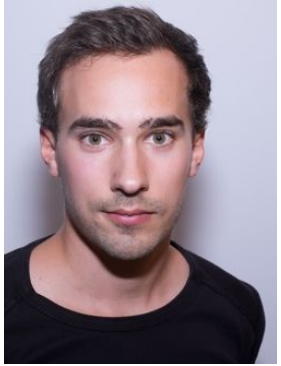
Thomas Lemberger

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

*D. Beyer, M.-C. Jakobs, T. Lemberger, and H. Wehrheim: **Reducer-Based Construction of Conditional Verifiers**. Proc. ICSE, 2018.*

*D. Beyer and T. Lemberger: **Conditional Testing: Off-the-Shelf Combination of Test-Case Generators**. Proc. ATVA, 2019.*

*D. Beyer, M.-C. Jakobs, and T. Lemberger: **Difference Verification with Conditions**. Proc. SEFM, 2020.*

*D. Beyer, J. Haltermann, T. Lemberger, and H. Wehrheim: **Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR**. Proc. ICSE, 2022.*

*D. Beyer and T. Lemberger: **Software Verification: Testing vs. Model Checking**. Proc. HVC, 2017.*

*D. Beyer, M. Dangl, T. Lemberger, and M. Tautschnig: **Tests from Witnesses: Execution-Based Validation of Verification Results**. Proc. TAP, 2018.*

*D. Beyer and T. Lemberger: T**estCov: Robust Test-Suite Execution and Coverage Measurement**. Proc. ASE, 2019.*

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

*D. Beyer and T. Lemberger: **Five Years Later: Testing vs. Model Checking**. STTT, under review.*

Publications presented here

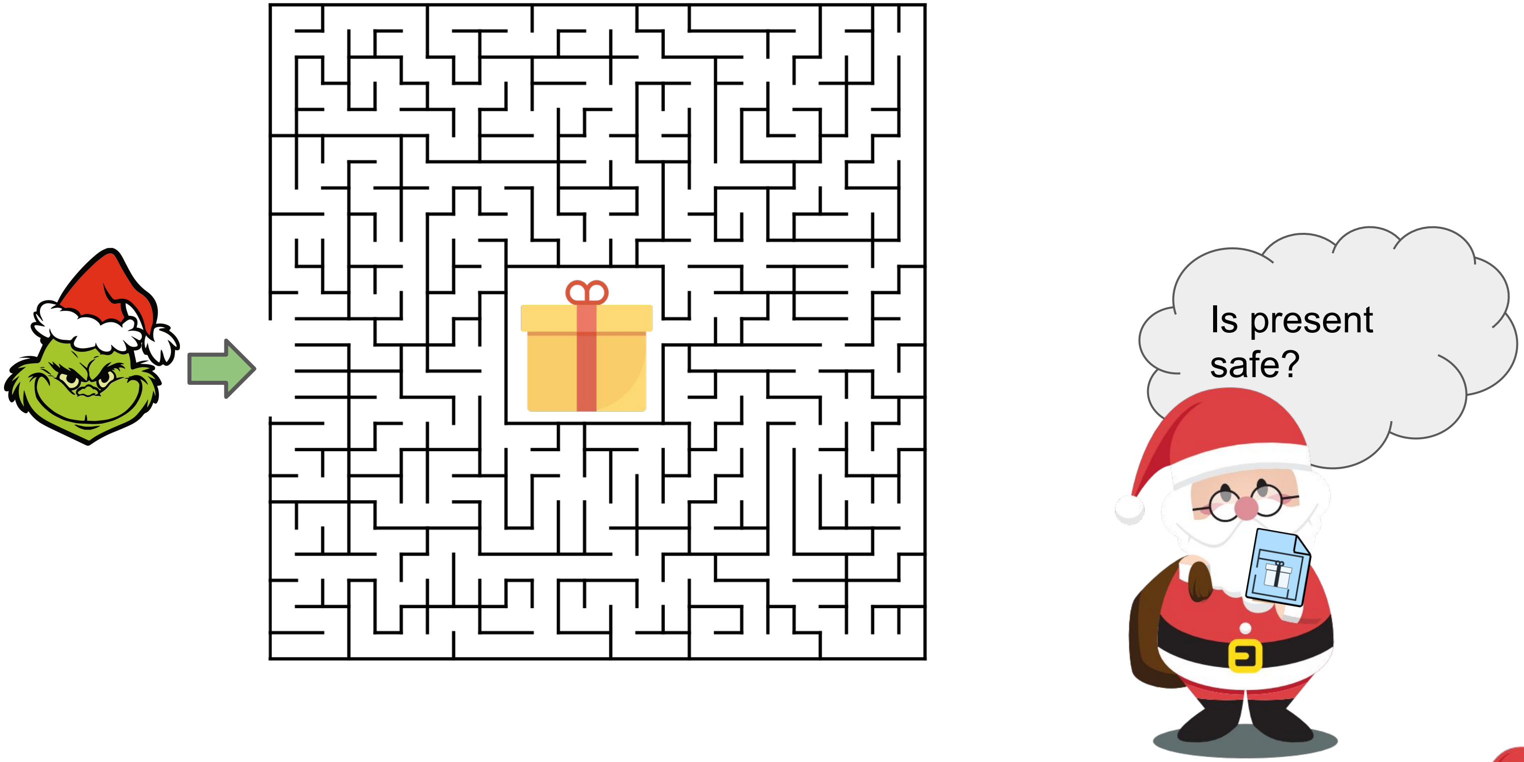
D. Beyer, M.-C. Jakobs, T. Lemberger, and H. Wehrheim: ***Reducer-Based Construction of Conditional Verifiers****.*
Proc. ICSE, 2018.

D. Beyer and T. Lemberger: ***Conditional Testing: Off-the-Shelf Combination of Test-Case Generators****. Proc. ATVA, 2019.*

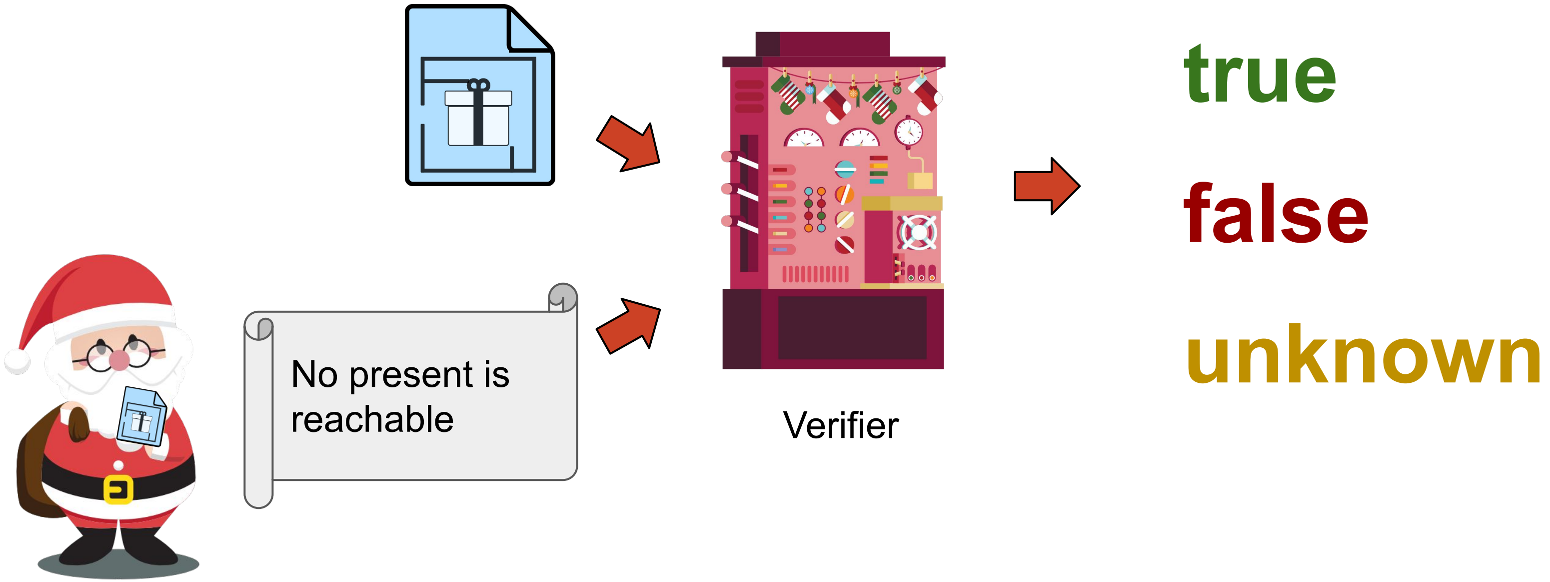
D. Beyer, J. Haltermann, T. Lemberger, and H. Wehrheim: ***Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR****. Proc. ICSE, 2022.*

Context

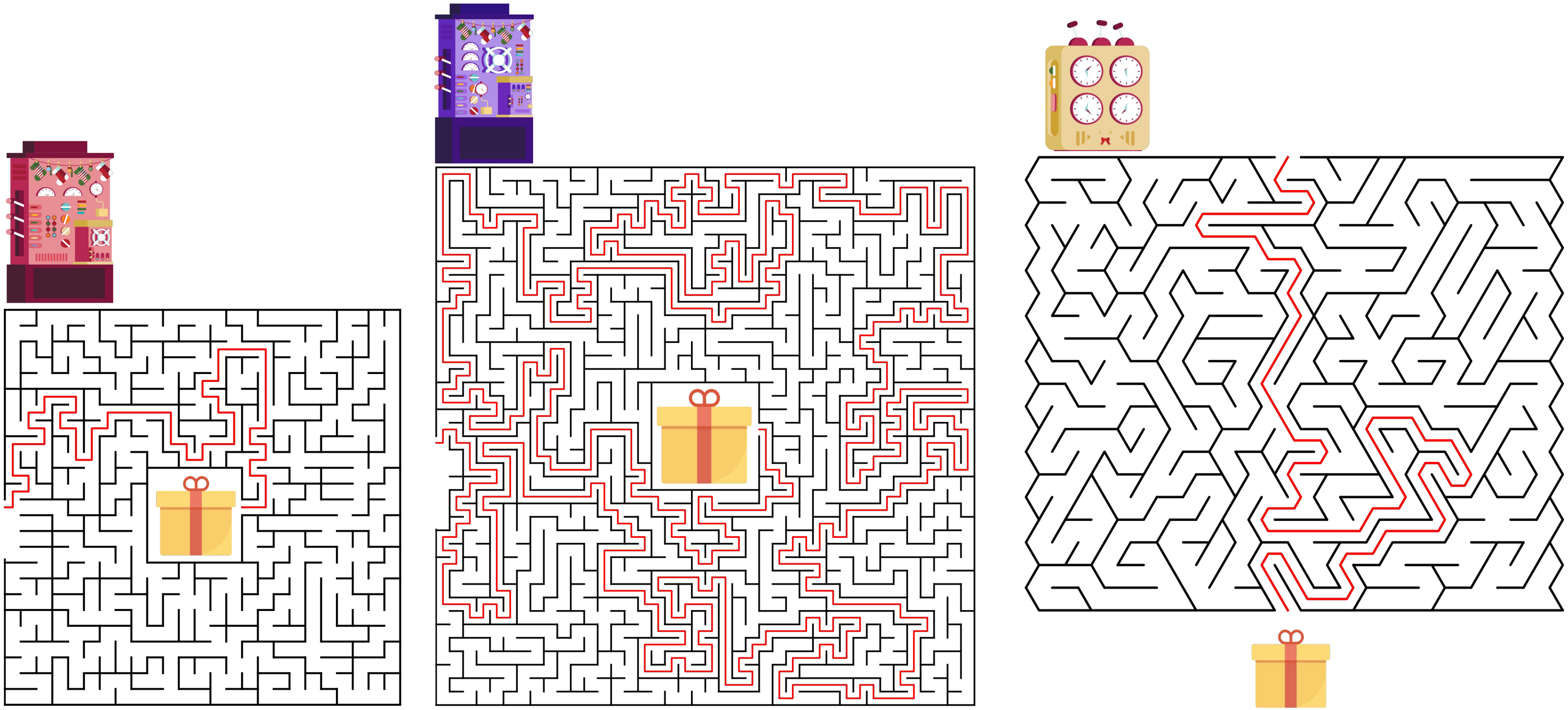
Automated Software Verification



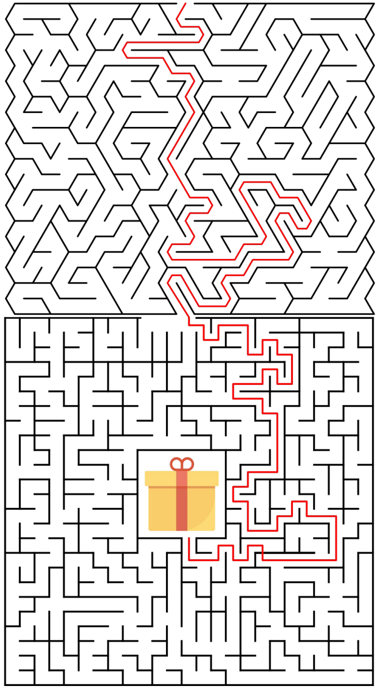
Automated Software Verification

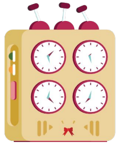


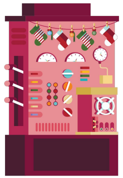
- Verifiers have different strengths and weaknesses



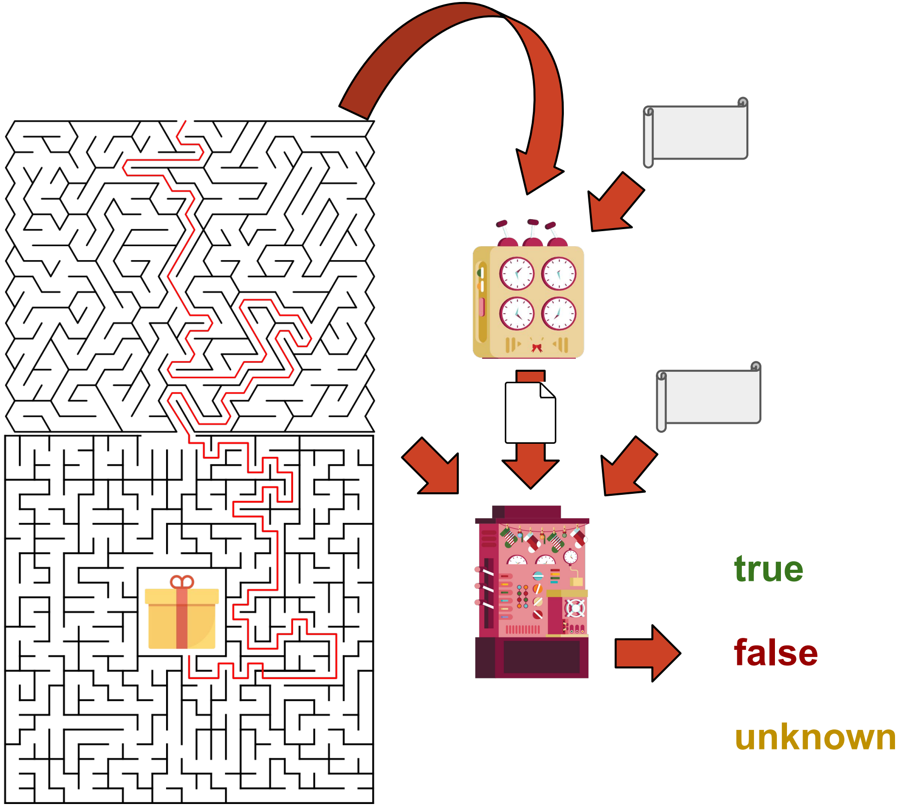
- Verifiers have different strengths and weaknesses







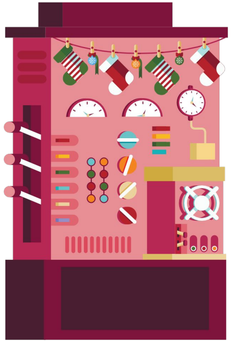
- Verifiers have different strengths and weaknesses
- ***Cooperative Verification*** tries to combine the strengths and mitigate the weaknesses



Automated Software Verification

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

No call to
reach_error()
is reachable



Verifier

true (proof)

