Thomas Lemberger

Towards Cooperative Software Verification with Test Generation and Formal Verification

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

February 28, 2024 · Ernst Denert SE-Preis · SE 2024, Linz

:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

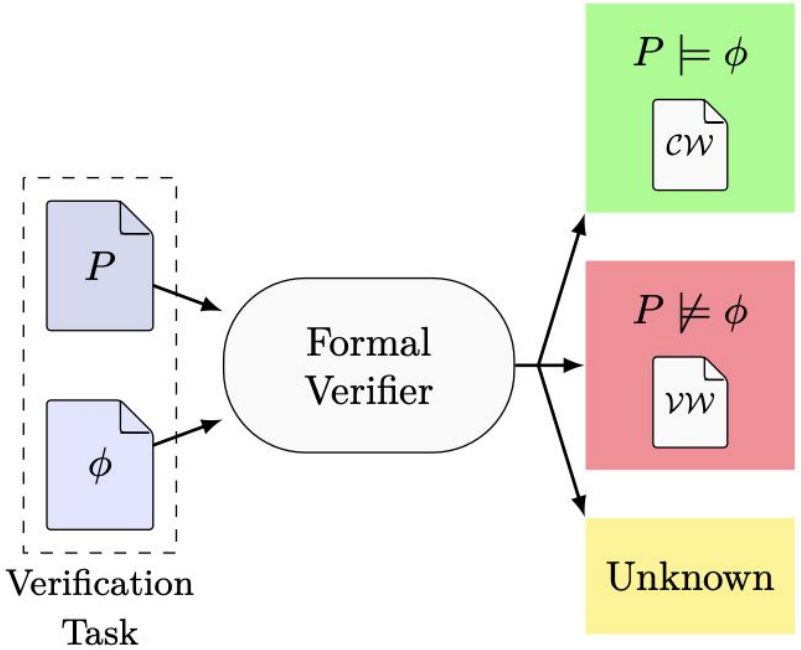

©99 ESA - CNES - ARIANESPACE / Photo Service Optique CSG.


Let’s get rid of software bugs!

- Automated Software Verification
  - Automated Formal Verification
  - Automated Test Generation
- C code without concurrency
- No:
  - Manual Tests
  - Interactive Verification
  - Code Synthesis
  - Machine Learning

```
 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 }
```

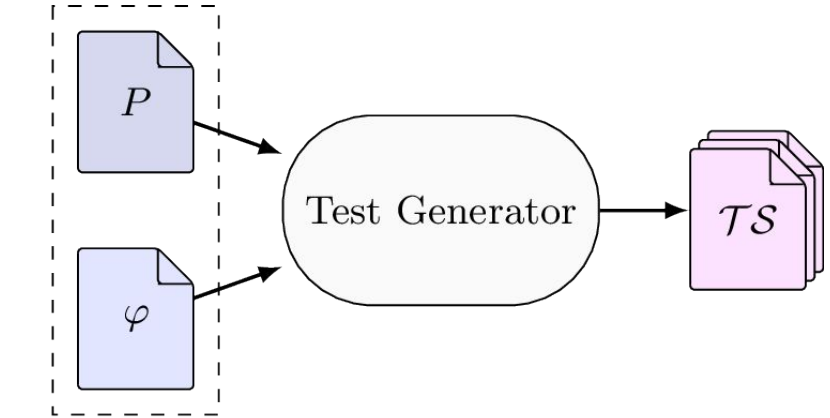
No call to `reach_error()` is reachable



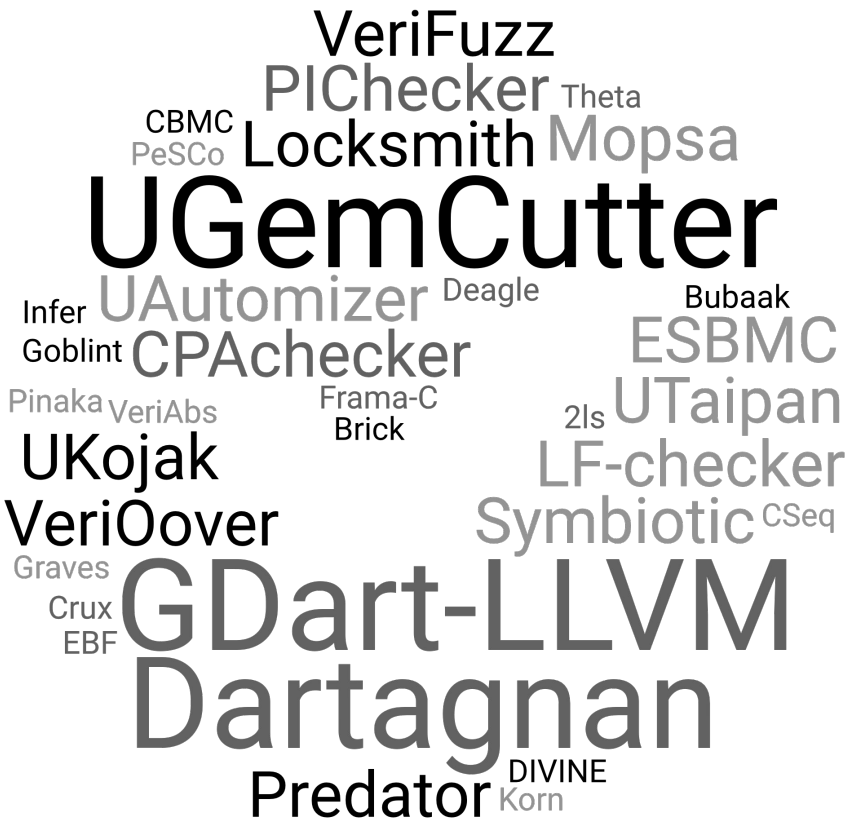
```
 1 int main(void) {
 2   unsigned int x = 0;
 3   unsigned short n = nondet();
 4   while (x < n) {
 5     x += 2;
 6   }
 7   if (x % 2 == 0) {}
 8   else
 9     reach_error();
10 }
```

Cover all program branches

Cover function `reach_error()`



Test-Generation Task


VeriFuzz
PIChecker
Theta
CBMC
Locksmith
Mopsa
PeSCo
UGemCutter
Infer
UAutomizer
Deagle
Bubaak
Goblint
CPAchecker
ESBMC
Pinaka
VeriAbs
Frama-C
Brick
2ls
UTaipan
UKojak
LF-checker
VeriOover
Symbiotic
CSeq
Graves
GDart-LLVM
Crux
EBF
Dartagnan
Predator
DIVINE
Korn


AFL
WASP-C
CoVeriTest
CPATiger
Klee
Cute
FShell
PRTest
TracerX
Crest
Legion/SymCC
FuSeBMC
Dart

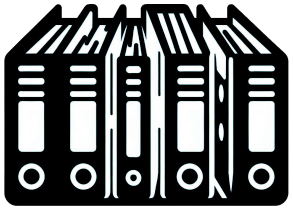
1. Tool comparisons

2. Concepts for combining verifiers and testers, off-the-shelf

Common input language

Common output language

Reliable measurements



Benchmark Set

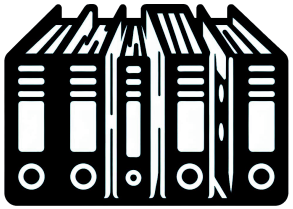
Result Format

Measurement Tools

ETAPS
EUROPEAN JOINT CONFERENCES ON THEORY & PRACTICE OF SOFTWARE

TACAS 2023

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



Benchmark Set

Result Format

Measurement Tools

sv-benchmarks	Witnesses	Witness Validators

Beyer. **Competition on Software Verification and Witness Validation: SV-COMP 2023.** *Proc. TACAS, 2023.*

Comparison of Testers: 404

```
// CREST:
CREST_int(x);
// KLEE:
klee_make_symbolic(&x,
  sizeof(x), “x”);
// AFL:
scanf(“%d\n”, &x);
```

```
// CREST:
0
// KLEE:
Structured data
// AFL:
\x00
```

```
Input methods
+ result format
```

Benchmark Set	Result Format	Measurement Tools
Proprietary formats	Proprietary formats	Proprietary execution methods, no real branch coverage

