();
    if (i != 1017) {
        while (i > 1017) {
            // branch 1.1
            i--;
        }
        // branch 1.2
        // .. snip ..
    } else {
        // branch 2
        // .. snip ..
    }
}
\end{verbatim}

Random Tester

branch 1.1, branch 1.2

Symbolic Execution

branch 1.1, branch 1.2, branch 2
Conditional Testing

Problem: We just came up with this!
→ Turn existing testers into conditional testers.

- Condition Consumer: Reducer
- Condition Provider: Test-Goal Extractor
Reducer for Conditional Testing

**Requirement:** Reachability Equivalence

Each program input that reaches a test goal in the residual program reaches the same test goal in the original program.

**Reducers:**
- Identity
- Pruning
Pruning Reducer

- Stop program execution if it can’t reach any remaining goal
- Here: syntactic reachability

```c
int i = nondet();
if (i != 1017) {
    while (i > 1017) {
        // branch 1.1
        i--;
    }
    // branch 1.2
    // .. snip ..
} else {
    // branch 2
    // .. snip ..
}
```

```c
int i = nondet();
if (i != 1017) {
    exit(1);
} else {
    // branch 2
    // .. snip ..
}
```
Test-Goal Extractor

- Program
- Coverage Criterion
- Test Suite
gcov-based Test-Goal Extractor

- Test execution + coverage measurement
- Read covered test goals from measurement

```c
int i = nondet();
if (i != 1017) {
    while (i > 1017) {
        // branch 1.1
        i--;
    } // branch 1.2
    // .. snip ..
} else {
    // branch 2
    // .. snip ..
}
```

\( \varphi: \text{cover branches} \)

\( i \mapsto 1200 \)
Off-the-shelf Tester to Conditional Tester

- Program
- Coverage Criterion
- Covered Test Goals
- Remaining Goals
- Residual Program
- Off-the-shelf Tester
- Test Suite
- extractor
- Covered Test Goals

Evaluation

- Components implemented as CondTest
  [https://gitlab.com/sosy-lab/software/conditional-testing](https://gitlab.com/sosy-lab/software/conditional-testing)
- Tools from Test-COMP 2019: CoVeriTest, CPA-Tiger, Klee
- Tasks: 1720 Cover-Branches tasks @ Test-Comp 2019
- Limits: 900 s CPU time, 15 GB memory

- Reproduction package: [https://doi.org/10.5281/zenodo.3352401](https://doi.org/10.5281/zenodo.3352401)
Evaluation

- Branch coverage of created test suites (%), per task
- Tool standalone, 900s (x-axis)
- tester\textsuperscript{seq}: CPA-Tiger + CoVeriTest + Klee, 300s each (y-axis)
Evaluation

CPA-Tiger + CoVeriTest + Klee, 300s each

<table>
<thead>
<tr>
<th>Task</th>
<th>branch coverage</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>prune</td>
</tr>
<tr>
<td>mod3.c.v+sep-reducer</td>
<td>75.0</td>
<td>+ 5.00</td>
<td>80.0</td>
</tr>
<tr>
<td>Problem07_label35</td>
<td>52.0</td>
<td>+ 2.00</td>
<td>54.0</td>
</tr>
<tr>
<td>Problem07_label37</td>
<td>54.2</td>
<td>+ 1.97</td>
<td>56.2</td>
</tr>
<tr>
<td>Problem04_label35</td>
<td>79.5</td>
<td>+ 1.79</td>
<td>81.3</td>
</tr>
<tr>
<td>Problem06_label02</td>
<td>57.0</td>
<td>+ 1.70</td>
<td>58.7</td>
</tr>
<tr>
<td>Problem06_label27</td>
<td>57.5</td>
<td>+ 1.09</td>
<td>58.6</td>
</tr>
<tr>
<td>Problem04_label02</td>
<td>80.2</td>
<td>+ 1.06</td>
<td>81.3</td>
</tr>
<tr>
<td>Problem06_label18</td>
<td>57.5</td>
<td>+ 1.05</td>
<td>58.6</td>
</tr>
<tr>
<td>Problem04_label16</td>
<td>79.1</td>
<td>+ 1.01</td>
<td>80.1</td>
</tr>
<tr>
<td>Problem04_label34</td>
<td>80.2</td>
<td>+ 0.99</td>
<td>81.2</td>
</tr>
</tbody>
</table>
Insights