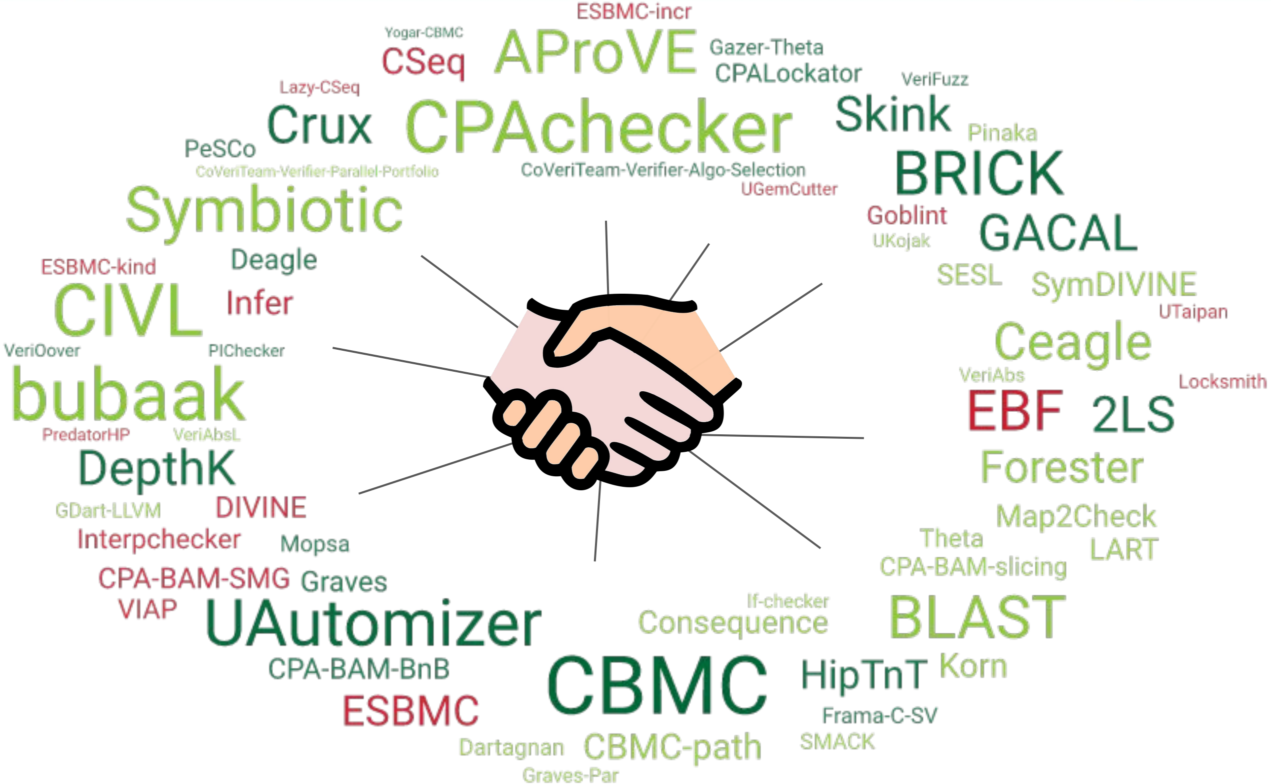
false (alarm)

unknown

ETAPS
EUROPEAN JOINT CONFERENCES ON
THEORY & PRACTICE OF SOFTWARE

TACAS 2023

12th Competition on Software Verification (SV-COMP 2023)

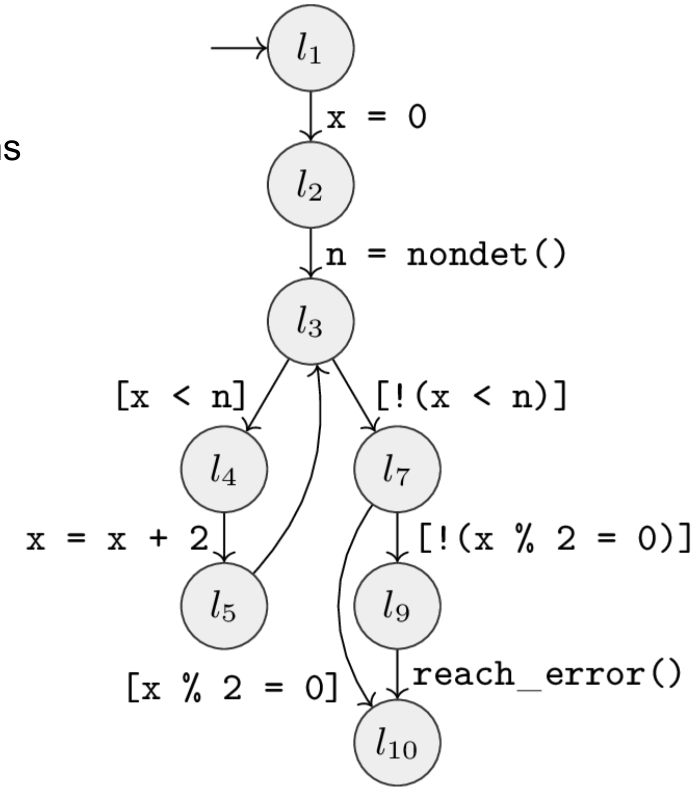


Background

Control-Flow Automaton (CFA)

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

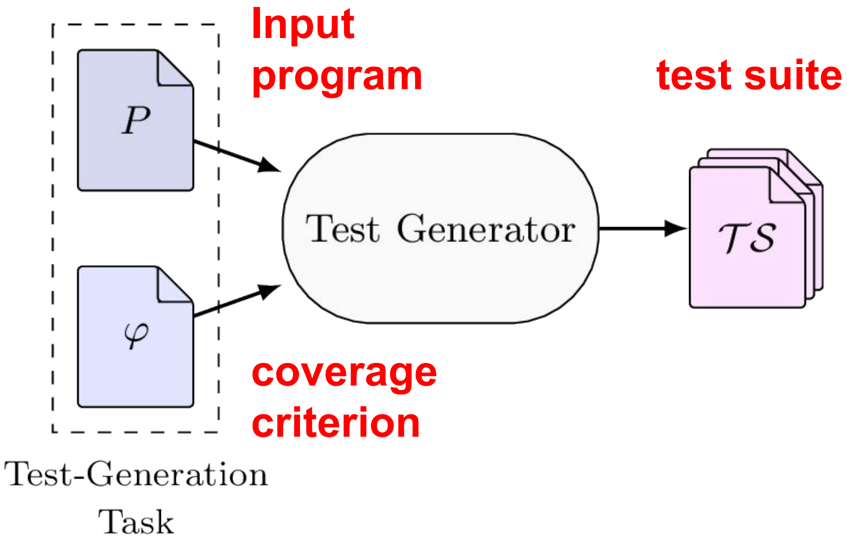
```c
int main(void) {
  unsigned int x = 0;
  unsigned short n = nondet();
  while (x < n) {
    x += 2;
  }
  if (x % 2 == 0) {}
  else
    reach_error();
}
```



Automated Software Verification

- Two approaches:
  - Automated Test Generation
  - Automated Formal Verification

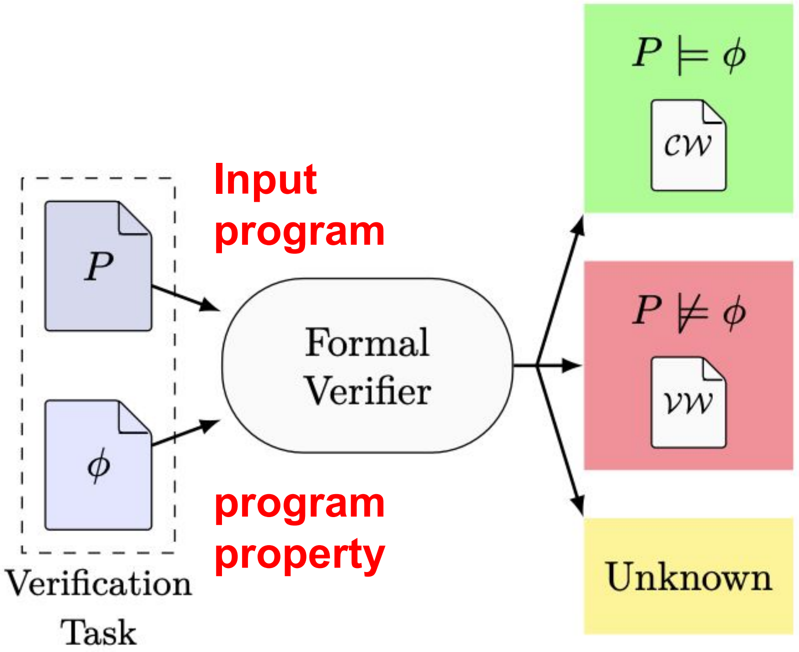
Test Generation



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.

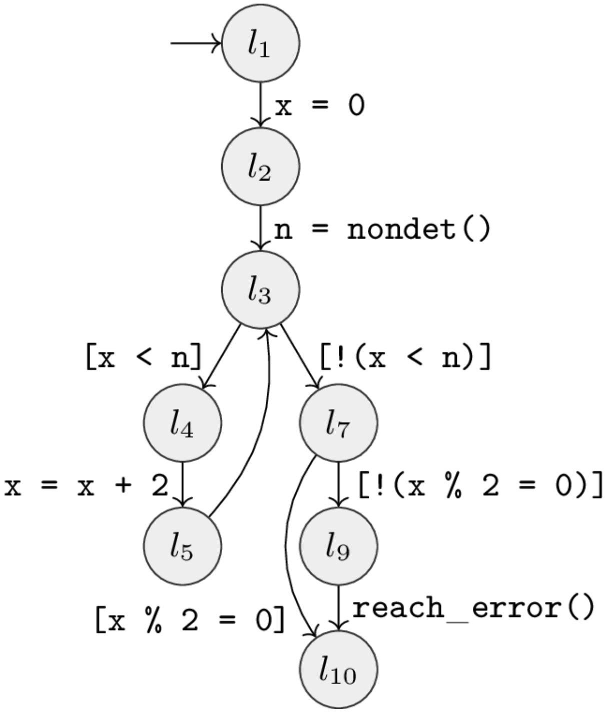
Formal Verification



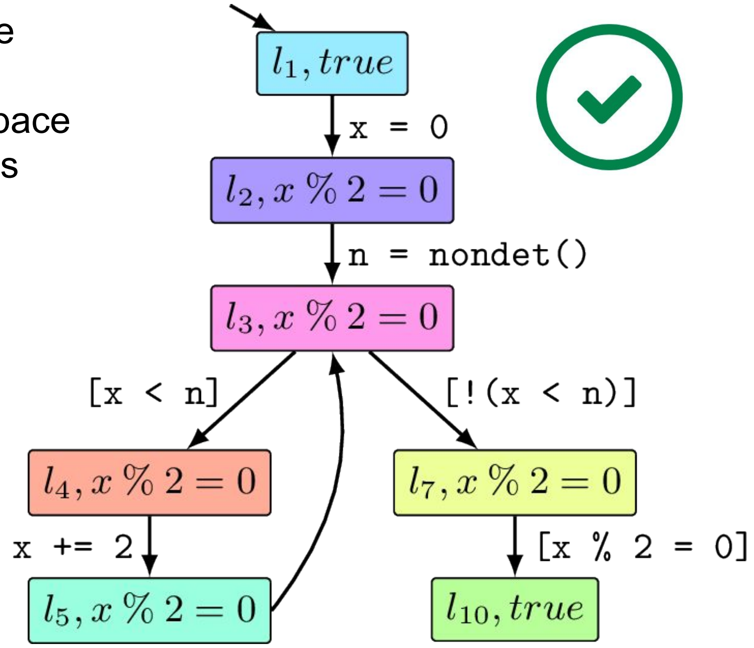
Common technique:

- Compute reachable (abstract) program state space.
- Any reachable state at call to `reach_error()`? → property violation.

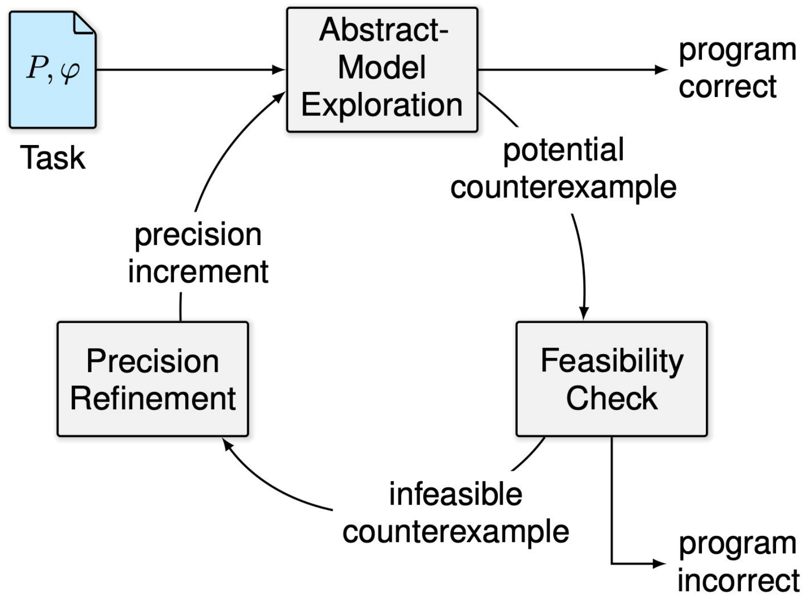
Example: Predicate Abstraction



- Program state space potentially infinite
- Abstract the state space with given predicates
- Here: `x % 2 = 0`



Counterexample-Guided Abstraction Refinement (CEGAR)



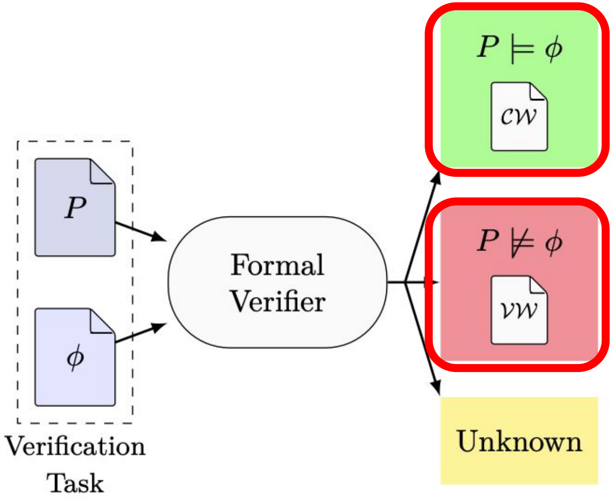
- Derive program abstraction as abstract as possible and as precise as necessary

- 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



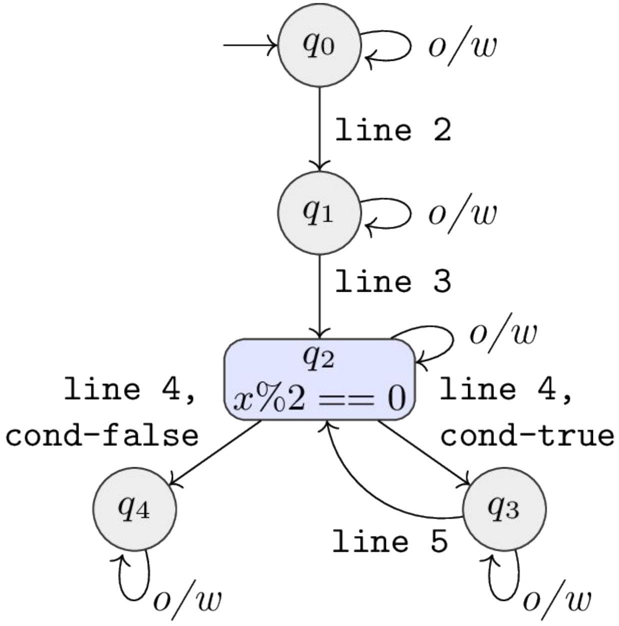
D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, T. Lemberger, and M. Tautschnig: Verification Witnesses. TOSEM, 2022.