Comparison of Testers

Robust test-suite execution *through Linux cgroups and overlay file systems*[3]

Coverage measurement *through code instrumentation*

```
extern int __VERIFIER_nondet_int();
// .. snip ..
int x = __VERIFIER_nondet_int();
```

```
<testcase>
  <input>1023</input>
  <input>254</input>
</testcase>
```

Benchmark Set	Result Format	Measurement Tools
Subset of sv-benchmarks[1]	Test-Suite Format[2]	TestCov[2] branch coverage, `reach_error()` coverage

[1] *Beyer, Lemberger.* **Software Verification: Testing vs. Model Checking.** *HVC, 2017.*
[2] *Beyer, Lemberger.* **TestCov: Robust Test-Suite Execution and Coverage Measurement**. *Proc. ASE, 2019.*
[3] *Beyer, Löwe, Wendler.* **Reliable Benchmarking: Requirements and Solutions.** *STTT, 2019.*

6th Competition on Software Testing (Test-Comp 2024)

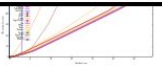
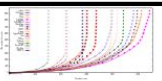
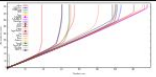
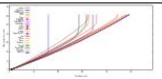
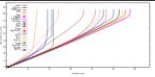
- ~11.000 test tasks
  - Cover-Error
  - Cover-Branches
- Offline competition
- 900s, 15GB memory per task
- Score: normalized accumulated coverage

Participants	Plots	cetfuzz	CoVeriTest	ESBMC-kind	FDSE	Fizzer	FuSeBMC	FuSeBMC-AI	HybridTiger	KLEE	KLEE
SoftwareSystems-DeviceDriversLinux64-ReachSafety 2 tasks		0	0	0	0	0	0	0	0	0	0
CPU time		0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s
Cover-Branches 9825 tasks		2197	4826	-	5132	5146	5478	5418	3987	3023	497
CPU time		1700000 s	5700000 s		4500000 s	6200000 s	8800000 s	8500000 s	4900000 s	2700000 s	610000
ReachSafety-Arrays 468 tasks		96	316	-	382	375	389	389	305	152	351
CPU time		150000 s	340000 s		40000 s	290000 s	420000 s	420000 s	320000 s	64000 s	310000
ReachSafety-BitVectors 62 tasks		36	49	-	49	48	49	49	16	32	48
CPU time		35000 s	21000 s		34000 s	41000 s	56000 s	56000 s	7900 s	22000 s	32000
ReachSafety-ControlFlow 87 tasks		47	50	-	53	54	58	58	21	36	56
CPU time		61000 s	48000 s		24000 s	63000 s	77000 s	77000 s	48000 s	41000 s	39000
ReachSafety-ECA											

Test-Comp 2023: *Beyer*. **Software Testing: 5th Comparative Evaluation: Test-Comp 2023.** *Proc. FASE, 2023*.

6th Competition on Software Testing (Test-Comp 2024)

- 2024 *(to be published)*:
  - 20 participants from 7 countries
  - 8 first-time participants
- All participants produce test suites in our XML format
- TestCov performs all coverage measurement

Participants	Plots	cetfuzz	CoVeriTest	ESBMC-kind	FDSE	Fizzer	FuSeBMC	FuSeBMC-AI	HybridTiger	KLEE	KLEE
DeviceDriversLinux64-ReachSafety 2 tasks		0	0	0	0	0	0	0	0	0	0
CPU time		0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s
Cover-Branches 9825 tasks		2197	4826	-	5132	5146	5478	5418	3987	3023	497
CPU time		1700000 s	5700000 s		4500000 s	6200000 s	8800000 s	8500000 s	4900000 s	2700000 s	610000
ReachSafety-Arrays 468 tasks		96	316	-	382	375	389	389	305	152	351
CPU time		150000 s	340000 s		40000 s	290000 s	420000 s	420000 s	320000 s	64000 s	310000
ReachSafety-BitVectors 62 tasks		36	49	-	49	48	49	49	16	32	48
CPU time		35000 s	21000 s		34000 s	41000 s	56000 s	56000 s	7900 s	22000 s	32000
ReachSafety-ControlFlow 87 tasks		47	50	-	53	54	58	58	21	36	56
CPU time		61000 s	48000 s		24000 s	63000 s	77000 s	77000 s	48000 s	41000 s	39000
ReachSafety-ECA		8	5		8	7	10	9	3	7	9

Test-Comp 2023: *Beyer*. **Software Testing: 5th Comparative Evaluation: Test-Comp 2023.** *Proc. FASE, 2023.*

6th Competition on Software Testing (Test-Comp 2024)

- I participate with PRTest[1], a Plain Random Tester
- < 500 lines of code

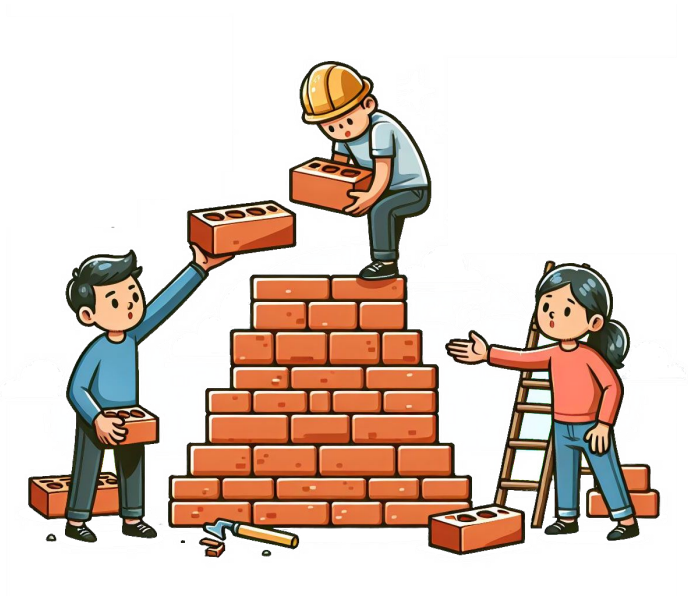
- Idea: simple baseline
- Fun fact: **best** for branch coverage in category 'Hardware'

cetfuzz	CoVeriTest	ESBMC-kind	FDSE	Fizzer	FuSeBMC	FuSeBMC-AI	HybridTiger	KLEE	KLEEF	Legion	Legion/SymCC	Owi	PRTest
559	851	–	682	862	751	760	631	286	698	569	640	1	965
10000 s	780000 s		530000 s	930000 s	960000 s	960000 s	860000 s	680000 s	920000 s	930000 s	920000 s	930000 s	960000 s

[1] *Lemberger.* **Plain random test generation with PRTest.** *STTT, 2020.*

Limitations of Test-Comp

- Only branch coverage + bug finding (no MC/DC, mutation testing)
- One fixed resource limit
  - 900s CPU time, 15 GB memory
- *Only a benchmark set*
- … but the largest available for the verification of C:
  - busybox, Linux, OpenBSD, sqlite, coreutils, …
- TestCov does not support concurrency



6th Competition on Software Testing (Test-Comp 2024)