● Effectiveness increases through combinations
● Encoding in program allows to apply testers without explicit condition support
Decomposing Verification Techniques

Motivation: CEGAR

- Common underlying schema
- Many tools implement CEGAR
- New idea → new implementation (lock-in effect)
Decomposing CEGAR

- Task
  - \( P, \varphi \)
- Abstract-Model Exploration
  - potential counterexample
  - precision increment
- Precision Refinement
- Feasibility Check
  - infeasible counterexample
  - program incorrect
  - program correct
- Potential Counterexample
- Infeasible Counterexample
- Abstract-Model Explorer
- Precision Refiner
- Precision Increment
- Feasibility Checker
- Potential Counterexample
- Infeasible Counterexample
- program correct
- program incorrect
Decomposing CEGAR

Exchange formats from SV-COMP → wide tool support

- Abstract description of counterexample
- Abstract description of rejected counterexample ("violation" witness)
- Description of candidate invariants ("correctness" witness)
Component-based CEGAR (C-CEGAR)
Evaluation

- **Implementation in CoVeriTeam**
  

- **Tools:**
  - CPAchecker with improvements
  - Ultimate Automizer SV-COMP 2021
  - FShell-witness2test SV-COMP 2021

- **Tasks:** 8347 ReachSafety tasks @ SV-COMP 2021

- **Limits:** 900s CPU time, 15GB memory

- **Reproduction package:** [https://doi.org/10.5281/zenodo.6062602](https://doi.org/10.5281/zenodo.6062602)
Evaluation

1. Constant overhead.
2. Lost predicates through invariant witnesses.

1. Benefit of different components?
## Evaluation

### Benefit of different components

**RQ 3.1: C-PredWit + different feasibility checker** *(with precision refiner CPAchecker)*

<table>
<thead>
<tr>
<th></th>
<th>overall</th>
<th>proof</th>
<th>correct</th>
<th>alarm</th>
<th>unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPAchecker</td>
<td>2 854</td>
<td>2 110</td>
<td>494</td>
<td>744</td>
<td>441</td>
</tr>
<tr>
<td>fShell-witness2Test</td>
<td>1 223</td>
<td>1 126</td>
<td>0</td>
<td>97</td>
<td>64</td>
</tr>
<tr>
<td>uAutomizer</td>
<td>1 941</td>
<td>1 614</td>
<td>4</td>
<td>327</td>
<td>29</td>
</tr>
</tbody>
</table>

**RQ 3.2: C-PredWit + different precision refiner** *(with feasibility checker CPAchecker)*

<table>
<thead>
<tr>
<th></th>
<th>overall</th>
<th>proof</th>
<th>correct</th>
<th>alarm</th>
<th>unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPAchecker</td>
<td>2 854</td>
<td>2 110</td>
<td>709</td>
<td>744</td>
<td>436</td>
</tr>
<tr>
<td>uAutomizer</td>
<td>1 739</td>
<td>1 430</td>
<td>29</td>
<td>309</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

- Improved effectiveness of verification
- Improved opportunities for cooperation
- Backed by rigorous experimental evaluation and reproduction packages
Backup Slides
Backup Reducers: Algorithm

Algorithm 1 REDUCER

Input: CFA $C = (L, \ell_0, G)$ \(\triangleright\) original program

\[ CA_A = \langle Q, \Sigma, \delta, q_0, F \rangle \text{ s.t. } q_r \notin Q \] \(\triangleright\) condition automaton