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

```
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 }
```

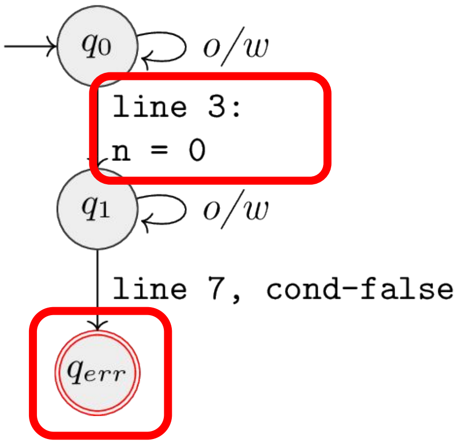


o/w: otherwise

Violation Witness (Path Witness)

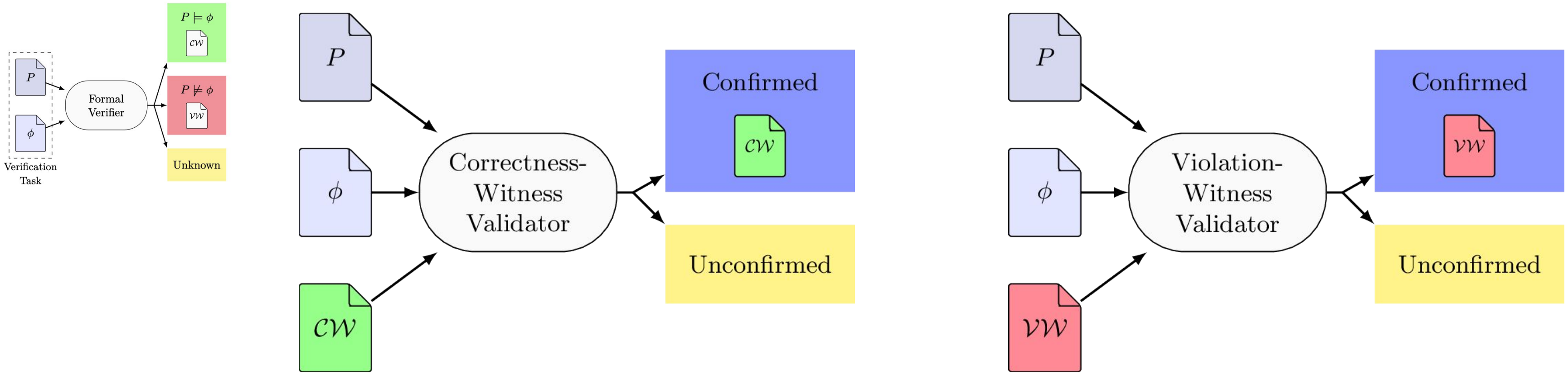
- Nodes: States
- Edges: *source-code guards* and *state-space guards*
- Accepting state: Violation reached

```
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 }
```



o/w: otherwise

Witness Validation



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

Success → Verification result confirmed

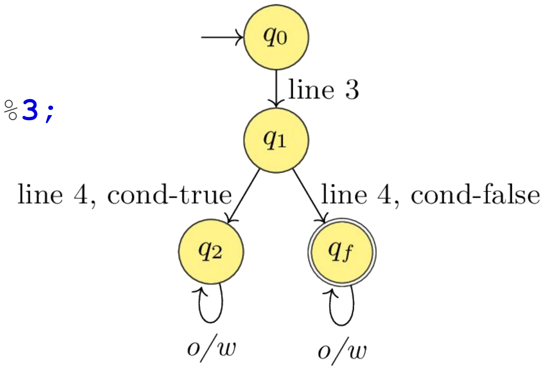
D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, T. Lemberger, and M. Tautschnig: Verification Witnesses. TOSEM, 2022.

Condition Automaton

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

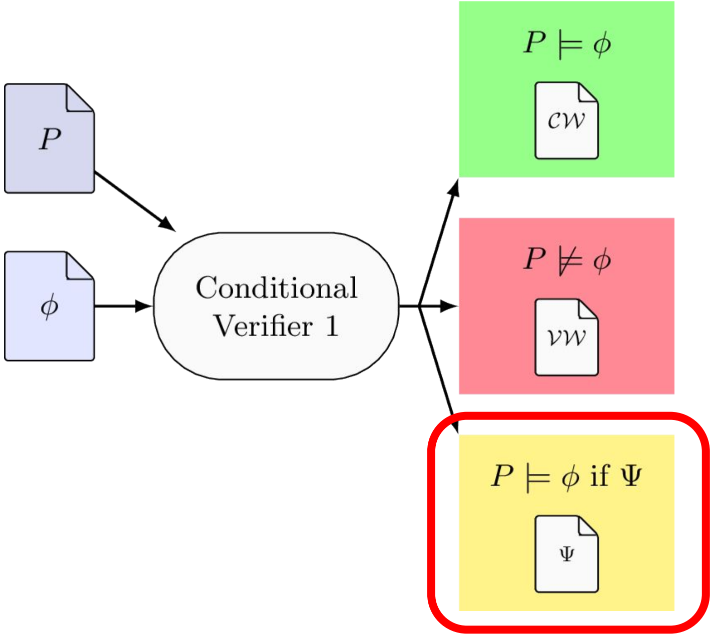
A condition *covers* a program execution if its run leads to an accepting state

```
1  int main() {
2    int out;
3    int val = nondet();
4    if (val >= 0) {
5      out = val%2 * val%3;
6    } else {
7      out = -val;
8    }
9    if (out < 0) {
10     reach_error();
11   }
12 }
```



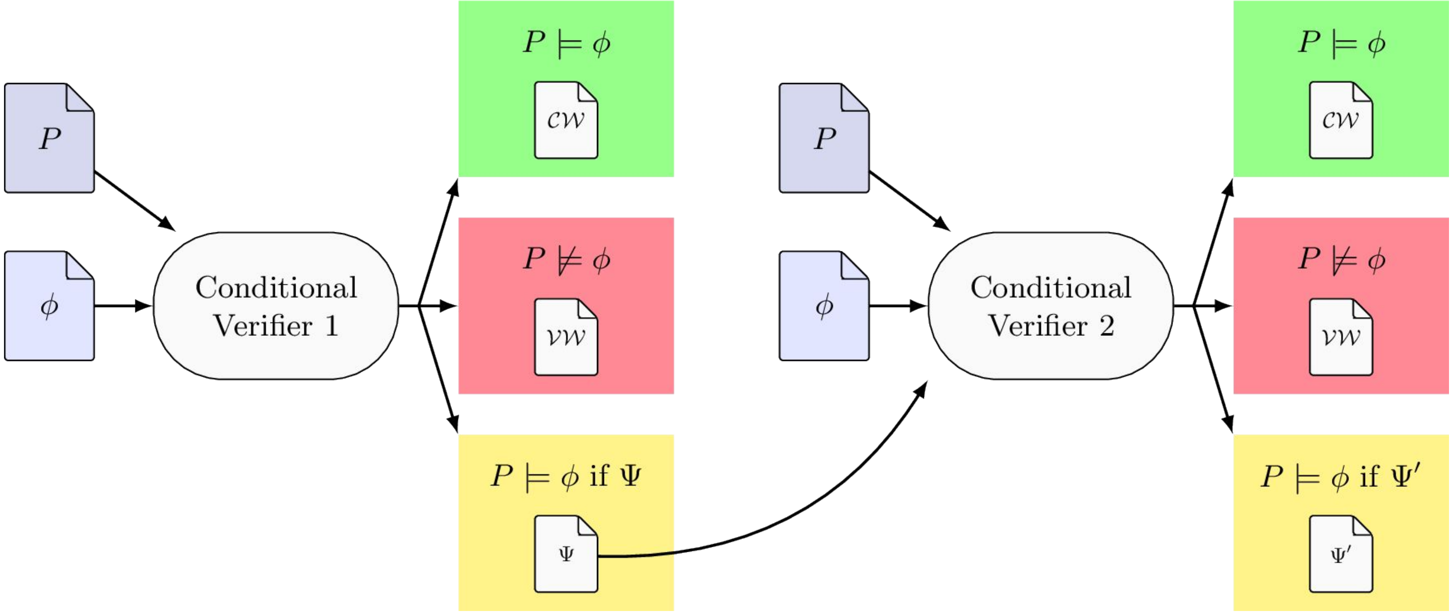
D. Beyer, T. A. Henzinger, M. E. Keremoglu, and P. Wendler:
Conditional Model Checking: A Technique to Pass Information between Verifiers. Proc. FSE, 2012.

Conditional Verification



D. Beyer, T. A. Henzinger, M. E. Keremoglu, and P. Wendler: Conditional Model Checking: A Technique to Pass Information between Verifiers. Proc. FSE, 2012.

Conditional Verification

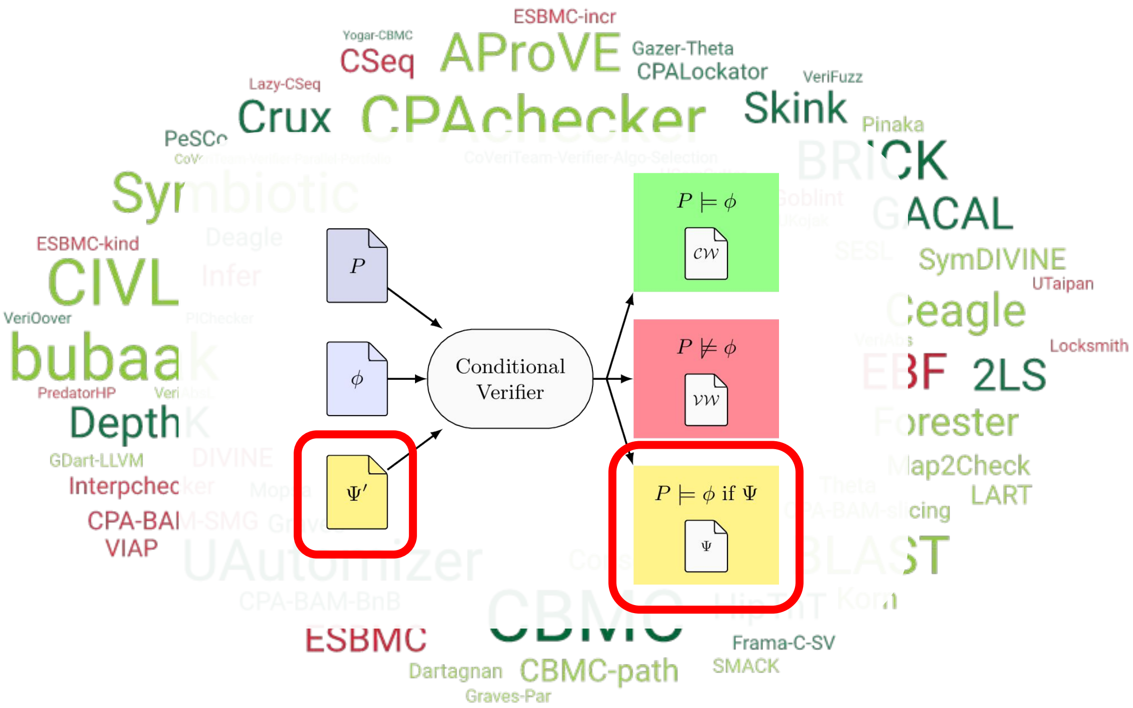


D. Beyer, T. A. Henzinger, M. E. Keremoglu, and P. Wendler:
Conditional Model Checking: A Technique to Pass Information between Verifiers. Proc. FSE, 2012.

Cooperative Software Verification with Condition Automata

D. Beyer, M.-C. Jakobs, T. Lemberger, and H. Wehrheim:
Reducer-Based Construction of Conditional Verifiers. Proc. ICSE, 2018.

Reducer-Based Construction of Conditional Verifiers

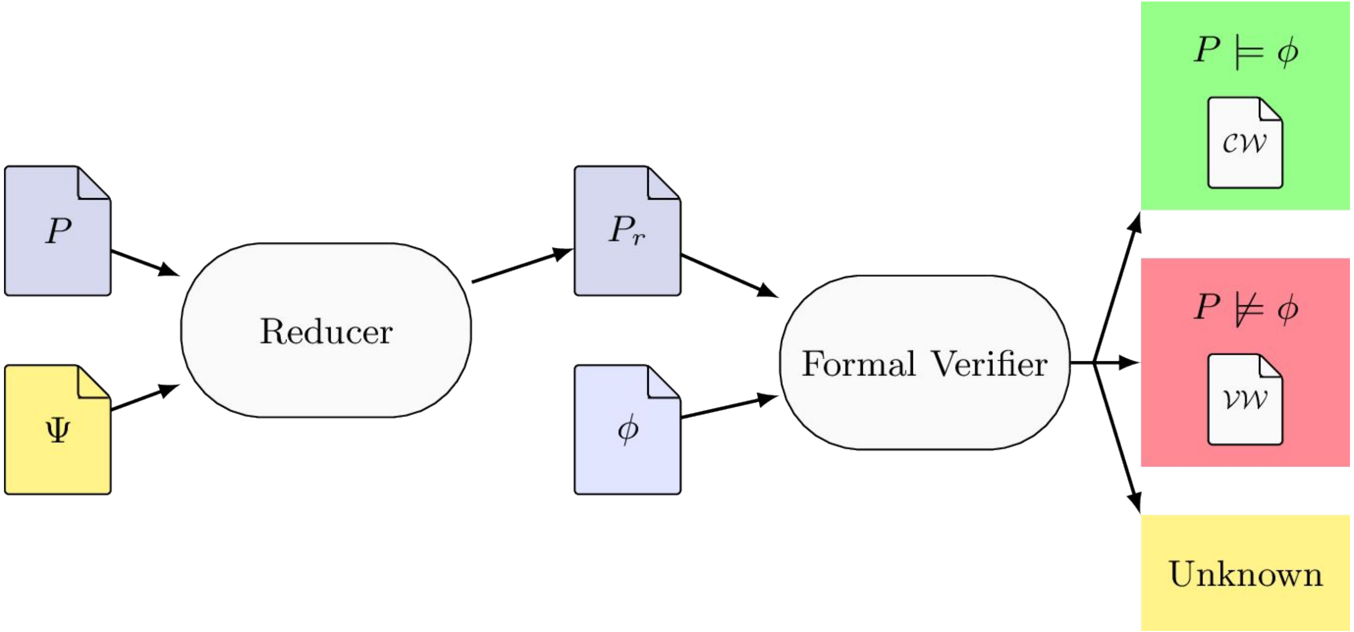


- Conditional Verification is great!
- But only one conditional verifier: CPAchecker.

Create providers of conditions?

Create consumers of conditions?

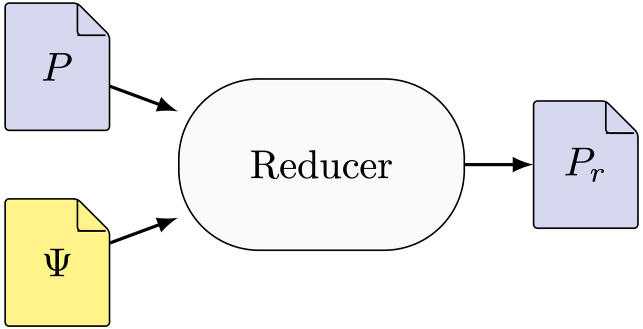
Reducer-Based Construction of Conditional Verifiers



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.

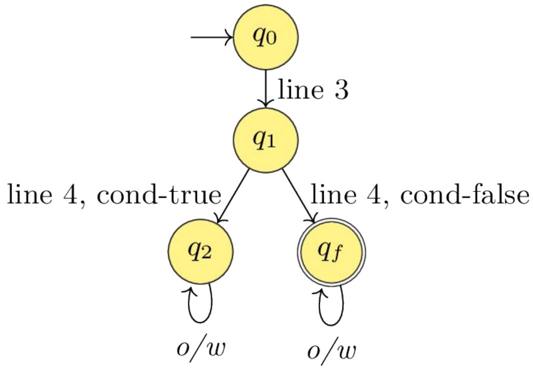


Reducers:

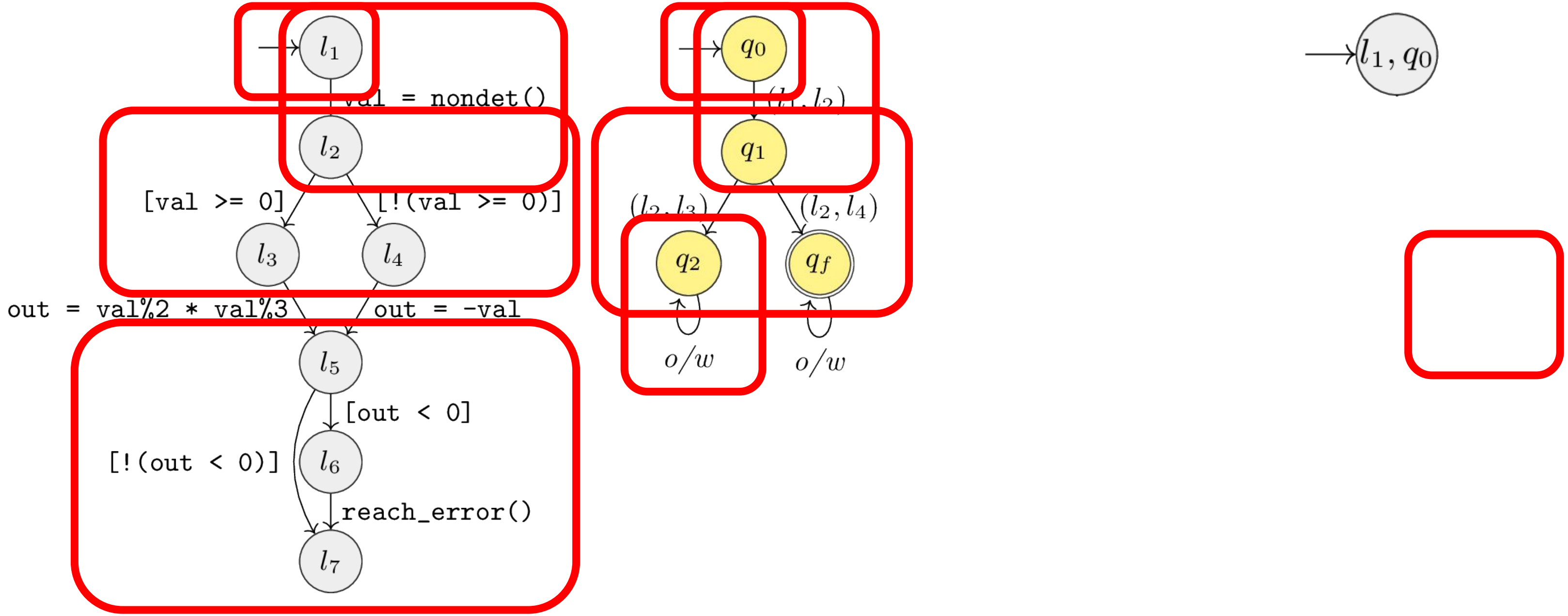
- Identity
- Parallel Composition

Reducer: Parallel Composition

```
 1 int main() {
 2   int out;
 3   int val = nondet();
 4   if (val >= 0) {
 5     out = val%2 * val%3;
 6   } else {
 7     out = -val;
 8   }
 9   if (out < 0) {
10     reach_error();
11   }
12 }
```



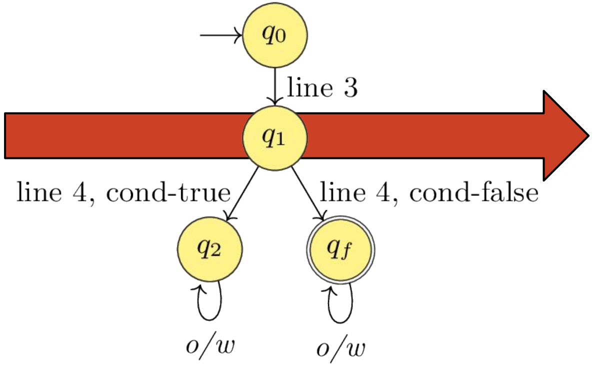
Reducer: Parallel Composition



Beyer, Jakobs, Lemberger, Wehrheim: Reducer-Based Construction of Conditional Verifiers, ICSE 2018.