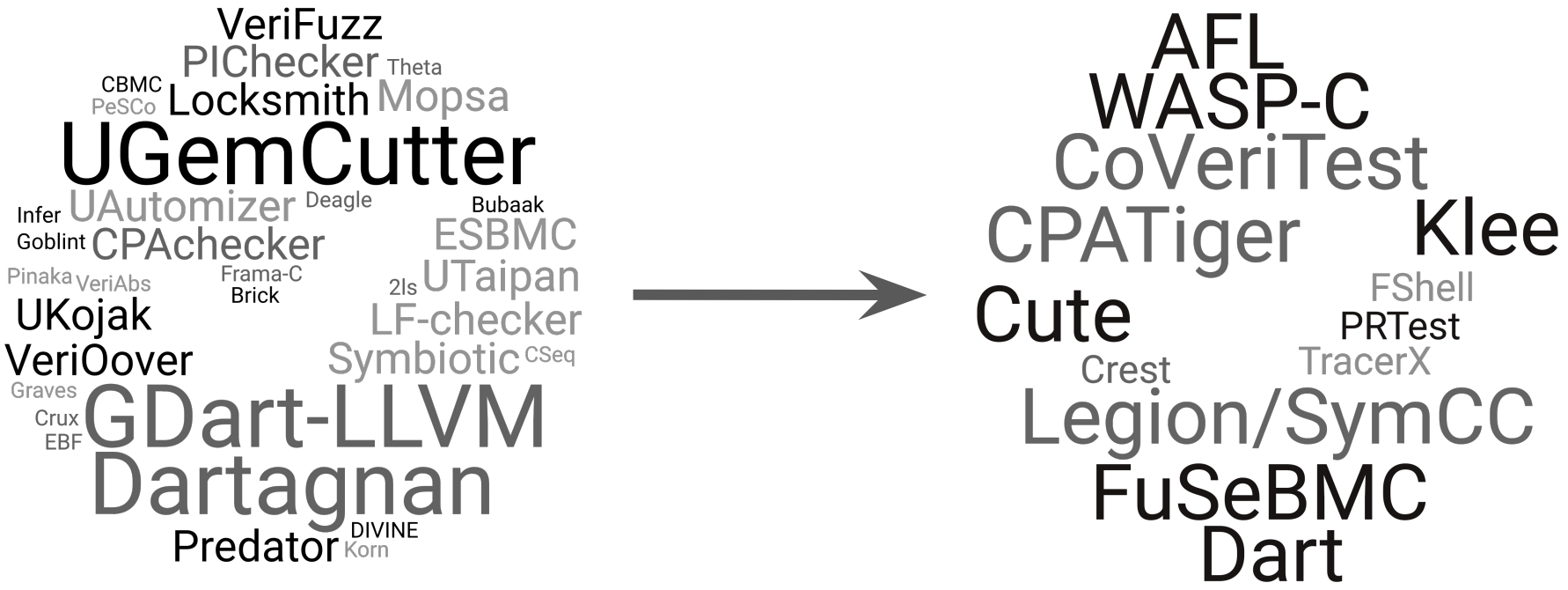
12th Competition on Software Verification (SV-COMP 2023)

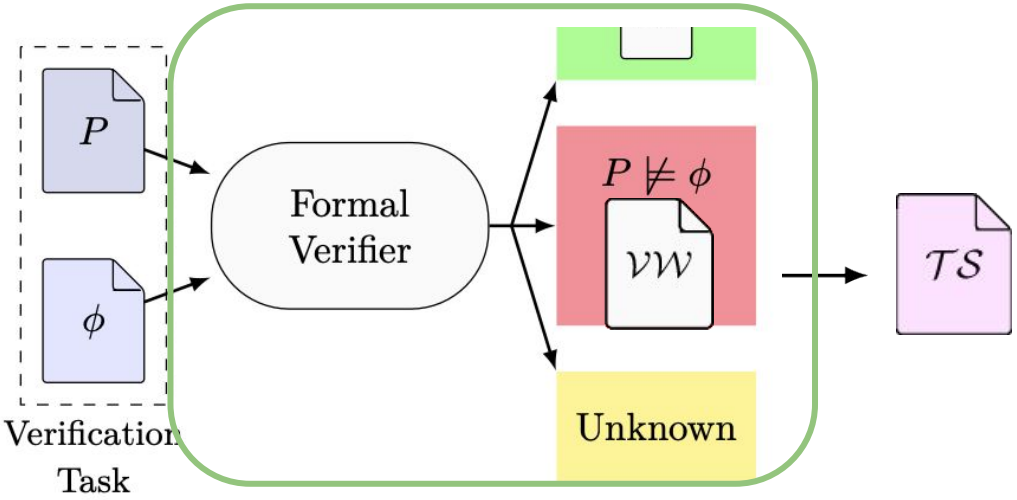
- Common input- and output-language simplify cooperation off-the-shelf

- Cooperation for formal verification:
  - Reducer-based construction of conditional verifiers *(ICSE 2018)*
  - Decomposing CEGAR with witness formats *(ICSE 2022)*

- Cooperation for testing:
  - **Tests from witnesses**
  - **Conditional testing**

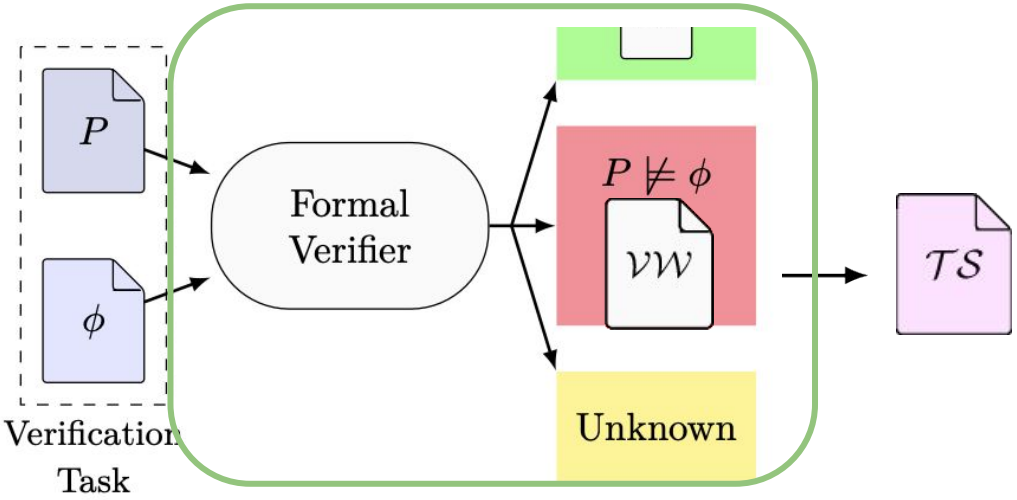
- Weakness: Information loss in witness export


VeriFuzz
PIChecker
Theta
CBMC
PeSCo
Locksmith
Mopsa
UGemCutter
Infer
UAutomizer
Deagle
Bubaak
Goblint
CPAchecker
ESBMC
Pinaka
VeriAbs
Frama-C
Brick
2ls
UTaipan
UKojak
LF-checker
VeriOover
Symbiotic
CSeq
Graves
Crux
EBF
GDart-LLVM
Dartagnan
Predator
DIVINE
Korn
AFL
WASP-C
CoVeriTest
CPATiger
Klee
Cute
FShell
PRTest
TracerX
Crest
Legion/SymCC
FuSeBMC
Dart



- Exists for proprietary formats[1]
- But we turn any SV-COMP verifier into a test generator[2]
- Advantages:
  - Precise validation through execution
  - Well-known user experience
- Tool: `CPA-witness2test`

[1]*Beyer, Chlipala, Henzinger, Jhala, Majumdar:* **Generating Tests from Counterexamples.** *Proc. ICSE, 2004.*
Visser, Pasareanu, Kurshid. **Test Input Generation with Java PathFinder.** *Proc. ISSTA, 2004.*
[2]*Beyer, Dangl, Lemberger, Tautschnig.* **Tests from Witnesses: Execution-Based Validation of Verification Results.** *Proc. TAP, 2018.*

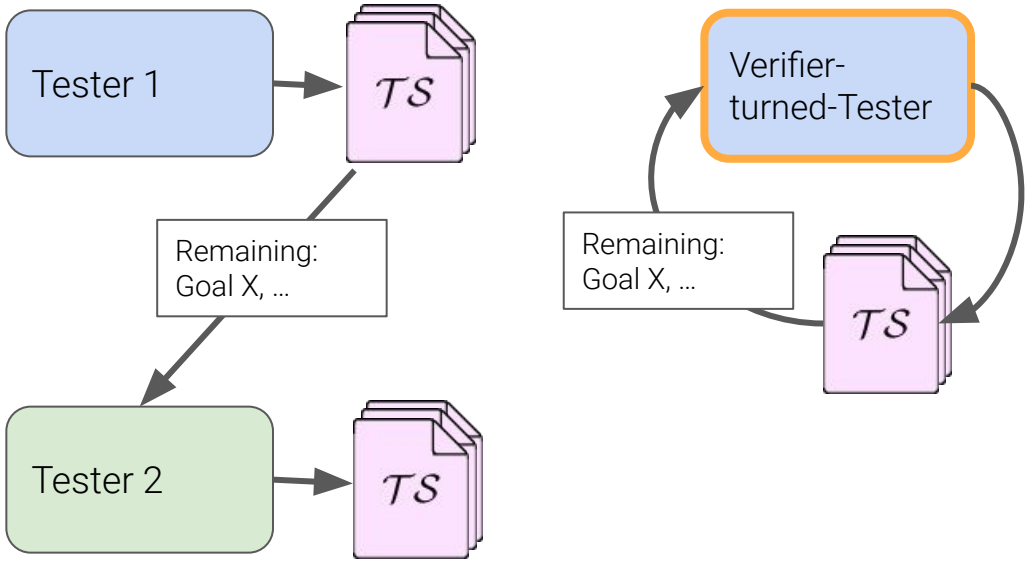


- Limitations:
  - Only works if witness precise enough
  - Only single test case