Output: CFA $C_r = (L_r, \ell_0, G_r)$ \(\triangleright\) residual program

1: $L_r := \{(\ell_0, q_0)\}; \ell_{0,r} := (\ell_0, q_0); G_r := \emptyset$;
2: waitlist := $L_r$;
3: while waitlist \(\neq\) $\emptyset$ do
4: choose $(\ell_1, q_1) \in$ waitlist; remove $(\ell_1, q_1)$ from waitlist;
5: for each $g = (\ell_1, op, \ell_2) \in G$ do
6: if $q_1 \in Q \land \exists (q_1, (G_1, true), q_2) \in \delta$ s.t. $g \in G_1$ then
7: for each $(q_1, (G_1, true), q_2) \in \delta$ s.t. $g \in G_1$ do
8: if $q_2 \notin F \land (\ell_2, q_2) \notin L_r$ then
9: waitlist := waitlist \(\cup\) $(\ell_2, q_2)$;
10: $L_r := L_r \cup \{(\ell_2, q_2)\}$;
11: $G_r := G_r \cup \{(\ell_1, q_1, op, (\ell_2, q_2))\}$;
12: else
13: if $(\ell_2, q_r) \notin L_r$ then
14: waitlist := waitlist \(\cup\) $(\ell_2, q_r)$;
15: $L_r := L_r \cup \{(\ell_2, q_r)\}$;
16: $G_r := G_r \cup \{(\ell_1, q_1, op, (\ell_2, q_r))\}$;
17: return $C_r$
Backup Reducers: Evaluation

- Combinations: CPAchecker predicate analysis + SV-COMP 2017 overall medalists:
  - CPA-seq
  - Smack
  - Ultimate Automizer

- Tasks: 5687 ReachSafety tasks @ SV-COMP 2017
  - 1501 unsafe tasks
  - 4186 safe tasks

- Limits: 900s CPU time, 15 GB memory
  - 100s predicate analysis + 900s CPA-seq/Smack/Ultimate Automizer

Intel Xeon E3-1230 v5 CPU with 8 processing units each, a frequency of 3.4 GHz, 33 GB of memory, and an Ubuntu 16.04 operating system with Linux kernel 4.4.
## Backup Reducers: Evaluation

<table>
<thead>
<tr>
<th></th>
<th>CPAseq</th>
<th>SMACK</th>
<th>UAuto</th>
<th>CPAseq</th>
<th>SMACK</th>
<th>UAuto</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>513</td>
<td>415</td>
<td>238</td>
<td>789</td>
<td>695</td>
<td>789</td>
</tr>
<tr>
<td>Correct proof</td>
<td>265</td>
<td>76</td>
<td>170</td>
<td>387</td>
<td>296</td>
<td>386</td>
</tr>
<tr>
<td>Correct alarm</td>
<td>248</td>
<td>339</td>
<td>68</td>
<td>402</td>
<td>399</td>
<td>403</td>
</tr>
<tr>
<td>Incorrect</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Incorrect proof</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Incorrect alarm</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Unknown</td>
<td>307</td>
<td>405</td>
<td>575</td>
<td>31</td>
<td>125</td>
<td>27</td>
</tr>
<tr>
<td>Total</td>
<td>820</td>
<td>820</td>
<td>820</td>
<td>820</td>
<td>820</td>
<td>820</td>
</tr>
</tbody>
</table>
Backup Reducers: Evaluation