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();
  }
}
```

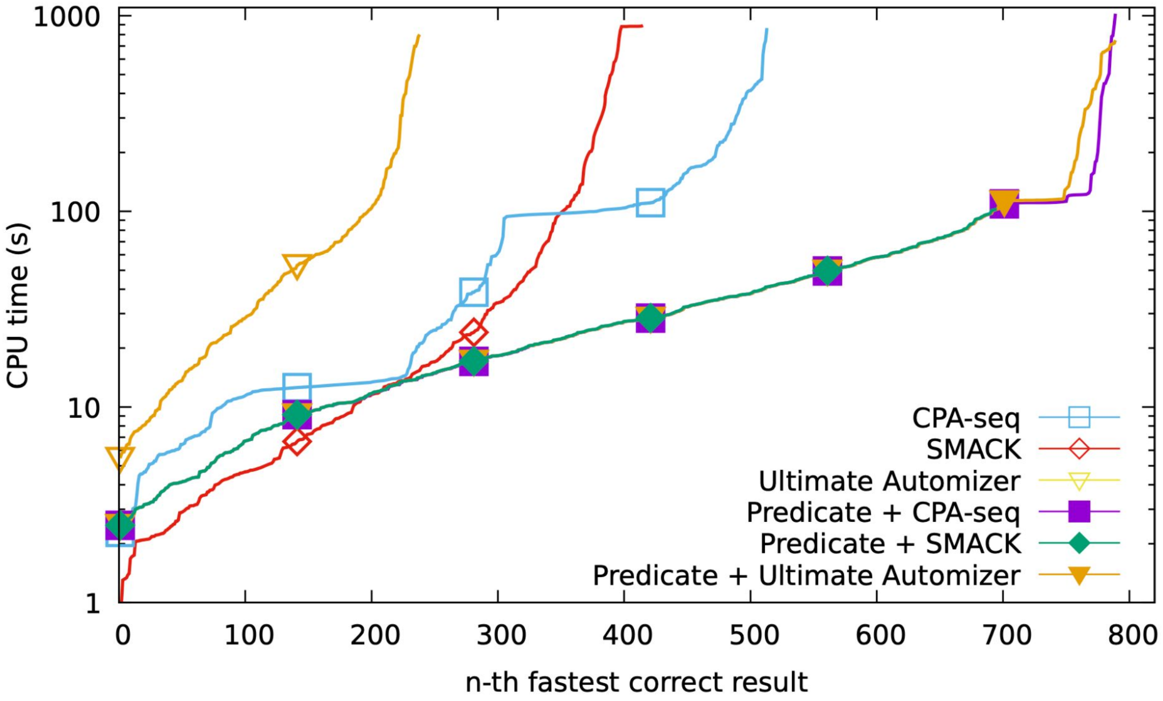



```c
int main() {
  int out;
  int val = nondet_int();
  if (val >= 0) {
    out = val%2 * val%3;
    if (out < 0) {
      reach_error();
    }
  } else { }
}
```

Evaluation

- Reducers *Identity* and *Parallel Composition*, implemented in `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:
  - 15 GB memory
  - 100 s predicate analysis + 900 s CPA-seq/Smack/Ultimate Automizer
- Reproduction package: 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.

Proc. ATVA, 2019.

Cooperation between Test Generators

```
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

Beyer, Lemberger: *Conditional Testing: Off-the-Shelf Combination of Test-Case Generators.* ATVA 2019.

Cooperation between Test Generators

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

Random Tester

Beyer, Lemberger: Conditional Testing: Off-the-Shelf Combination of Test-Case Generators. ATVA 2019.

Cooperation between Test Generators

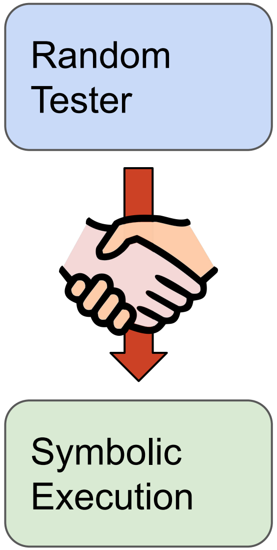
```c
int main() {
  int i = nondet();
  if (i != 1017) {
    while (i > 1017) {
      // branch 1.1
      i--;
    }
    // branch 1.2
    // .. snip ..
  } else {
    // branch 2
    // .. snip ..
  }
}
```

Symbolic Execution

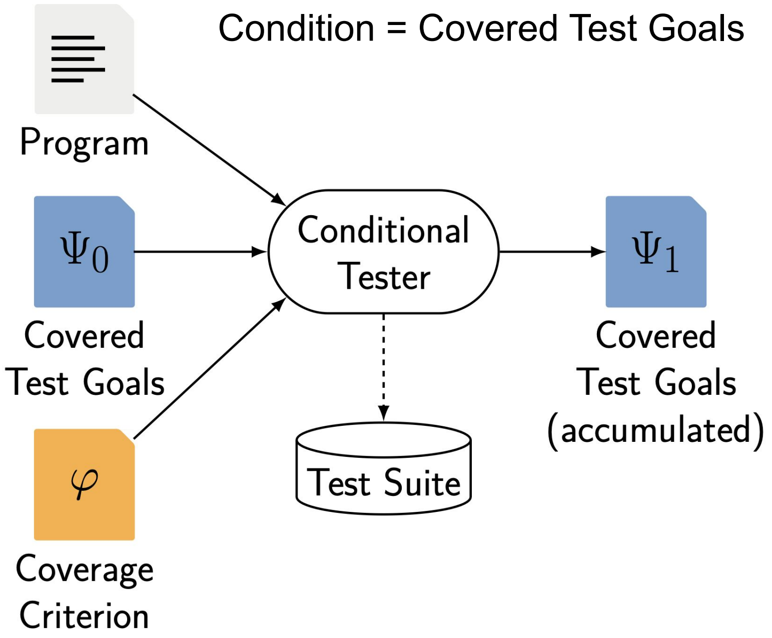
Beyer, Lemberger: *Conditional Testing: Off-the-Shelf Combination of Test-Case Generators.* ATVA 2019.

Cooperation between Test Generators

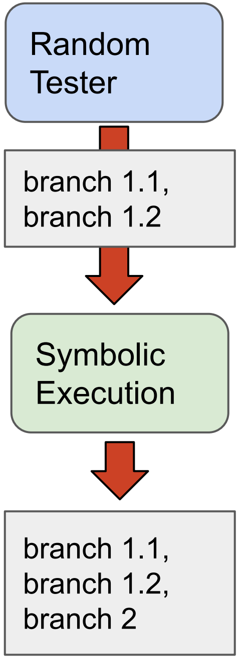
```c
int main() {
  int i = nondet();
  if (i != 1017) {
    while (i > 1017) {
      // branch 1.1
      i--;
    }
    // branch 1.2
    // .. snip ..
  } else {
    // branch 2
    // .. snip ..
  }
}
```