[1]*Beyer, Chlipala, Henzinger, Jhala, Majumdar:* **Generating Tests from Counterexamples.** *Proc. ICSE, 2004.*
Visser, Pasareanu, Kurshid. **Test Input Generation with Java PathFinder.** *Proc. ISSTA, 2004.*
[2]*Beyer, Dangl, Lemberger, Tautschnig.* **Tests from Witnesses: Execution-Based Validation of Verification Results.** *Proc. TAP, 2018.*


VeriFuzz
PIChecker
Theta
CBMC
PeSCo
Locksmith
Mopsa
UGemCutter
Infer
UAutomizer
Deagle
Goblint
CPAchecker
Pinaka
VeriAbs
Frama-C
Brick
UKojak
VeriOover
Graves
Crux
EBF
GDart-LLVM
Dartagnan
Predator
DIVINE
Korn
AFL
WASP-C
CoVeriTest
Tiger
Klee
FShell
PRTest
TracerX
Crest
Legion/SymCC
FuSeBMC
Dart

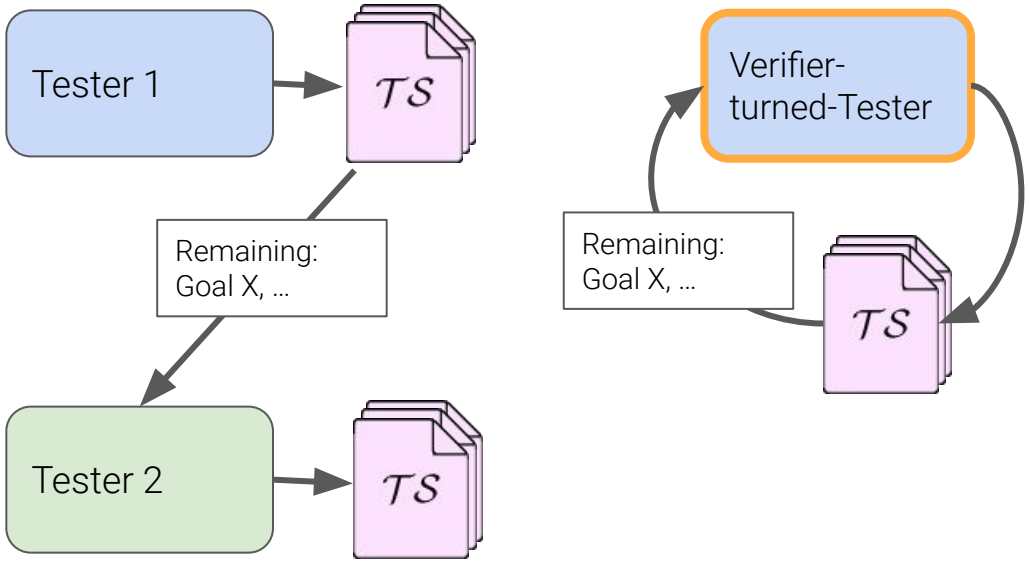
Conditional Testing[1]



- Communicate information about remaining coverage goals through coverage measurement and code transformation
- Tool: **`CondTest`**

[1] *Beyer, Lemberger.* **Conditional Testing - Off-the-Shelf Combination of Test-Case Generators.** *Proc. ATVA, 2019.*
[2] *Majumdar, Sen.* **Hybrid Concolic Testing.** *Proc. ICSE 2007.*
Daca, Gupta, Henzinger. **Abstraction-driven Concolic Testing.** *Proc. VMCAI 2015.*
[3] *Beyer, Henzinger, Keremoglu, Wendler.* **Conditional Model Checking: A Technique to Pass Information between Verifiers.** *Proc. FSE, 2012.*
[4] *Beyer, Jakobs, Lemberger, Wehrheim.* **Reducer-Based Construction of Conditional Verifiers.** *Proc. ICSE, 2018.*

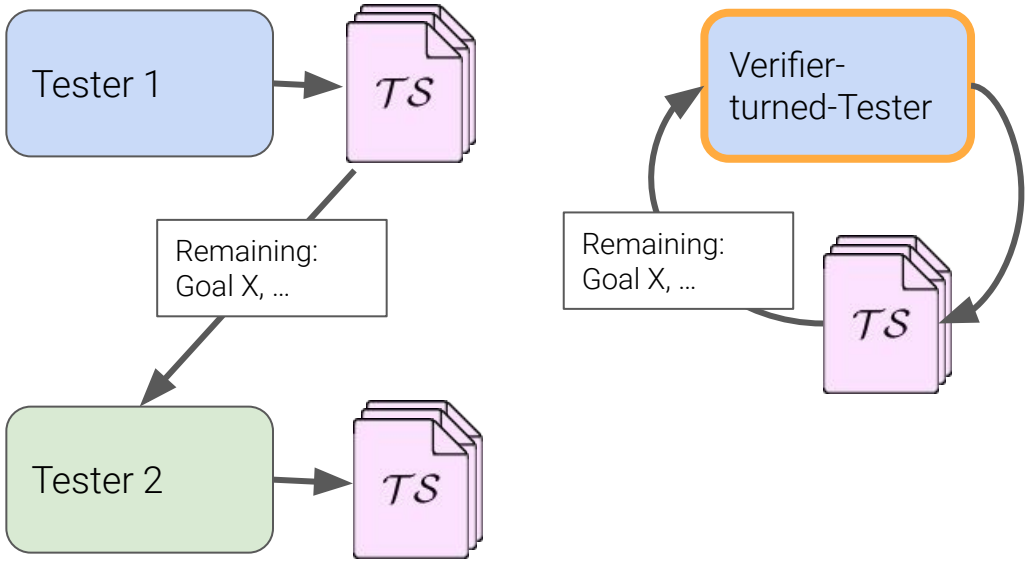
Conditional Testing[1]



- In-tool cooperations existed before[2]
- We developed a similar approach for formal verifiers[3]

[1] *Beyer, Lemberger.* **Conditional Testing - Off-the-Shelf Combination of Test-Case Generators.** *Proc. ATVA, 2019.*
[2] *Majumdar, Sen.* **Hybrid Concolic Testing.** *Proc. ICSE 2007.*
Daca, Gupta, Henzinger. **Abstraction-driven Concolic Testing.** *Proc. VMCAI 2015.*
[3] *Beyer, Jakobs, Lemberger, Wehrheim.* **Reducer-Based Construction of Conditional Verifiers.** *Proc. ICSE, 2018.*