<table>
<thead>
<tr>
<th>Task</th>
<th>R</th>
<th>CPASEQ</th>
<th>SMACK</th>
<th>UAutomizer</th>
<th>+CPASEQ</th>
<th>+SMACK</th>
<th>+UAutomizer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>S</td>
<td>t (s)</td>
<td>M (GB)</td>
<td>S</td>
<td>t (s)</td>
<td>M (GB)</td>
</tr>
<tr>
<td>lin-4.2 vlsi_ir</td>
<td>T</td>
<td>X</td>
<td>910</td>
<td>7.9</td>
<td>X</td>
<td>890</td>
<td>0.97</td>
</tr>
<tr>
<td>lin-3.14 vsp1</td>
<td>T</td>
<td>X</td>
<td>920</td>
<td>6.9</td>
<td>X</td>
<td>890</td>
<td>0.70</td>
</tr>
<tr>
<td>lin-3.14 vxge</td>
<td>T</td>
<td>X</td>
<td>930</td>
<td>11</td>
<td>X</td>
<td>190</td>
<td>14</td>
</tr>
<tr>
<td>lin-4.2 w83781d</td>
<td>T</td>
<td>X</td>
<td>910</td>
<td>6.7</td>
<td>X</td>
<td>900</td>
<td>3.7</td>
</tr>
<tr>
<td>lin-4.2 zd1211rw</td>
<td>T</td>
<td>X</td>
<td>930</td>
<td>6.3</td>
<td>X</td>
<td>890</td>
<td>0.96</td>
</tr>
<tr>
<td>lin-3.14 vmxnet3</td>
<td>T</td>
<td>X</td>
<td>930</td>
<td>6.9</td>
<td>X</td>
<td>890</td>
<td>1.2</td>
</tr>
<tr>
<td>lin-3.14 skge</td>
<td>T</td>
<td>X</td>
<td>950</td>
<td>7.3</td>
<td>X</td>
<td>940</td>
<td>3.6</td>
</tr>
<tr>
<td>lin-3.16 ath5k</td>
<td>T</td>
<td>X</td>
<td>950</td>
<td>5.9</td>
<td>X</td>
<td>950</td>
<td>4.7</td>
</tr>
<tr>
<td>lin-3.14 ipw2200</td>
<td>T</td>
<td>X</td>
<td>950</td>
<td>7.6</td>
<td>X</td>
<td>950</td>
<td>6.6</td>
</tr>
<tr>
<td>lin-3.14 bttv</td>
<td>T</td>
<td>X</td>
<td>950</td>
<td>5.8</td>
<td>X</td>
<td>910</td>
<td>5.0</td>
</tr>
<tr>
<td>lin-4.2 cciss</td>
<td>T</td>
<td>X</td>
<td>920</td>
<td>7.1</td>
<td>X</td>
<td>330</td>
<td>12</td>
</tr>
<tr>
<td>floodmax.4</td>
<td>T</td>
<td>X</td>
<td>910</td>
<td>3.0</td>
<td>X</td>
<td>880</td>
<td>0.53</td>
</tr>
<tr>
<td>sep20</td>
<td>T</td>
<td>X</td>
<td>900</td>
<td>3.2</td>
<td>X</td>
<td>880</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Backup Reducers: Evaluation

Challenge: Blow-up of program size

Relation program size before reduction
/ program size after reduction:

- Min: 0.0006
- Mean: 0.14
- Max: 11.5
Backup Reducers: Evaluation

Challenge: Blow-up of program size

Relation program size before reduction
/ program size after reduction:

- Min: 0.0006
- Mean: 0.14
- Max: 11.5
Backup Reducers: Evaluation

Challenge: Blow-up of program size

Relation program size before reduction /
program size after reduction:

- Min: 0.0006
- Mean: 0.14
- Max: 11.5

Backup CondTest: Goal Annotation

```cpp
int main() {
    int x = input();
    if (x != 161) {
        GOAL_47;
        // ...
    } else {
        GOAL_67;
        // ...
    }
}
```
Backup CondTest: Verification Witnesses to Tests

- Reducer: identity + annotate goals with \_\_VERIFIER\_error
- Apply cyclic tester

\[ \varphi \xrightarrow{\text{crit-to-spec}} \phi \]

\[ P \xrightarrow{\text{Formal Verifier}} \xrightarrow{\text{Witness-to-test}} \text{Test Case} \]

\[ \Psi \xrightarrow{\text{reducer}^{\text{annot}}} \xrightarrow{\text{tester}^{\text{cond}}} \xrightarrow{\text{extractor}^{\text{exec}}} \text{Test Suite} \]
Fig. 15. Branch coverage of test suites created by original tools vs. their integration in $\text{tester}^{\text{cond}}$ (in percent)
Motivation: CEGAR (the good)

```
1 int main(void) { 
2     unsigned int x = 0;
3     unsigned short n = nondet();
4     while (x < n) { 
5         x += 2;
6     }
7     if (x % 2 == 0) {} 
8     else 
9         reach_error();
10 }
```
Motivation: CEGAR (the bad)