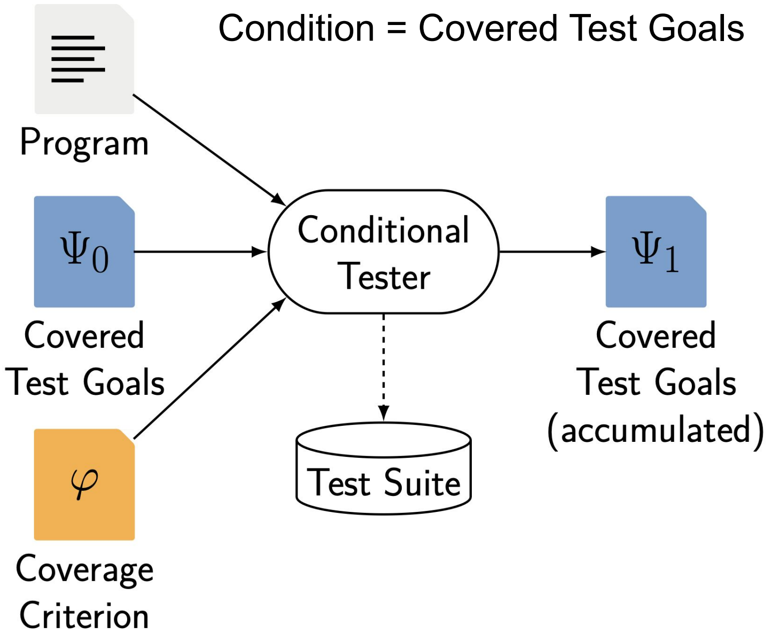
Conditional Testing




```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

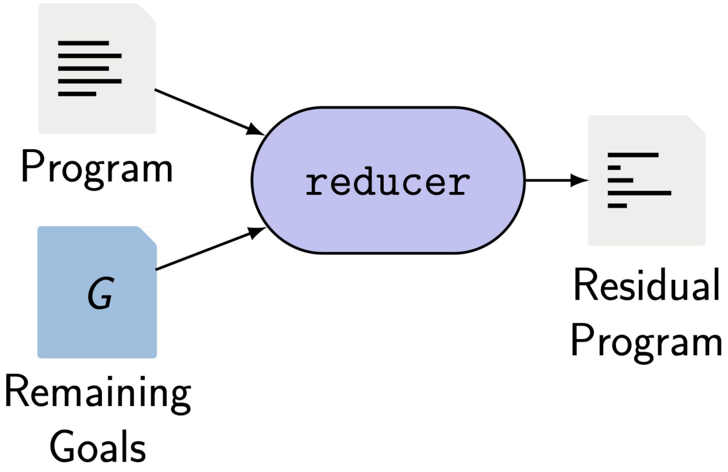


Problem: We just came up with this!

→ 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:

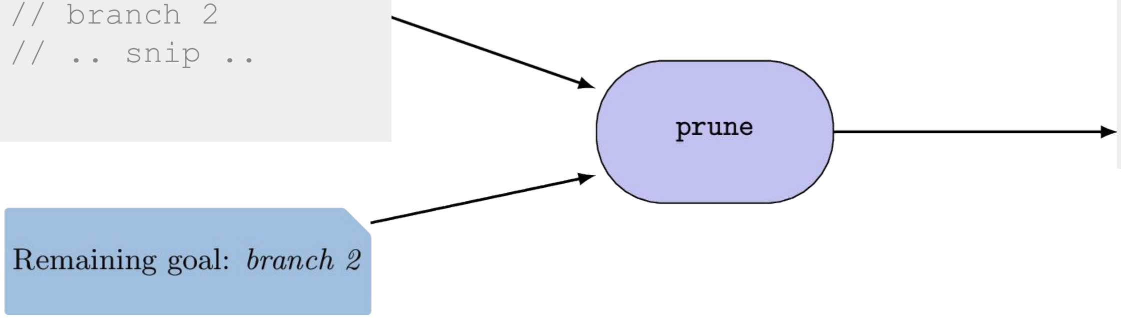
- Identity
- Pruning

Beyer, Lemberger: *Conditional Testing: Off-the-Shelf Combination of Test-Case Generators.* ATVA 2019.

Pruning Reducer

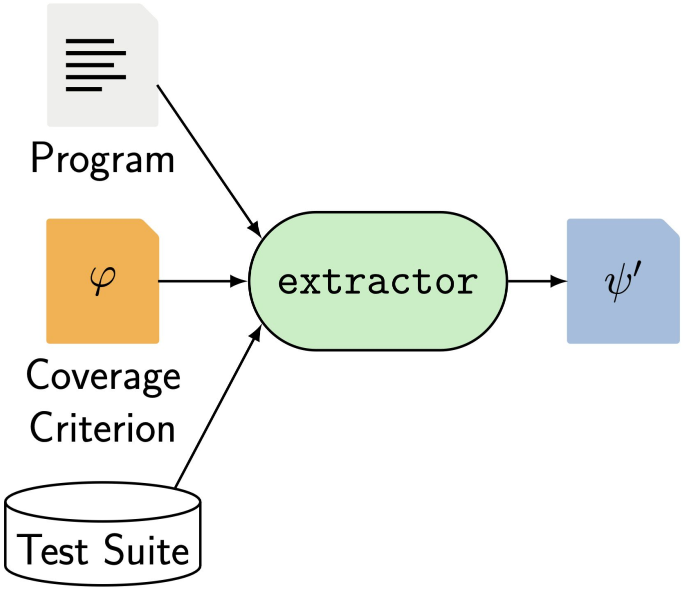
```c
int i = nondet();
if (i != 1017) {
  while (i > 1017) {
    // branch 1.1
    i--;
  }
  // branch 1.2
  // .. snip ..
} else {
  // branch 2
  // .. snip ..
}
```

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




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

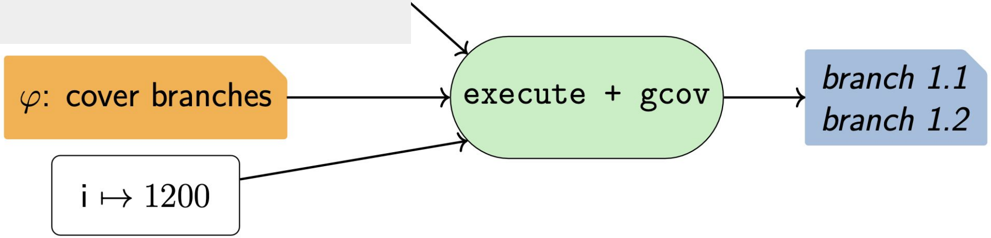
Test-Goal Extractor



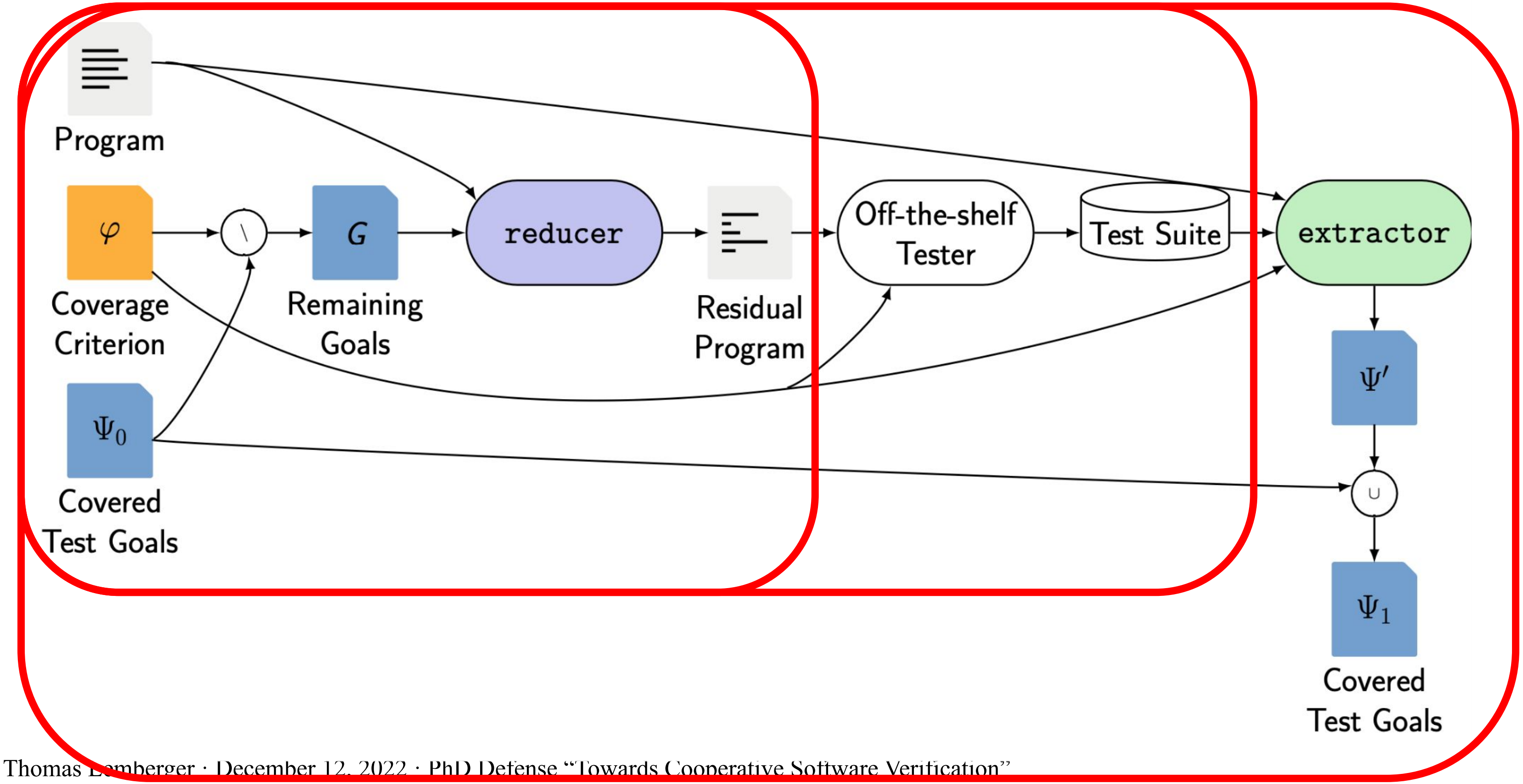
gcov-based Test-Goal Extractor

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

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



Off-the-shelf Tester to Conditional Tester

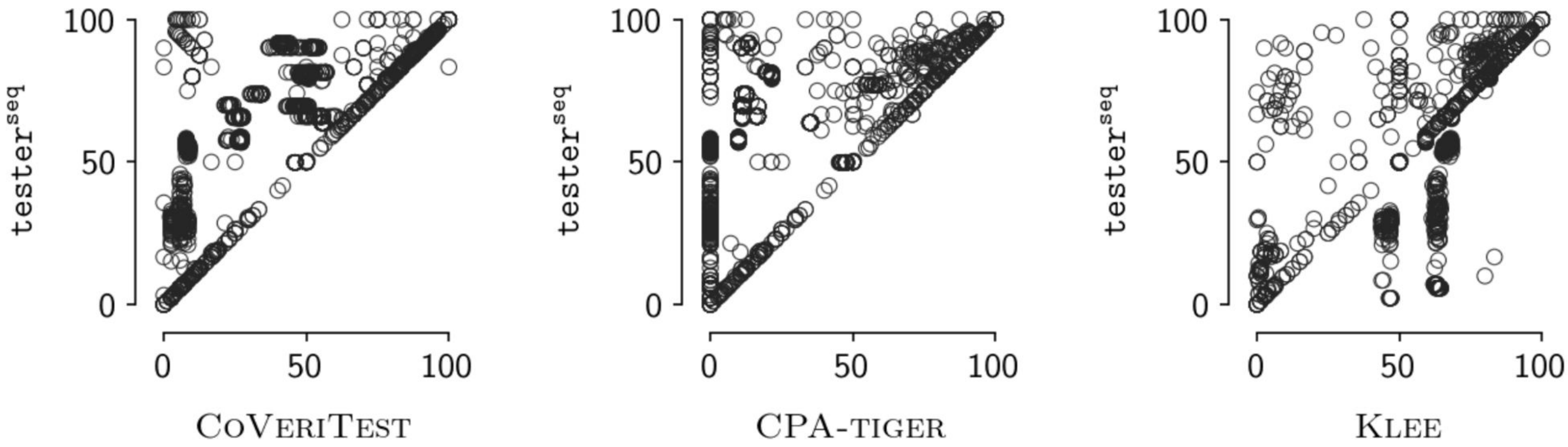


Evaluation

- Components implemented as `CondTest`
  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

Evaluation



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

Evaluation

`CPA-Tiger + CoVeriTest + Klee`, 300 s each

`id`: no info. exchange `prune`: info. exchange

Task	branch coverage		
	id	$\rightarrow$	**prune**
mod3.c.v+sep-reducer	75.0	+5.00	80.0
Problem07_label35	52.0	+2.00	54.0
Problem07_label37	54.2	+1.97	56.2
Problem04_label35	79.5	+1.79	81.3
Problem06_label02	57.0	+1.70	58.7
Problem06_label27	57.5	+1.09	58.6
Problem04_label02	80.2	+1.06	81.3
Problem06_label18	57.5	+1.05	58.6
Problem04_label16	79.1	+1.01	80.1
Problem04_label34	80.2	+0.99	81.2

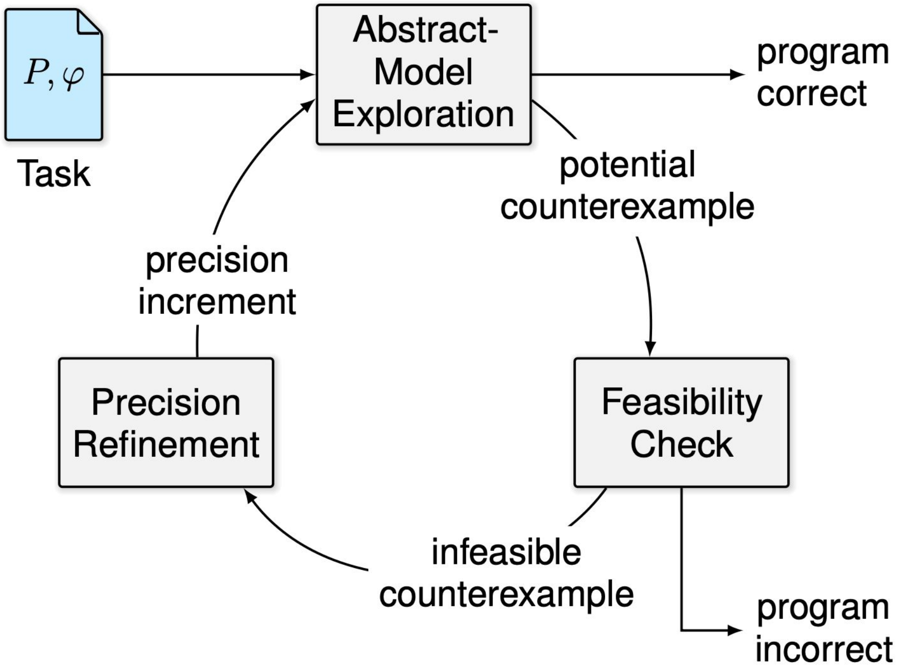
Insights

- Effectiveness increases through combinations
- Encoding in program allows to apply testers without explicit condition support

Decomposing Verification Techniques

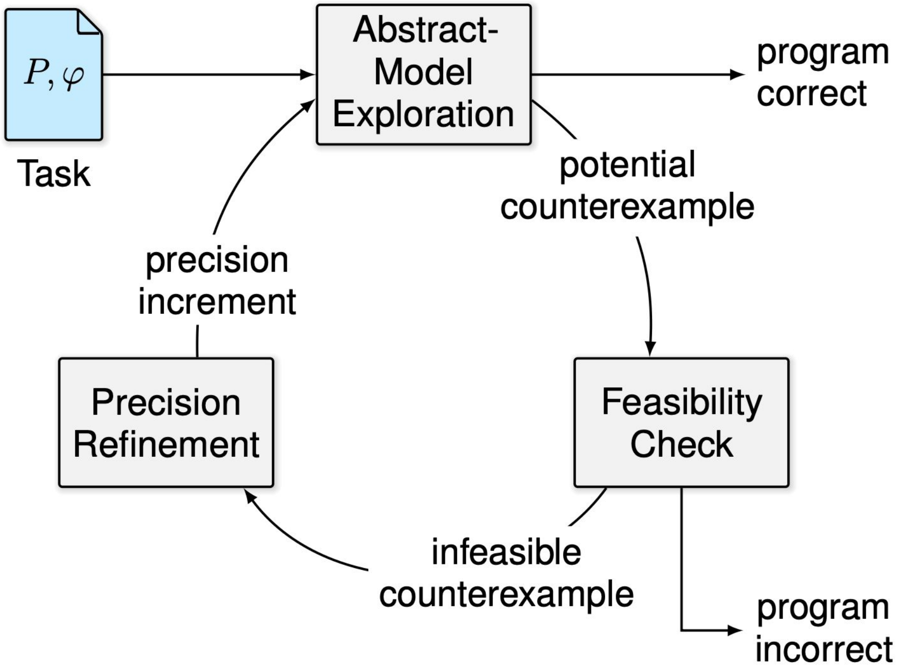
D. Beyer, J. Haltermann, T. Lemberger, and H. Wehrheim:
Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR.
Proc. ICSE, 2022.

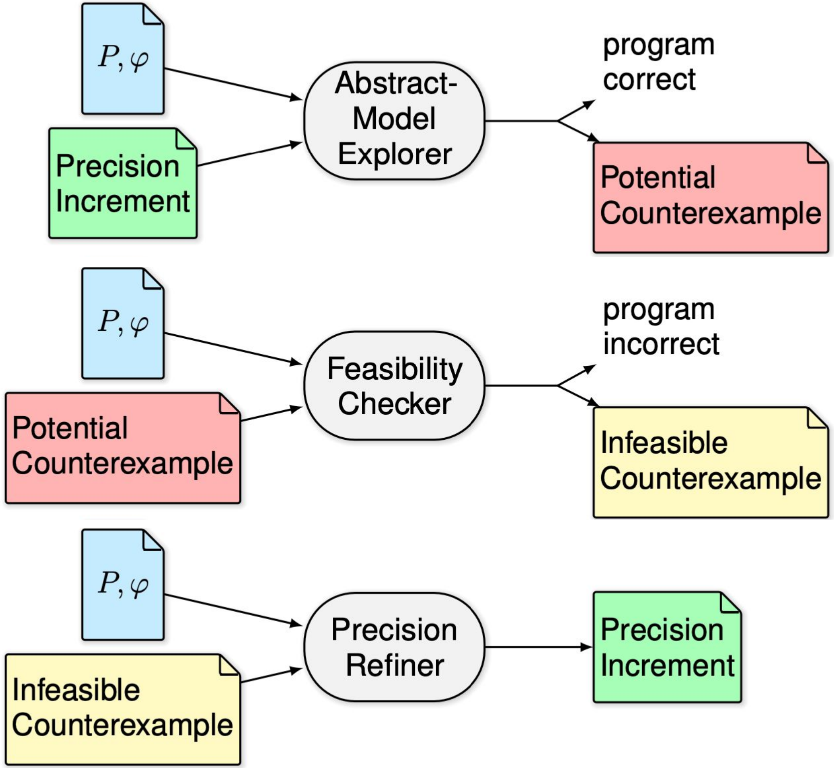
Motivation: CEGAR



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

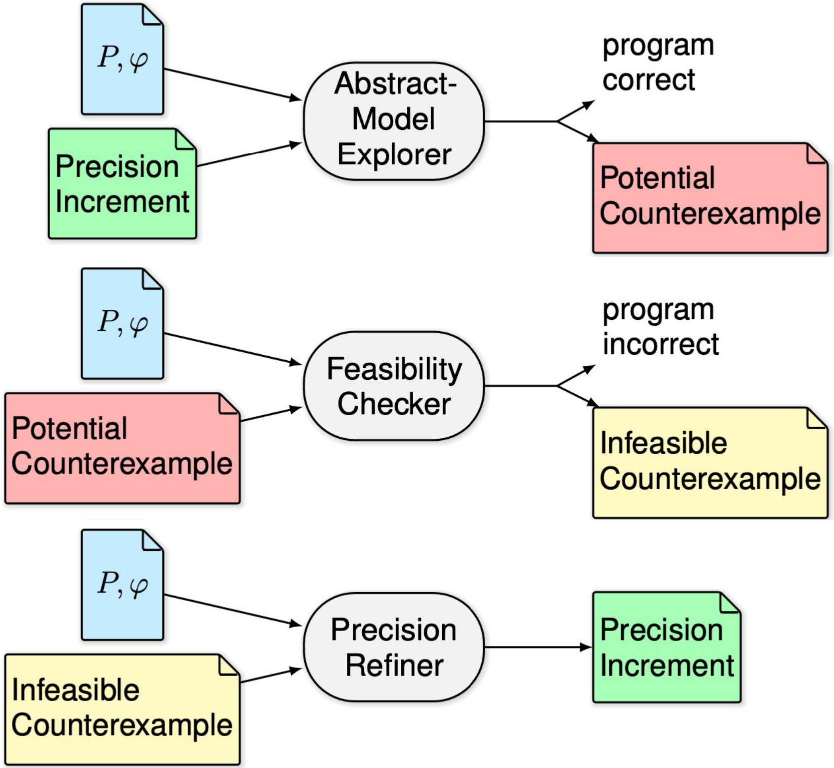
Decomposing CEGAR





Beyer, Haltermann, Lemberger, Wehrheim: *Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR*, ICSE 2022.

Decomposing CEGAR



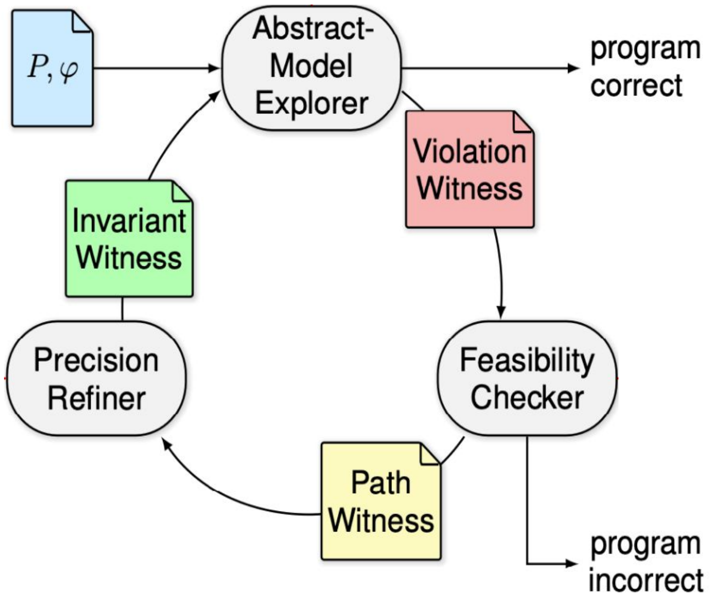
Exchange formats from SV-COMP
→ wide tool support

Abstract description of counterexample

Abstract description of rejected counterexample (“violation” witness)

Description of candidate invariants (“correctness” witness)

Component-based CEGAR (C-CEGAR)



Evaluation

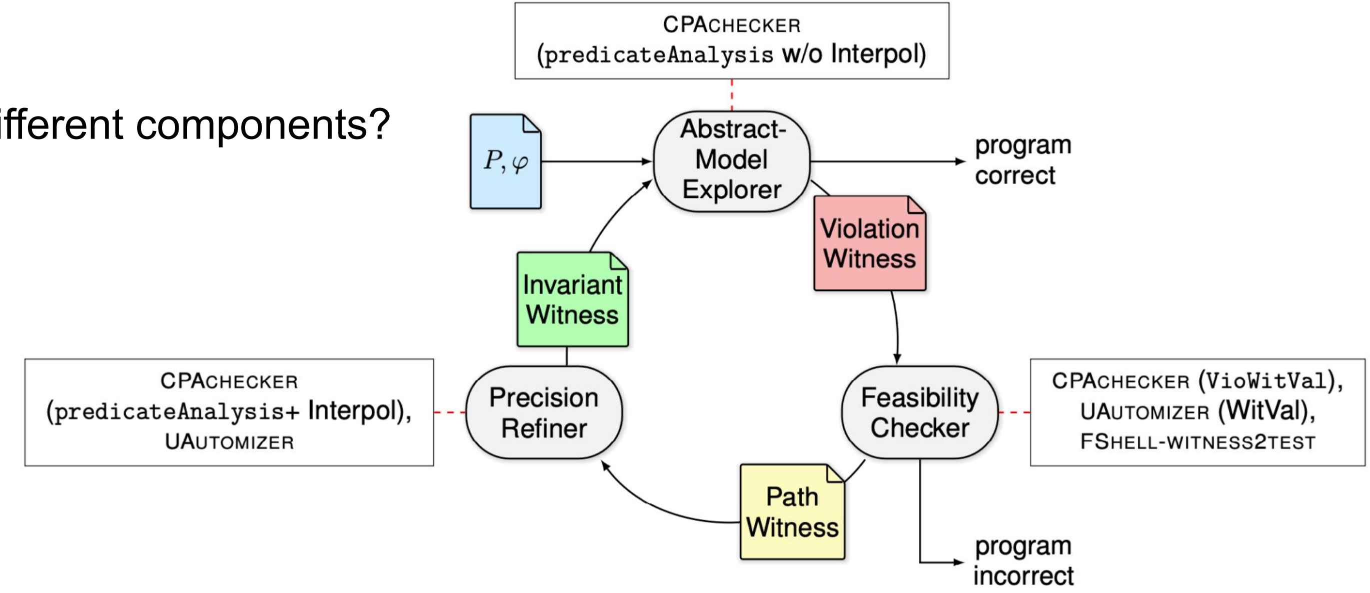
- Implementation in CoVeriTeam
  https://gitlab.com/sosy-lab/software/coveriteam/-/tree/main/examples/Component-based_CEGAR
- Tools:
  - `CPAchecker` with improvements
  - `Ultimate Automizer` SV-COMP 2021
  - `FShell-witness2test` SV-COMP 2021
- Tasks: 8347 ReachSafety tasks @ SV-COMP 2021
- Limits: 900 s CPU time, 15 GB memory

- Reproduction package: 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

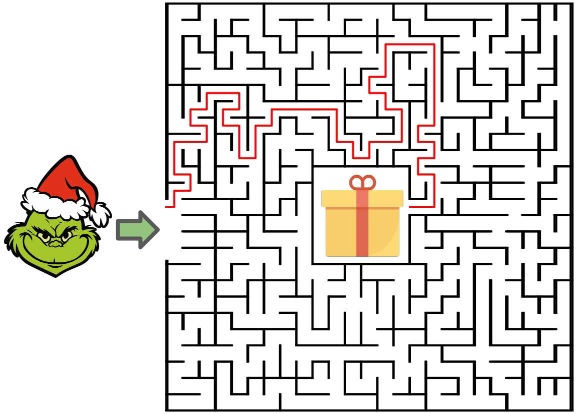
RQ 3.1: C-PredWit + different feasibility checker (with precision refiner CPAchecker)

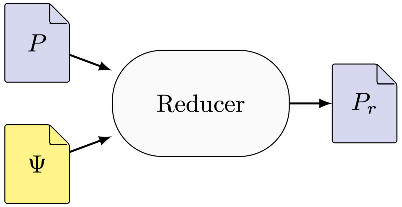
		correct			
	overall	**proof**	unique	**alarm**	unique
CPAchecker	2 854	2 110	494	744	441
FShell-witness2test	1 223	1 126	0	97	64
UAutomizer	1 941	1 614	4	327	29

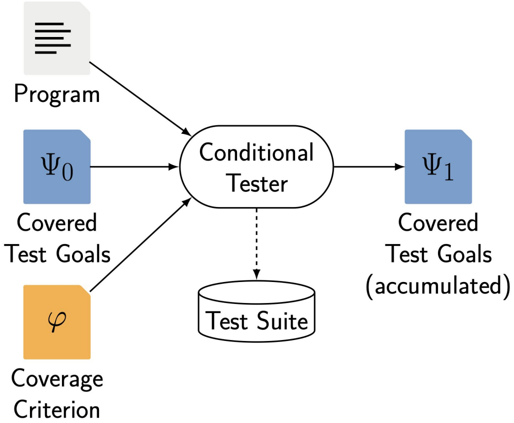
RQ 3.2: C-PredWit + different precision refiner (with feasibility checker CPAchecker)

		correct			
	overall	**proof**	unique	**alarm**	unique
CPAchecker	2 854	2 110	709	744	436