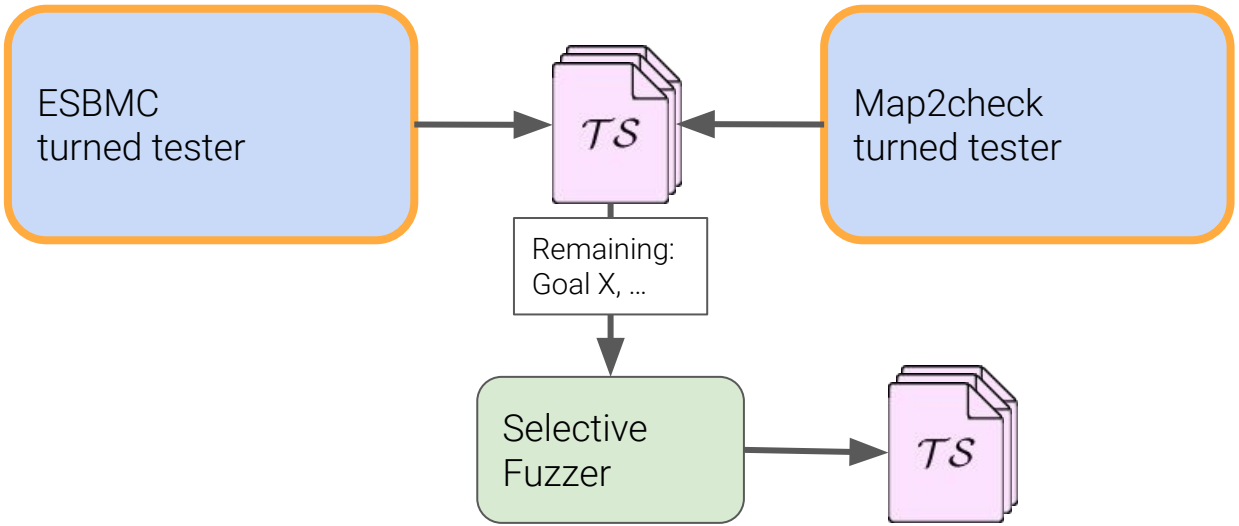
Conditional Testing[1]



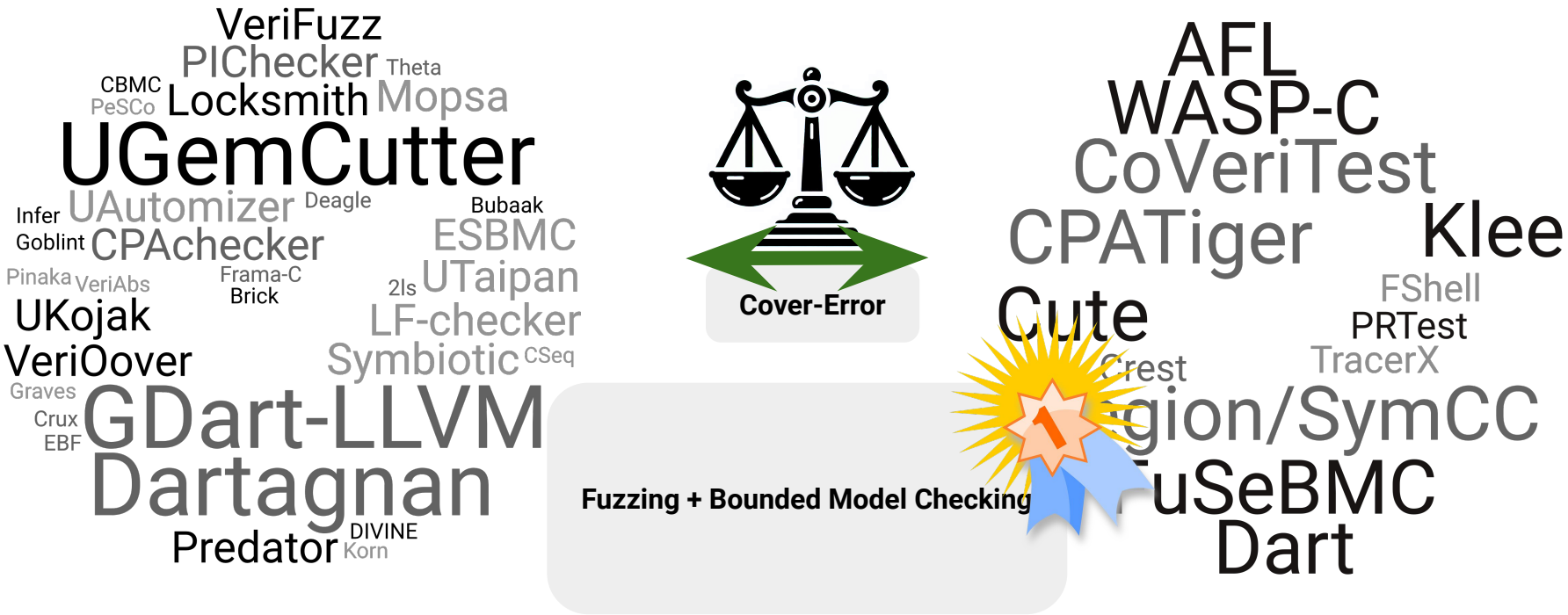
- Advantages:
  - Avoid redundant work
  - Flexible combinations
  - Turn verifiers through cyclic combination into full-fledged test generators
- Limitations: Code transformations are imprecise (syntax-based) or expensive (semantics-based)

[1] *Beyer, Lemberger.* **Conditional Testing - Off-the-Shelf Combination of Test-Case Generators.** *Proc. ATVA, 2019.*

- FuSeBMC[1] uses this technique to combine ESBMC and Map2check
- Overall winner of Test-Comp 2022, 2023, and 2024

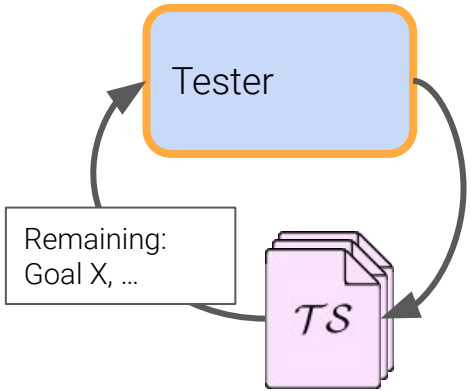


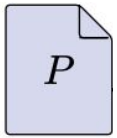
[1]*Alshmrany, Menezes, Gadelha, Cordeiro.* **FuSeBMC: A White-Box Fuzzer for Finding Security Vulnerabilities in C Programs (Competition Contribution).** *Proc. FASE, 2021.*



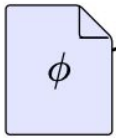
Beyer, Lemberger. **Six Years Later: Testing vs. Model Checking.** *Under submission, STTT, 2024.*

- Made Test-Comp possible: Fair comparison, common language for > 20 testers
- Accidentally showed that plain random testing can be very effective
- Turned all SV-COMP verifiers into test generators
- Concepts and tooling for off-the-shelf cooperations
- Conditional testing used by Test-Comp winner FuSeBMC
- All tools are open source and evaluation results publicly available



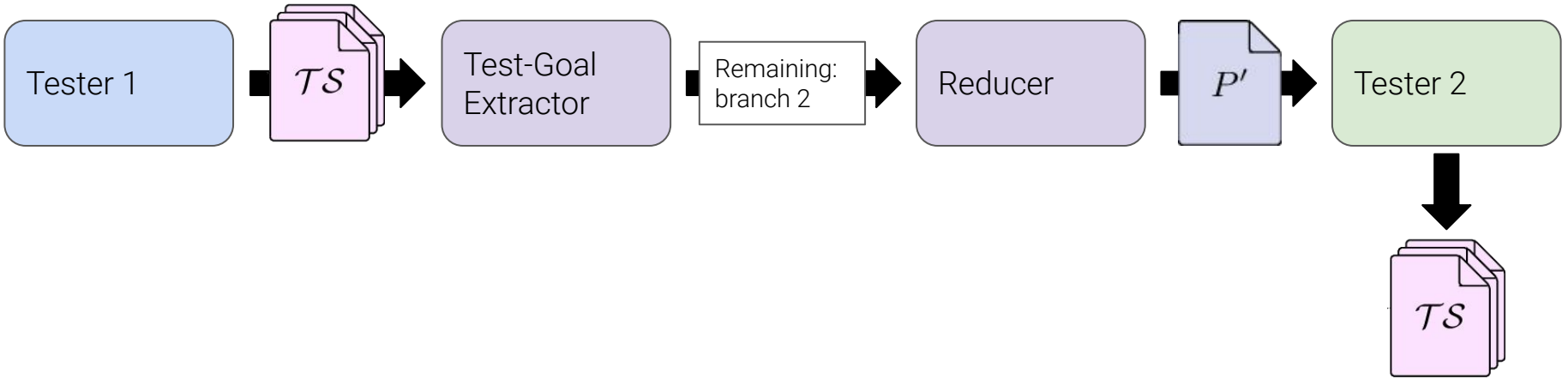



```
extern unsigned int __VERIFIER_nondet_uint();
int main() {
  int i, n=__VERIFIER_nondet_uint(), sn=0;
  for(i=1; i<=n; i++) {
    if (i<10)
      sn = sn + a;
  }
  __VERIFIER_assert(sn==n*a || sn == 0);
 }
```