Craig Interpolation

```c
int main(void) {
    unsigned int x = 0;
    unsigned short n = nondet();
    while (x < n) {
        x += 2;
    }
    if (x % 2 == 0) {}
    else
        reach_error();
}
```
### Backup CondTest: Evaluation with Verifier

vb: CPA-Tiger + CoVeriTest + Klee, 200s

each + ESBMC, 300s

<table>
<thead>
<tr>
<th>Task</th>
<th>branch coverage</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>prune</td>
<td>→</td>
<td>vb</td>
</tr>
<tr>
<td>Problem08_label30</td>
<td>5.72 + 56.2</td>
<td></td>
<td>62.0</td>
</tr>
<tr>
<td>Problem08_label32</td>
<td>5.72 + 56.1</td>
<td></td>
<td>61.9</td>
</tr>
<tr>
<td>Problem08_label06</td>
<td>5.72 + 56.1</td>
<td></td>
<td>61.8</td>
</tr>
<tr>
<td>Problem08_label35</td>
<td>5.72 + 56.0</td>
<td></td>
<td>61.7</td>
</tr>
<tr>
<td>Problem08_label00</td>
<td>5.72 + 55.9</td>
<td></td>
<td>61.6</td>
</tr>
<tr>
<td>Problem08_label11</td>
<td>5.72 + 55.8</td>
<td></td>
<td>61.5</td>
</tr>
<tr>
<td>Problem08_label19</td>
<td>5.72 + 55.7</td>
<td></td>
<td>61.5</td>
</tr>
<tr>
<td>Problem08_label29</td>
<td>5.67 + 55.7</td>
<td></td>
<td>61.4</td>
</tr>
<tr>
<td>Problem08_label22</td>
<td>5.72 + 55.7</td>
<td></td>
<td>61.5</td>
</tr>
<tr>
<td>Problem08_label56</td>
<td>5.72 + 55.7</td>
<td></td>
<td>61.5</td>
</tr>
</tbody>
</table>
Backup C-CEGAR: CoVeriTeam Configuration

```java
1 explorer = ActorFactory.create(ProgramValidator,
2     "cpa-predicate-NoRefinement.yml");
3 checker = ActorFactory.create(ProgramValidator,
4     "cpa-validate-violation-witnesses.yml");
5 refiner = ActorFactory.create(ProgramValidator,
6     "uautomizer.yml");
```

**Figure 9: Example configuration of C-CEGAR components in CoVeriTeam**
Backup C-CEGAR: Issues with Witness Usage

```c
1 int main(void) {
2     unsigned int x = 1;
3     unsigned int y = 0;
4
5     while (y < 1024) {
6         x = 0;
7         y++;
8     }
9     if (x == 0) {}
10     else
11         error();
12 }
```

- $y = 0$ after one unrolling: $x = 0$
- Line 5, cond-true
- Line 5, cond-false
- Line 9, cond-false
- $q_{err}$
Evaluation

1. Overhead of a stateless, component-based approach ($C\text{-Pred}$)?

**Efficiency:**

- Overhead of a stateless, component-based approach ($C\text{-Pred}$)?
  - 6.5% decrease
  - Modulo runtime limit: 1.7% decrease
    - Reason: different counterexample check

**Effectiveness:**

- 6.5% decrease
- Modulo runtime limit: 1.7% decrease
  - Reason: different counterexample check
Backup C-CEGAR: Evaluation

Table 1: Comparison of CPAchecker’s predicate abstraction and the component-based version in two variations

<table>
<thead>
<tr>
<th></th>
<th>correct</th>
<th></th>
<th>incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>overall</td>
<td>proof</td>
<td>alarm</td>
</tr>
<tr>
<td>Pred</td>
<td>3,769</td>
<td>2,556</td>
<td>1,213</td>
</tr>
<tr>
<td>C-Pred</td>
<td>3,524</td>
<td>2,450</td>
<td>1,074</td>
</tr>
<tr>
<td>C-PredWit</td>
<td>2,854</td>
<td>2,110</td>
<td>744</td>
</tr>
</tbody>
</table>