UAutomizer	1 739	1 430	29	309	1

Conclusion

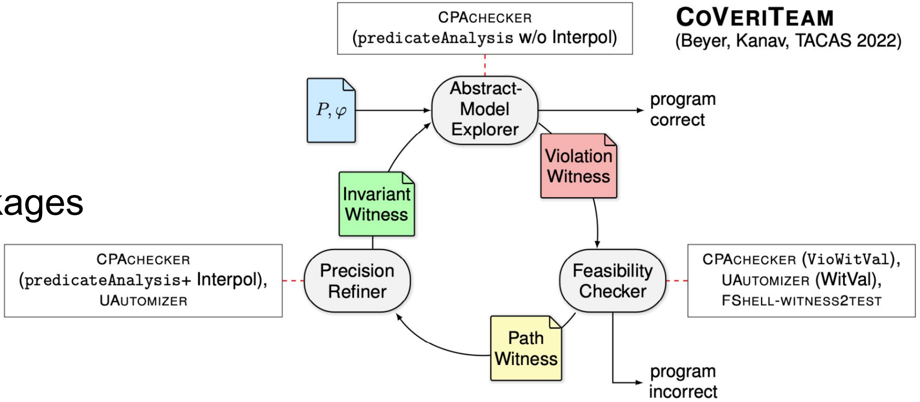






- Improved effectiveness of verification
- Improved opportunities for cooperation

- Backed by rigorous
  experimental evaluation and reproduction packages



Backup Slides

Backup Reducers: Algorithm

Algorithm 1 REDUCER

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

CA $A = (Q, \Sigma, \delta, q_0, F)$ s.t. $q_r \notin Q$ ▷ *condition automaton*

Output: CFA $C_r = (L_r, \ell_{0,r}, G_r)$ ▷ *residual program*

$L_r := \{(\ell_0, q_0)\}$; $\ell_{0,r} := (\ell_0, q_0)$; $G_r := \emptyset$;
waitlist $:= L_r$;
while waitlist $\neq \emptyset$ **do**
  choose $(\ell_1, q_1) \in$ waitlist; remove (ℓ_1, q_1) from waitlist;
  for each $g = (\ell_1, op, \ell_2) \in G$ **do**
    if $q_1 \in Q \wedge \exists (q_1, (G_1, true), q_2) \in \delta$ s.t. $g \in G_1$ **then**
      for each $(q_1, (G_1, true), q_2) \in \delta$ s.t. $g \in G_1$ **do**
        if $q_2 \notin F \wedge (\ell_2, q_2) \notin L_r$ **then**
          waitlist $:=$ waitlist $\cup \{(\ell_2, q_2)\}$;
        $L_r := L_r \cup \{(\ell_2, q_2)\}$;
        $G_r := G_r \cup \Big\{\big((\ell_1, q_1), op, (\ell_2, q_2)\big)\Big\}$;
    else
      if $(\ell_2, q_r) \notin L_r$ **then**
        waitlist $:=$ waitlist $\cup \{(\ell_2, q_r)\}$;
      $L_r := L_r \cup \{(\ell_2, q_r)\}$;
      $G_r := G_r \cup \Big\{\big((\ell_1, q_1), op, (\ell_2, q_r)\big)\Big\}$;
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

	CPASEQ	SMACK	UAUTO	Predicate +		
				CPASEQ	SMACK	UAUTO
Correct	513	415	238	789	695	789
Correct proof	265	76	170	387	296	386
Correct alarm	248	339	68	402	399	403
Incorrect	0	0	7	0	0	4
Incorrect proof	0	0	4	0	0	0
Incorrect alarm	0	0	3	0	0	4
Unknown	307	405	575	31	125	27
Total	820	820	820	820	820	820

Backup Reducers: Evaluation

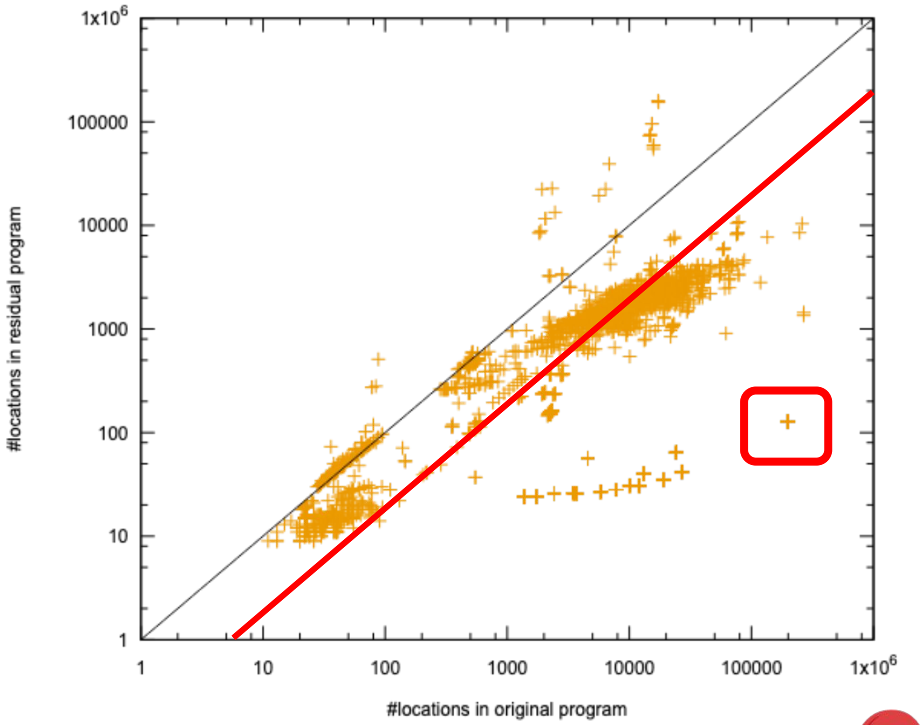
Task	R	CPAseq			Smack			UAutomizer			+CPAseq			+Smack			+UAutomizer		
		S	t (s)	M (GB)	S	t (s)	M (GB)	S	t (s)	M (GB)	S	t (s)	M (GB)	S	t (s)	M (GB)	S	t (s)	M (GB)
lin-4.2 vlsi_ir	T	✗	910	7.9	✗	890	0.97	✗	900	13	✓	490	10	✗	130	0.67	✗	150	0.77
lin-3.14 vsp1	T	✗	920	6.9	✗	890	0.70	✗	910	14	✗	550	1.5	✗	610	1.5	✓	640	1.5
lin-3.14 vxge	T	✗	930	11	✗	190	14	✗	19	0.51	✗	760	1.4	✗	630	1.5	✓	650	1.5
lin-4.2 w83781d	T	✗	910	6.7	✗	900	3.7	✗	910	14	✗	690	1.5	✗	660	1.4	✓	660	1.5
lin-4.2 zd1211rw	T	✗	930	6.3	✗	890	0.96	✗	140	11	✗	720	1.5	✗	670	1.5	✓	660	1.5
lin-3.14 vmxnet3	T	✗	930	6.9	✗	890	1.2	✗	900	10	✗	540	1.5	✗	640	1.4	✓	670	1.4
lin-3.14 skge	T	✗	950	7.3	✗	940	3.6	✗	410	15	✗	650	1.5	✗	600	1.5	✓	670	1.5
lin-3.16 ath5k	T	✗	950	5.9	✗	950	4.7	✗	900	13	✗	710	1.5	✗	730	1.5	✓	710	1.5
lin-3.14 ipw2200	T	✗	950	7.6	✗	950	6.6	✗	15	0.39	✗	700	1.5	✗	730	1.5	✓	720	1.5
lin-3.14 bttv	T	✗	950	5.8	✗	910	5.0	✗	20	0.51	✗	720	1.5	✗	770	1.4	✓	750	1.5
lin-4.2 cciss	T	✗	920	7.1	✗	330	12	✗	900	4.7	✓	790	10	✗	120	0.77	✗	180	5.3
floodmax.4	T	✗	910	3.0	✗	880	0.53	✗	910	13	✓	900	4.3	✗	110	0.42	✗	1 100	7.9
sep20	T	✗	900	3.2	✗	880	0.10	✗	910	13	✓	1 000	2.6	✗	110	0.27	✗	150	0.99

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

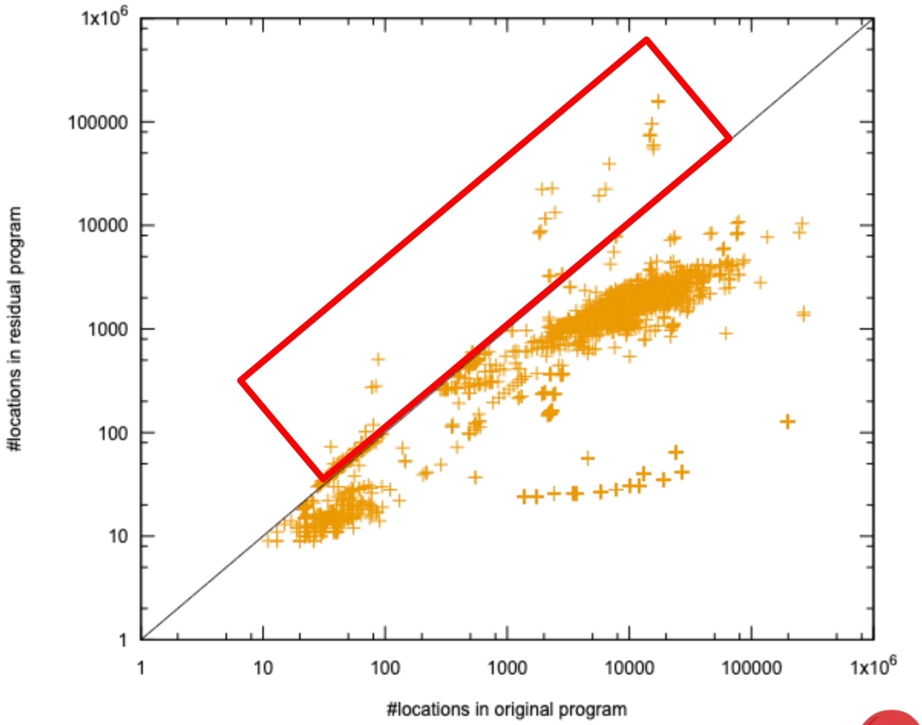


Backup Reducers: Evaluation

Challenge: Blow-up of program size

Relation program size before reduction / program size after reduction:

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



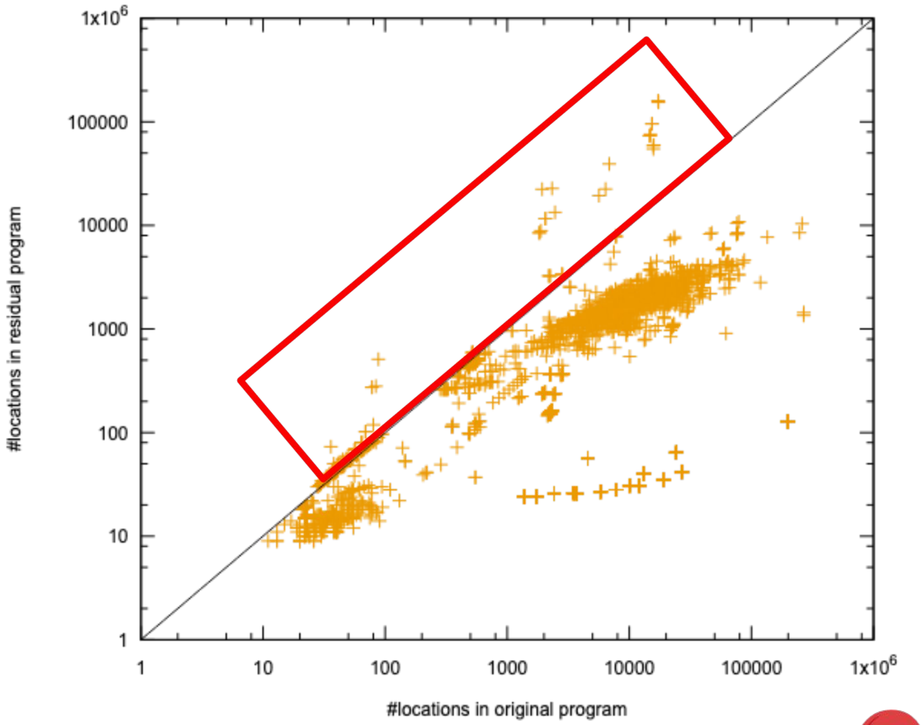
Backup Reducers: Evaluation

Challenge: Blow-up of program size

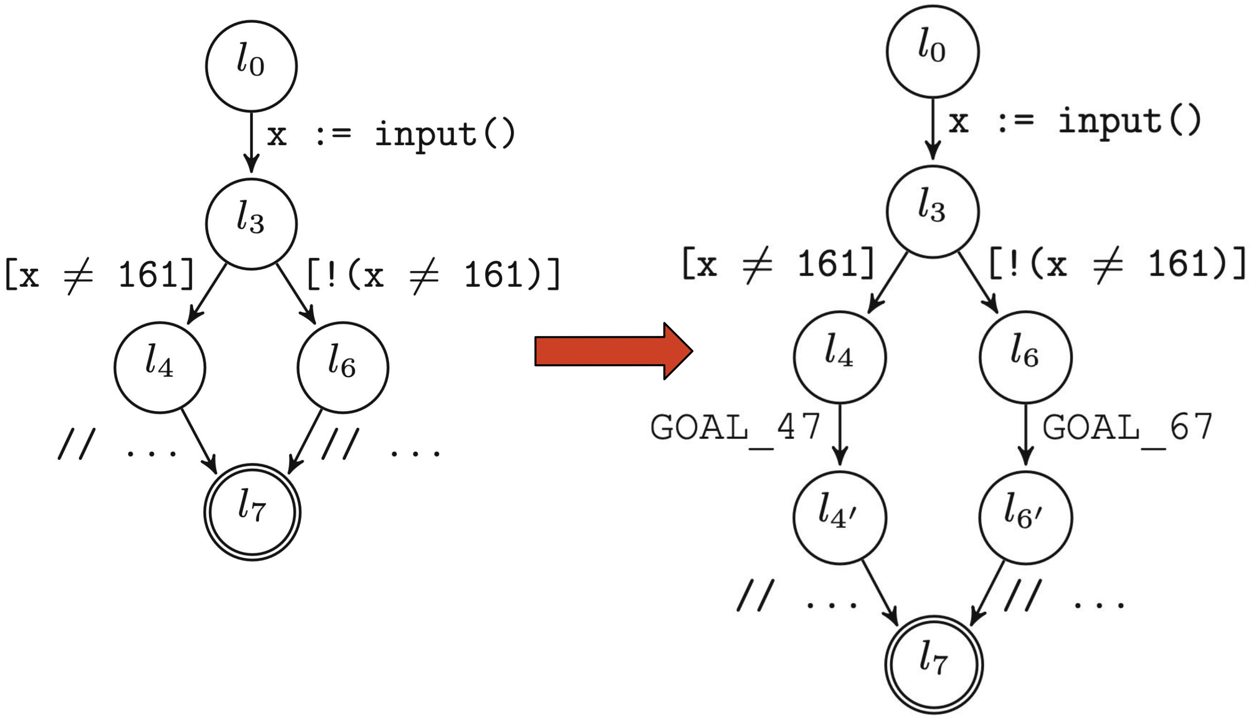
Relation program size before reduction / program size after reduction:

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

cf. *D. Beyer and M.-C. Jakobs: FRed: Conditional Model Checking via Reducers and Folders. SEFM 2020.*

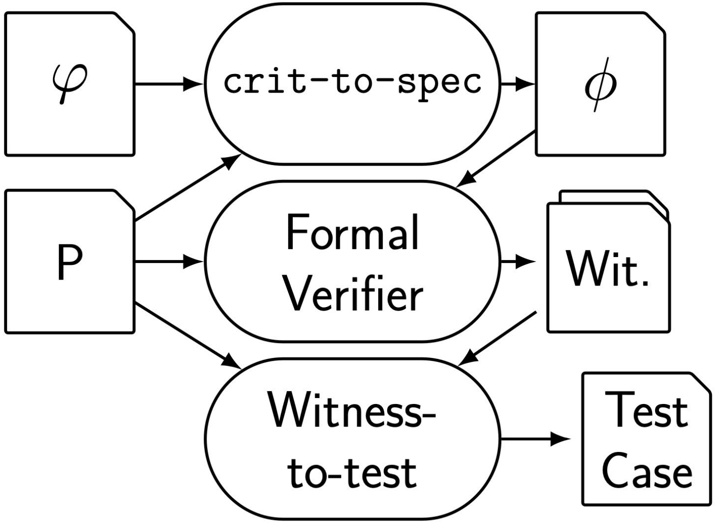


Backup CondTest: Goal Annotation

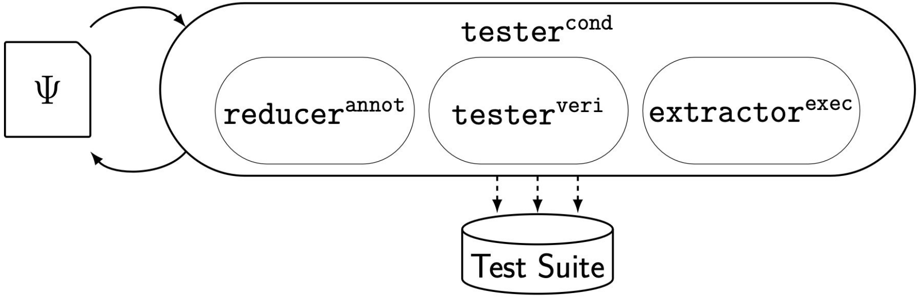