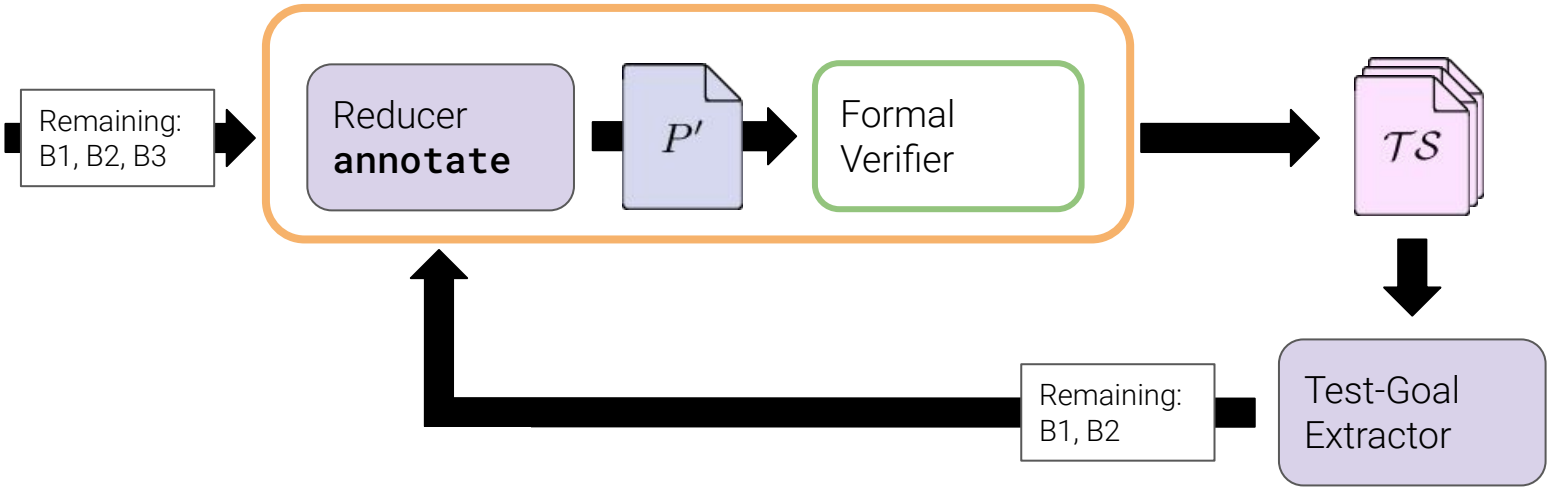



```
CHECK( init(main()), LTL(G !
call(reach_error())) )
```

Conditional Testing



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

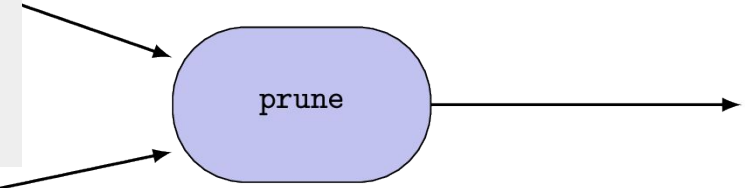


- Turn any verifier into a full-blown test generator

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

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

- Stop program execution when it can't reach any remaining goal
- Syntactic reachability only
- Poor opportunities for pruning



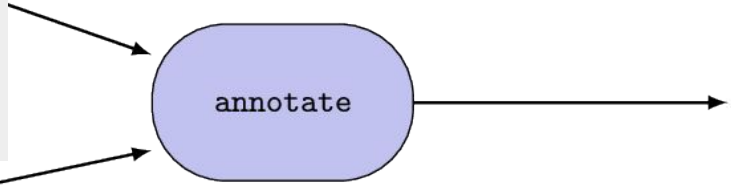
Remaining goal: *branch 2*

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

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

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

- Annotate relevant goals with **`reach_error()`**
- For formal verifiers



Remaining goal: *branch 2*

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

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

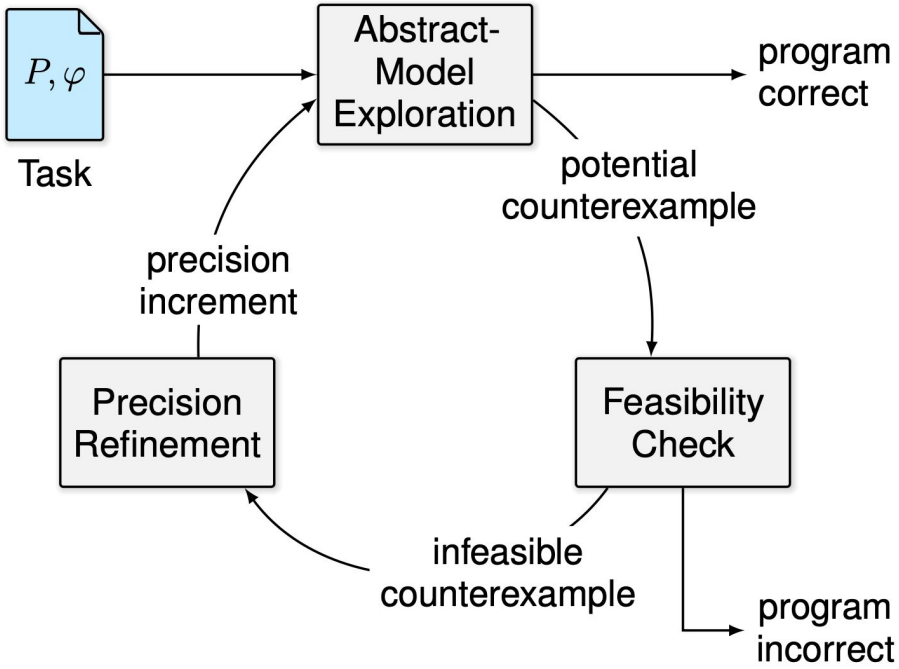
Decomposing Software Verification


```yaml
[...]
content:
 - invariant:
    type: loop_invariant
    location:
      file_name: "./program.c"
      line: 4
      column: 9
      function: main
    value: "x % 2 == 0"
    format: c_expression
```

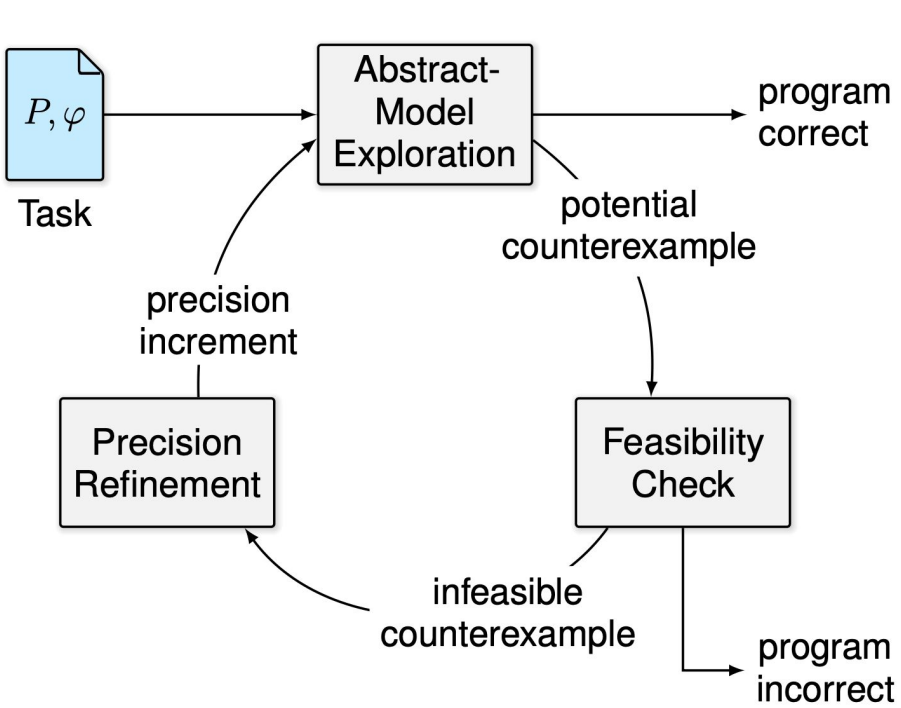
Beyer, Dangl, Dietsch, Heizmann, Lemberger, Tautschnig. **Verification Witnesses.** *ACM Trans. Softw. Eng. Methodol., 2022.*