- C-* Impact on effectiveness: 6.5% decrease.
- Accounting for the speed difference: 1.7% decrease
- Witness Impact on effectiveness: 20% decrease.
Backup C-CEGAR: Evaluation

Stateless, component-based approach (C-Pred) vs. internal CEGAR (Pred)

- Decompose internal CEGAR with Craig interpolation
- Use proprietary predicate map (predmap) to communicate precision increment
Backup C-CEGAR: Evaluation

Stateless, component-based approach ($C{-}\text{Pred}$) vs. internal CEGAR ($\text{Pred}$)

Efficiency:
- Constant-size overhead of 13

Effectiveness:
- 6.5% decrease
- With increased runtime limit: down to 1.7% decrease
  - Reason: different counterexample check
Backup C-CEGAR: Evaluation

Median factor of increase in run time

Number of CEGAR iterations to solve task

0.9  3.1  4.7  4.4  8.0  7.3  6.6  6.0  6.2  5.2

1  2  3  4  5  6  7  8  9  10
Backup C-CEGAR: Evaluation

Exchange formats: Predmap (C-Pred) vs. Invariant Witnesses (C-PredWit)

- Efficiency: No impact
- Impact on effectiveness: 20% decrease
  - Computed predicates are not consistently added to invariant witness
Example: Enumerative Algorithm

Formal Verification

\[ l_1 \]
\[ x = 0 \]
\[ l_2 \]
\[ n = \text{nondet()} \]
\[ l_3 \]
\[ [x < n] \]
\[ ![x < n] \]
\[ l_4 \]
\[ x = x + 2 \]
\[ l_5 \]
\[ [x \% 2 = 0] \]
\[ l_6 \]
\[ ![x \% 2 = 0] \]
\[ l_7 \]
\[ [x \% 2 = 0] \]
\[ l_8 \]
\[ \text{reach_error()} \]
\[ l_9 \]
\[ l_{10} \]
Example: Enumerative Algorithm

Formal Verification
Example: Enumerative Algorithm

Formal Verification

- $l_1$: $x = 0$
- $l_2$: $n = \text{nondet}()$
- $l_3$: $[x < n]$
- $l_4$: $[(x < n)]$
- $l_5$: $x = x + 2$
- $l_6$: $[!(x \% 2 = 0)]$
- $l_7$: $l_9$: $\text{reach\_error}()$
- $l_8$: $l_3$, $x \leftrightarrow 0$, $n \leftrightarrow 0$
- $l_9$: $l_1$, true

Choose one possible value
Example: Enumerative Algorithm

\[ x = 0 \]

\[ n = \text{nondet()} \]

\[ x < n \]

\[ x \mod 2 = 0 \]

\[ x \rightarrow 0, n \rightarrow 0 \]
Example: Enumerative Algorithm

\[
\begin{align*}
l_1 & \quad x = 0 \\
l_2 & \quad n = \text{nondet}() \\
l_3 & \quad [x < n] \\
l_4 & \quad [!(x < n)] \\
l_5 & \quad x = x + 2 \\
l_7 & \quad [!(x \% 2 = 0)] \\
l_6 & \quad \text{reach_error}() \\
l_10 & \quad \\
l_2 & \quad l_2, x \mapsto 0 \\
l_3 & \quad l_3, x \mapsto 0, n \mapsto 0 \\
l_7 & \quad l_7, x \mapsto 0, n \mapsto 0 \\
l_1, true & \quad 
\end{align*}
\]
Example: Enumerative Algorithm

Formal Verification

x = 0
n = nondet()
[x < n]
[l1]
[l2] x = x + 2
[l3] ![x \% 2 = 0]
[l4] ![x < n]
[l7]  
[l10] reach_error()
Example: Enumerative Algorithm

Formal Verification

Thomas Lemberger · December 12, 2022 · PhD Defense “Towards Cooperative SoftwareVerification”