```
int main() {
  int x = input();
  if (x != 161) {
    GOAL_47:;
   // ...
  } else {
    GOAL_67:;
   // ...
  }
}
```

Backup CondTest: Verification Witnesses to Tests



- Reducer: identity + annotate goals with __VERIFIER_error
- Apply cyclic tester



Backup CondTest: Evaluation CondTest Overhead

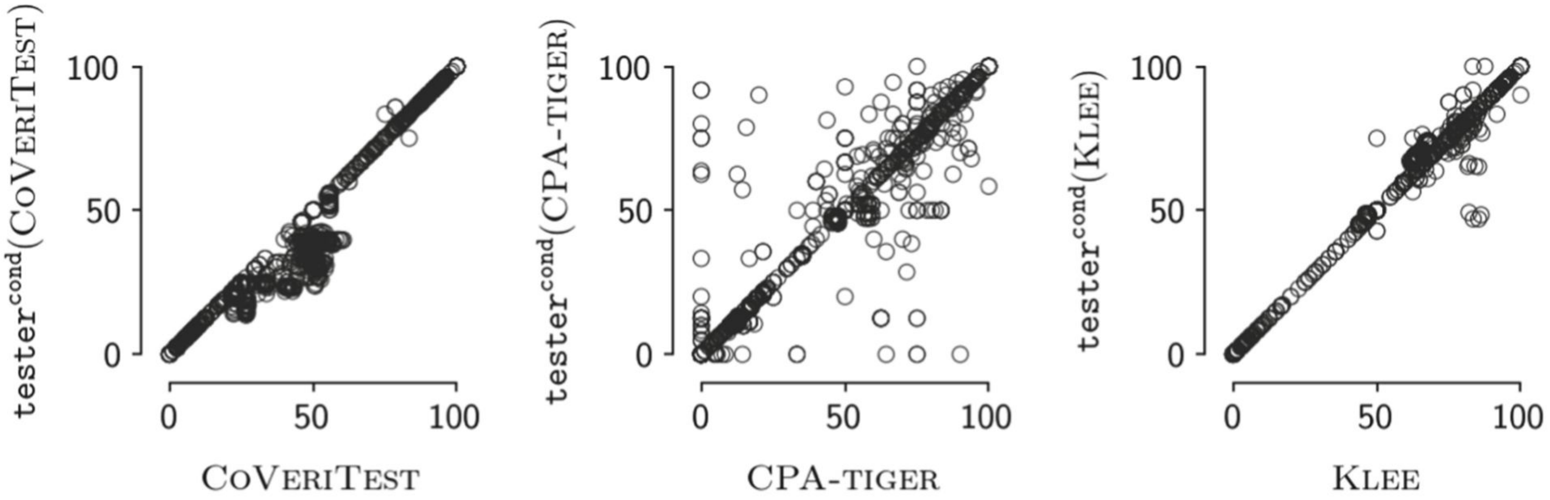
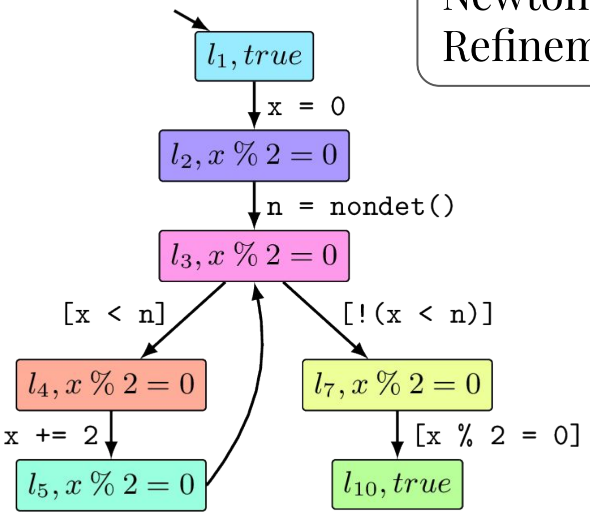


Fig. 15. Branch coverage of test suites created by original tools vs. their integration in $\texttt{tester}^{\texttt{cond}}$ (in percent)

Motivation: CEGAR (the good)

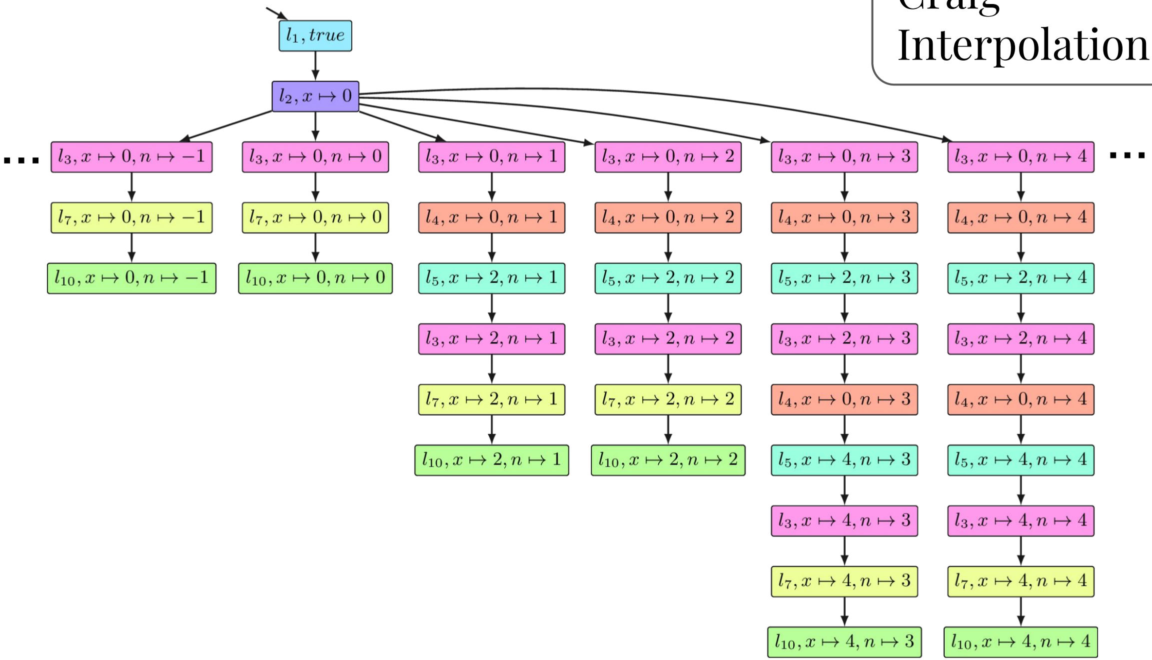
Newton Refinement




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

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

Task	branch coverage		
	prune	→	vb
Problem08_label30	5.72	+56.2	62.0
Problem08_label32	5.72	+56.1	61.9
Problem08_label06	5.72	+56.1	61.8
Problem08_label35	5.72	+56.0	61.7
Problem08_label00	5.72	+55.9	61.6
Problem08_label11	5.72	+55.8	61.5
Problem08_label19	5.72	+55.7	61.5
Problem08_label29	5.67	+55.7	61.4
Problem08_label22	5.72	+55.7	61.5
Problem08_label56	5.72	+55.7	61.5

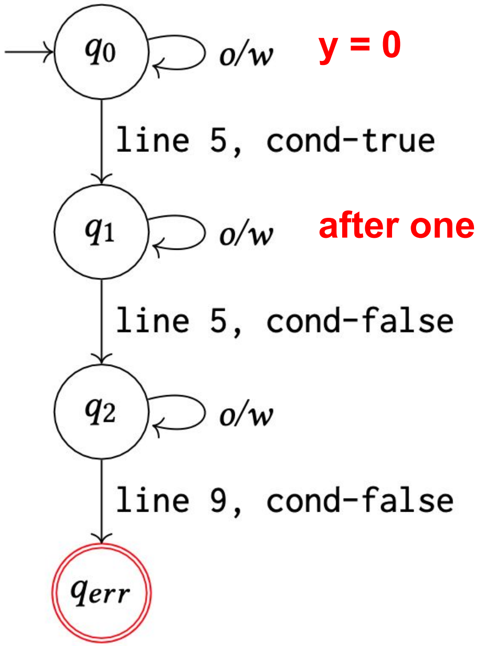
Backup C-CEGAR: CoVeriTeam Configuration