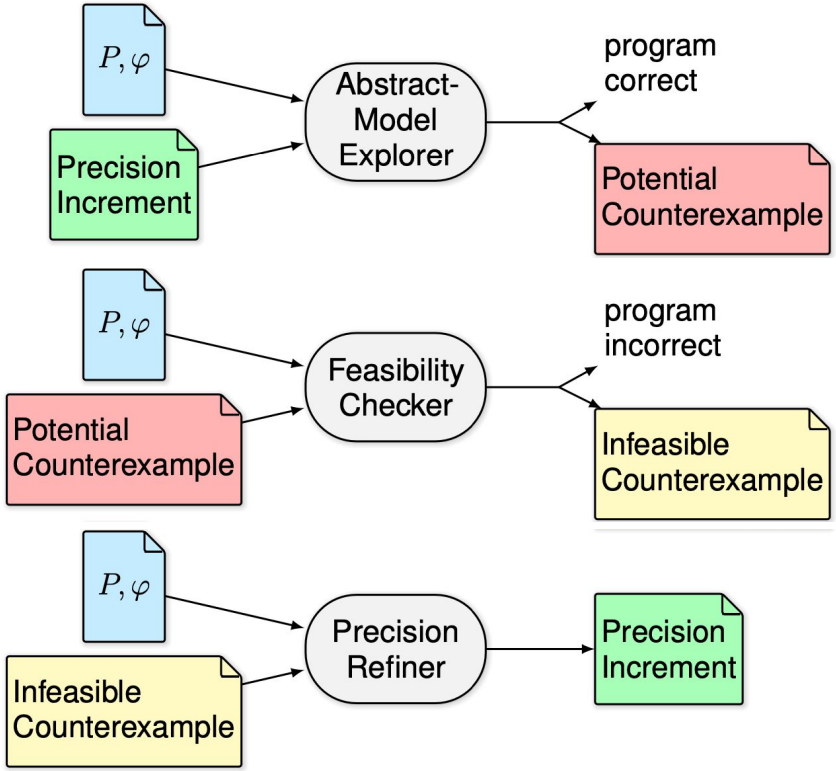

```yaml
[...]
content:
 - segment:
   - waypoint:
       type: "branching"
       action: "follow"
       constraint:
         value: "false"
       location:
         file_name: "program.c"
         line: 7
         column: 4
 - segment:
   - waypoint:
       type: "target"
       action: "follow"
       location:
         file_name: "program.c"
         line: 9
         column: 3
```

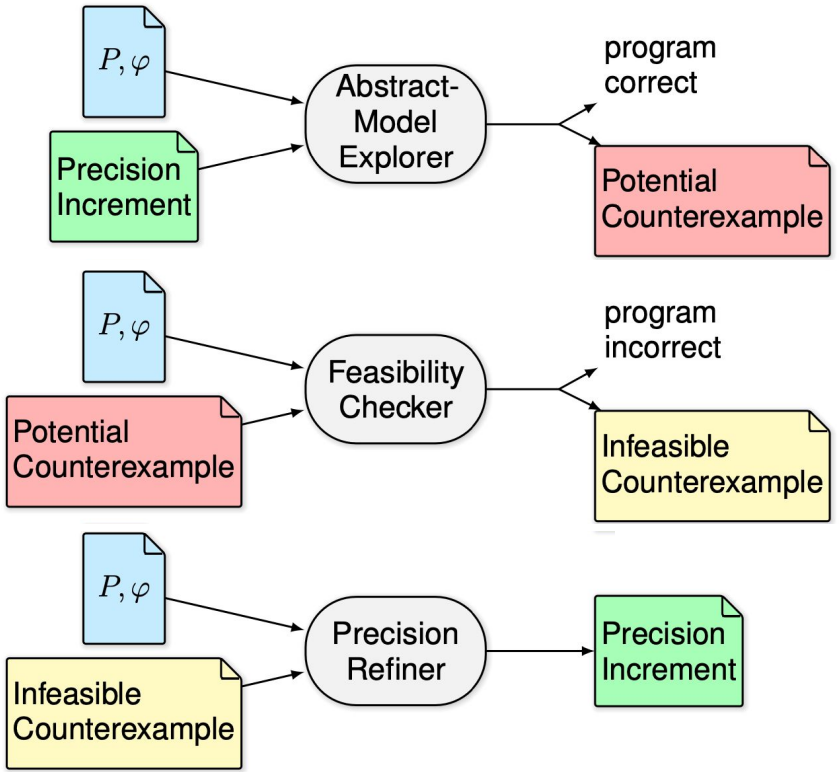
Beyer, Dangl, Dietsch, Heizmann, Lemberger, Tautschnig. **Verification Witnesses.** *ACM TOSEM, 2022.*



- Common underlying schema
- Many tools implement CEGAR
- New idea → new implementation


P, φ
Task
Abstract-Model Exploration
program correct
potential counterexample
precision increment
Precision Refinement
Feasibility Check
infeasible counterexample
program incorrect


P, φ
Precision Increment
Abstract-Model Explorer
program correct
Potential Counterexample
P, φ
Potential Counterexample
Feasibility Checker
program incorrect
Infeasible Counterexample
P, φ
Infeasible Counterexample
Precision Refiner
Precision Increment

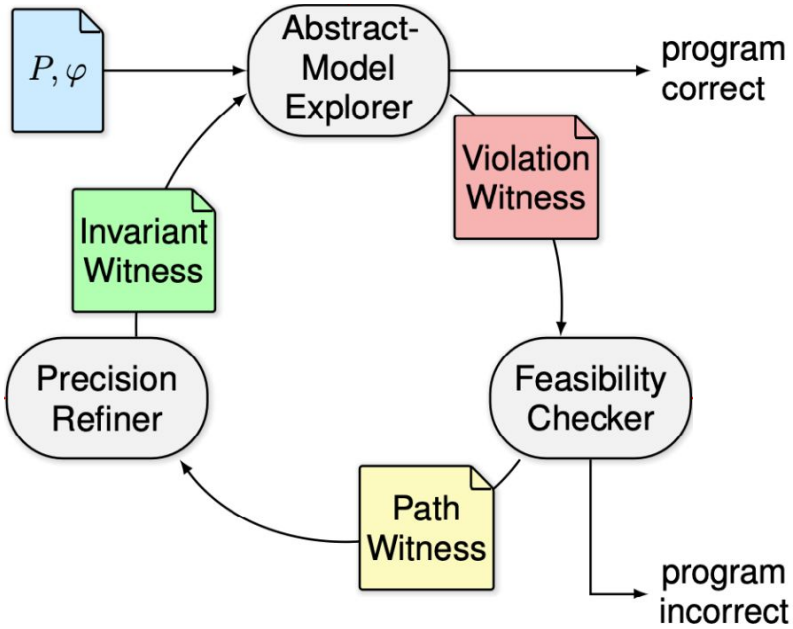


Exchange formats from SV-COMP
→ wide tool support

Violation Witness — Abstract description of counterexample

Path Witness — Abstract description of rejected counterexample (“violation” witness)