```
explorer = ActorFactory.create(ProgramValidator,
    "cpa-predicate-NoRefinement.yml");
checker = ActorFactory.create(ProgramValidator,
    "cpa-validate-violation-witnesses.yml");
refiner = ActorFactory.create(ProgramValidator,
    "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 }
```



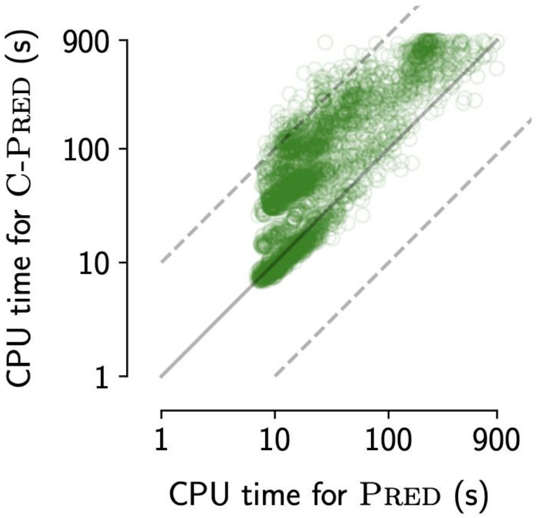
y = 0

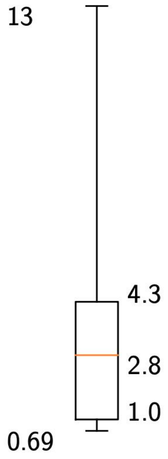
after one unrolling: x = 0

Evaluation

1. Overhead of a stateless, component-based approach (`C-Pred`)?

Efficiency:





(b) Distribution of run time factor for C-PRED, compared to PRED

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

	correct			incorrect	
	overall	proof	alarm	proof	alarm
Pred	3 769	2 556	1 213	3	9
C-Pred	3 524	2 450	1 074	0	3
C-PredWit	2 854	2 110	744	0	1

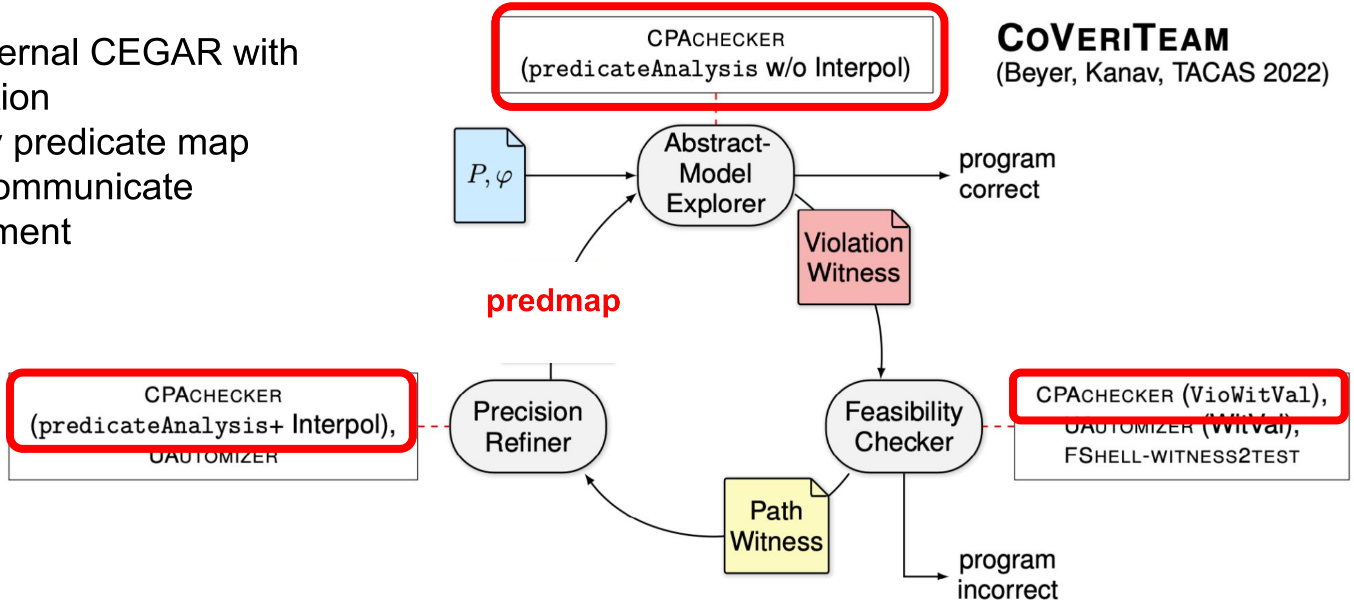
- C-* Impact on effectiveness: 6.5% decrease.
- Accounting for the speed difference: 1.7% decrease

- Witness Impact on effectiveness: 20% decrease.

Backup C-CEGAR: Evaluation

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

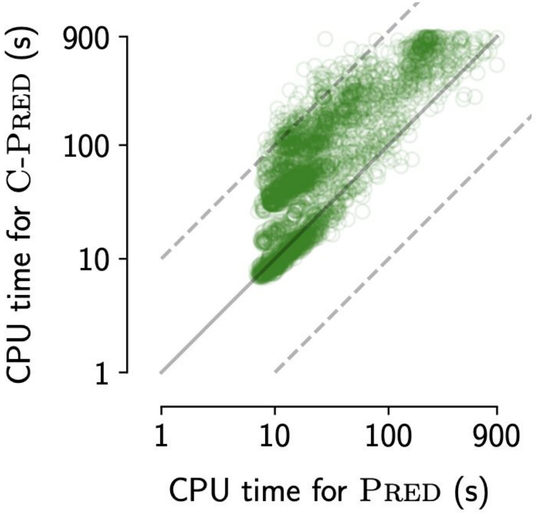
- Decompose internal CEGAR with Craig interpolation
- Use proprietary predicate map (predmap) to communicate precision increment



Backup C-CEGAR: Evaluation

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

Efficiency:

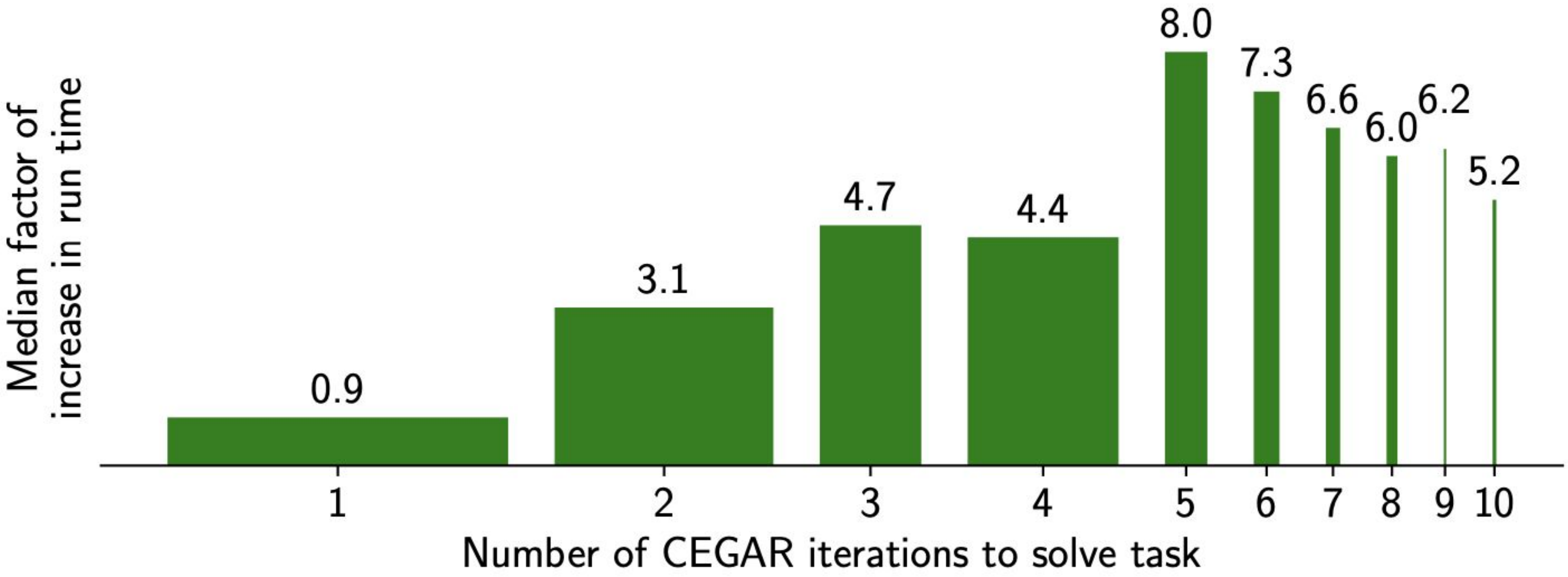


- Constant-size overhead of 13

Effectiveness:

- 6.5 % decrease
- With increased runtime limit: down to 1.7 % decrease
  - Reason: different counterexample check

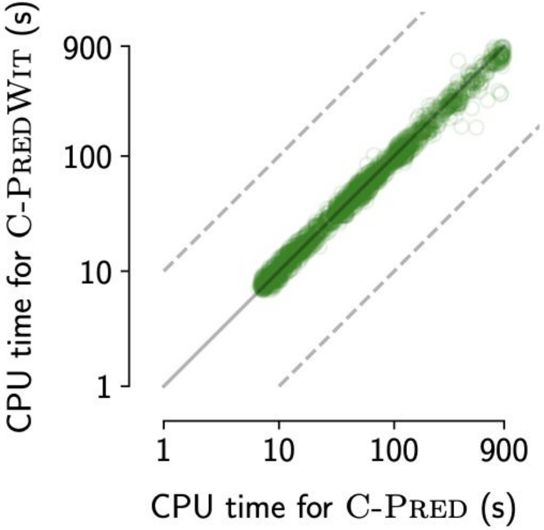
Backup C-CEGAR: Evaluation



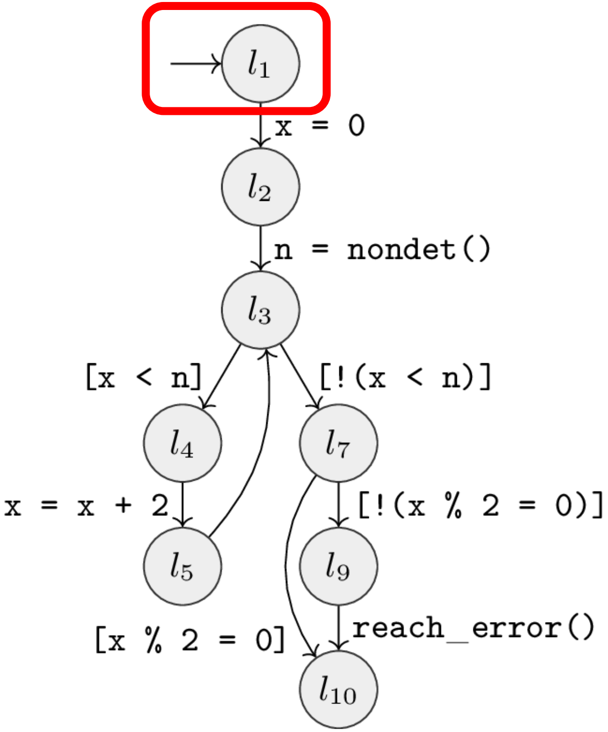
Backup C-CEGAR: Evaluation

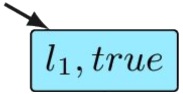
Exchange formats: Predmap (C-Pred) vs. Invariant Witnesses (C-PredWit)

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

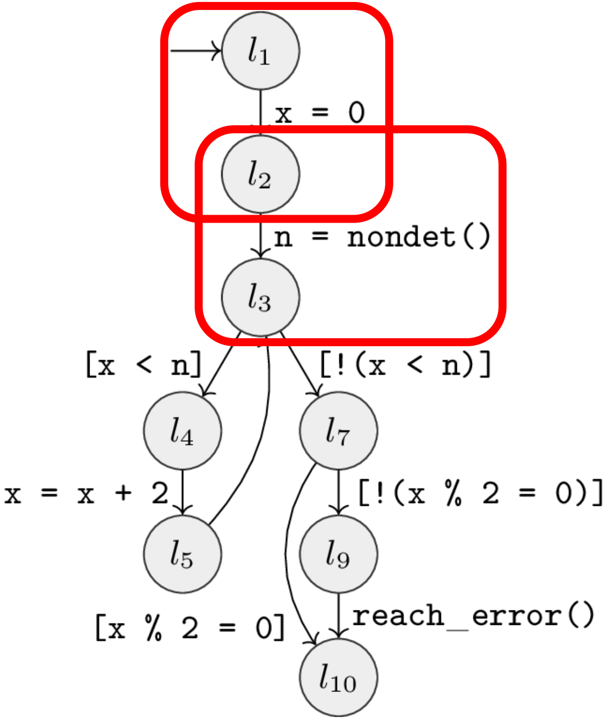


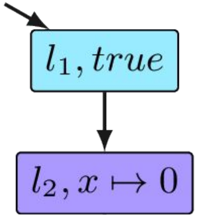
Example: Enumerative Algorithm



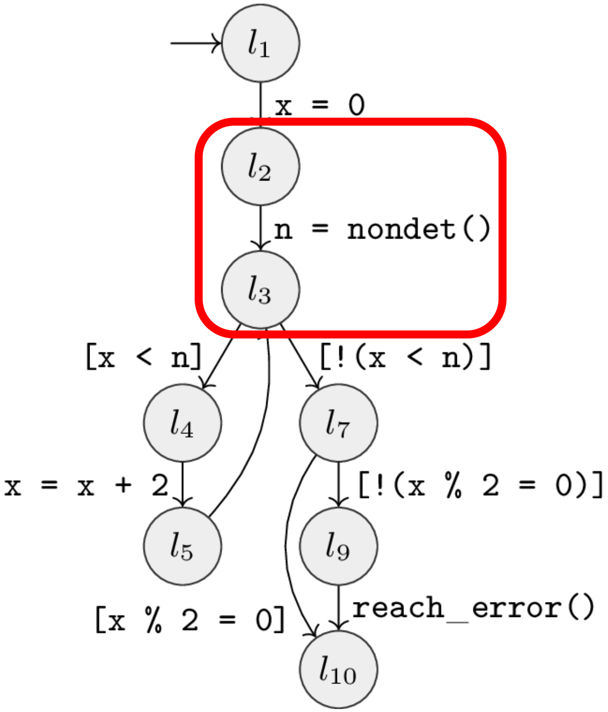


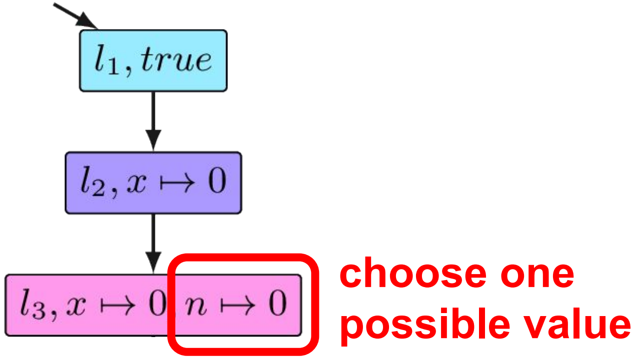
Example: Enumerative Algorithm



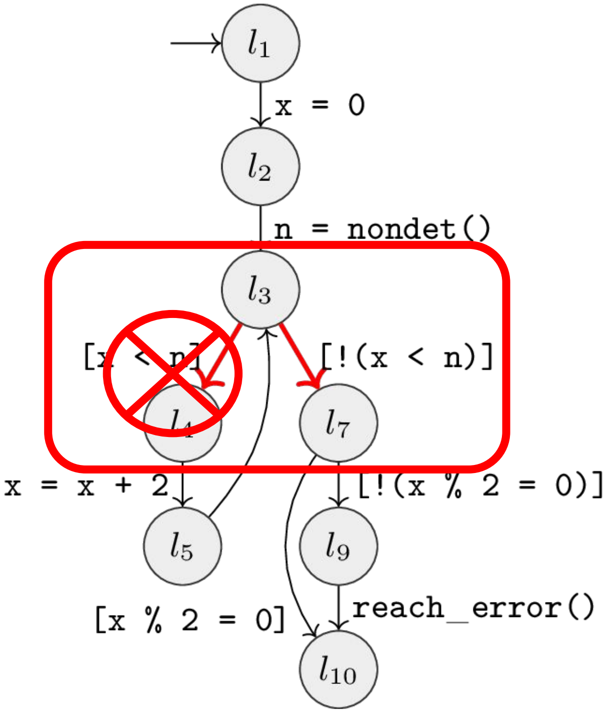


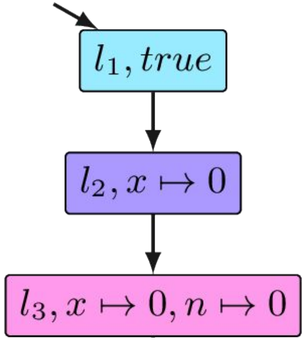
Example: Enumerative Algorithm



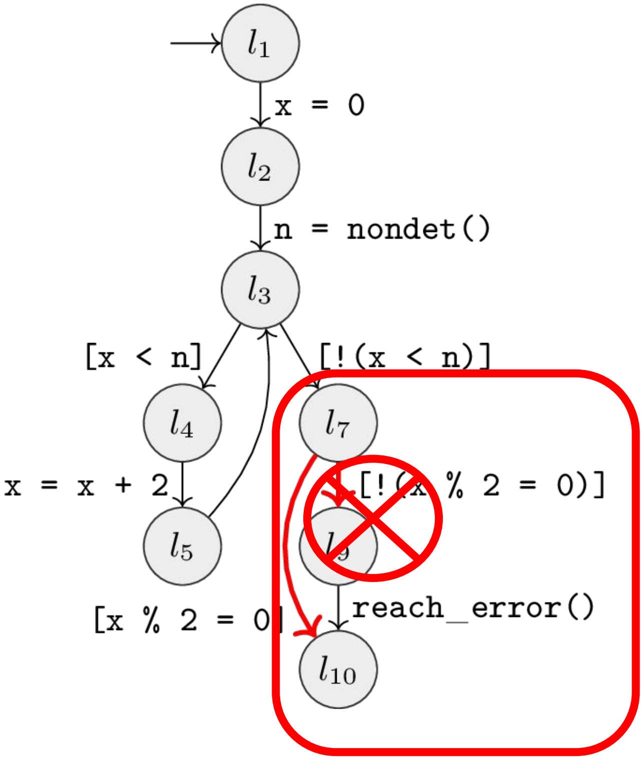


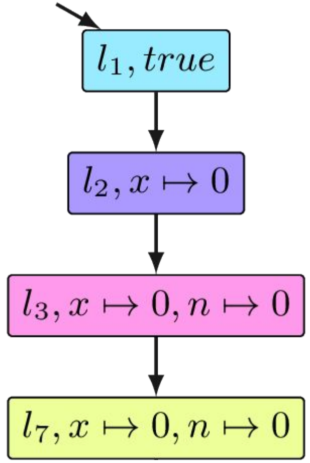
Example: Enumerative Algorithm



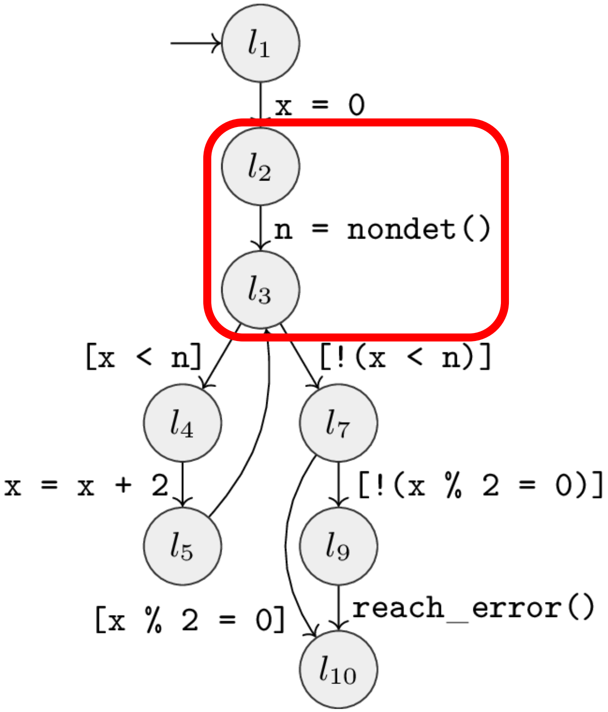


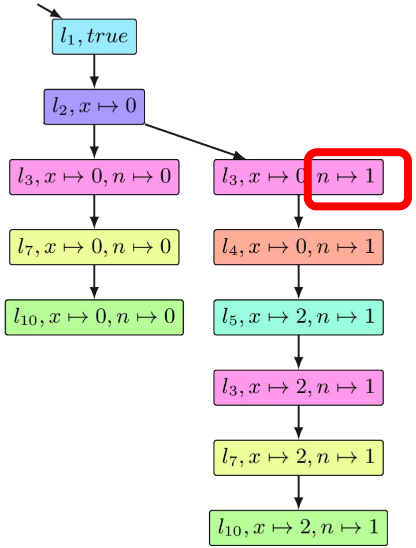
Example: Enumerative Algorithm





Example: Enumerative Algorithm





Example: Enumerative Algorithm