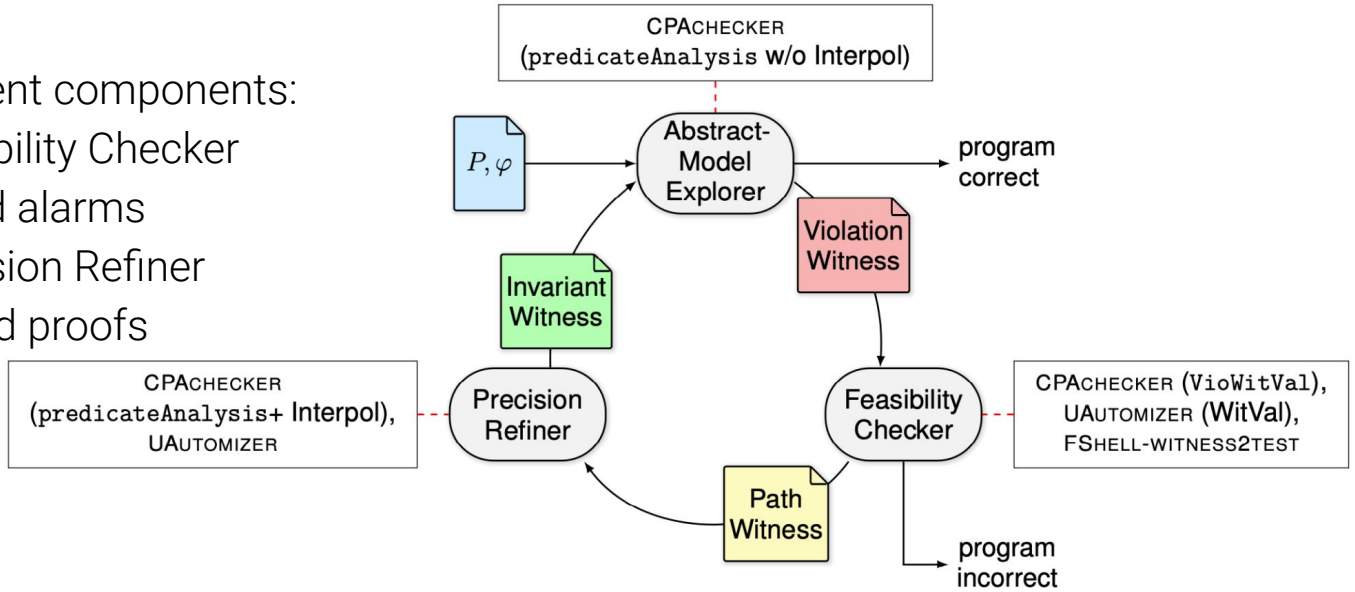
Invariant Witness — Description of candidate invariants (“correctness” witness)


P, φ
Abstract-Model Explorer
program correct
Violation Witness
Invariant Witness
Precision Refiner
Feasibility Checker
Path Witness
program incorrect

- Implementation in CoVeriTeam

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

3. Benefits from different components:
   - Different Feasibility Checker
     +93 found alarms
   - Different Precision Refiner
     + 29 found proofs



- Reliable and Reproducible Coverage Measurement
- Native test execution of alien programs is risky
- gcov and llvm-cov do not report actual *branch* coverage

```c
#include <stdio.h>
int main() {
    int x; scanf("%d", &x);
    int y; scanf("%d", &y);

    if (x > 0 && y > x) {
        return 0;
    } else {
        return 1;
    }
}
```

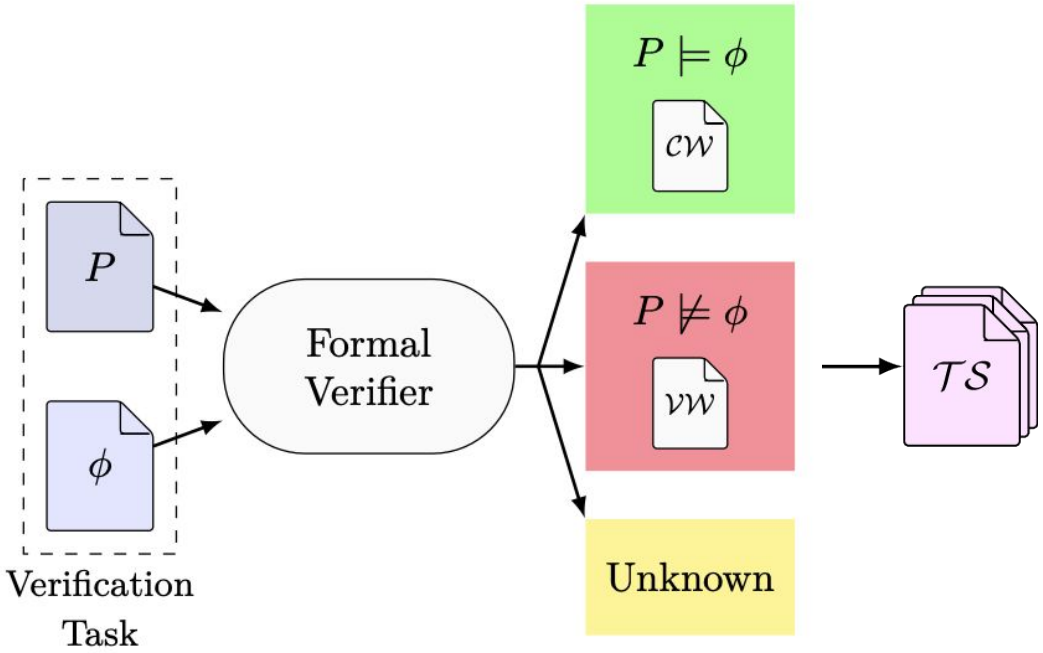
Input: x = -1, y = 0

```
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
branch  2 never executed
branch  3 never executed
#####:    7:        return 0;
    -:    8:    } else {
    1:    9:        return 1;
    -:   10:    }
    -:   11:}
```

- Reliable and Reproducible Coverage Measurement
- Native test execution of alien programs is risky
- gcov and llvm-cov do not report actual *branch* coverage

- **TestCov** provides:
  - Coverage instrumentation on the C-code level with clang libtooling
  - Lightweight containerization with BenchExec, per test execution
  - Support for sv-benchmarks properties and TestComp test-suite formats

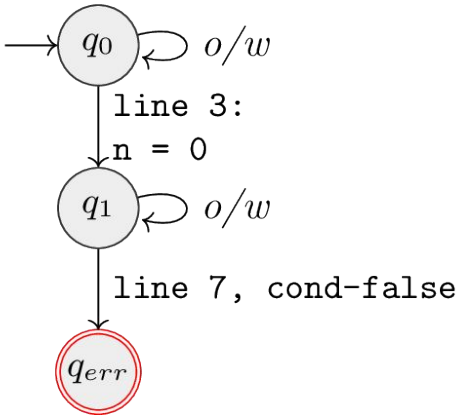
- Used in Test-Comp for 6 years


$P \models \phi$
cw
P
$P \not\models \phi$
Formal Verifier
vw
$\mathcal{TS}$
ϕ
Unknown
Verification Task

Mention related work: Tests from counterexamples, seahorn

- Make any verifier a test-case generator
- Foundation: established standard of violation witnesses[1]

```
 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 }
```

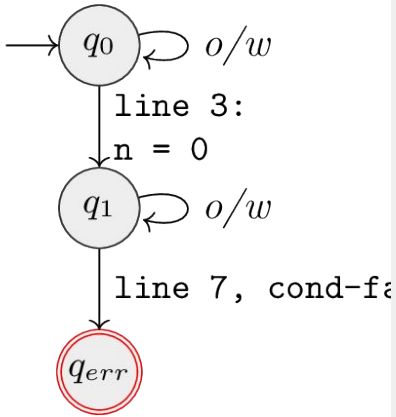


[1] *Beyer, Dangl, Dietsch, Heizmann, Lemberger, Tautschnig.* **Verification Witnesses.** *ACM Trans. Softw. Eng. Methodol., 2022.*

- Make any verifier a test-case generator

1. Map concrete state-space guards in violation witness to input methods
2. Create harness

```
 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 }
```




```
 1 ushort nondet() {
 2   static int calls = 0;
 3   unsigned short return_val;
 4   switch (calls) {
 5   case 0:
 6     return_val = 0;
 7   default:
 8     abort();
 9   }
10   calls++;
11   return return_val;
12 }
```

[1] *Beyer, Dangl, Lemberger, Tautschnig.* **Tests from Witnesses: Execution-Based Validation of Verification Results.** *Proc. TAP 2018.*

- Two implementations: CPA-witness2test and FShell-witness2test
- Limitation: Witness must contain concrete input vector (strengthening witness is possible)
- Success rate: 35 %

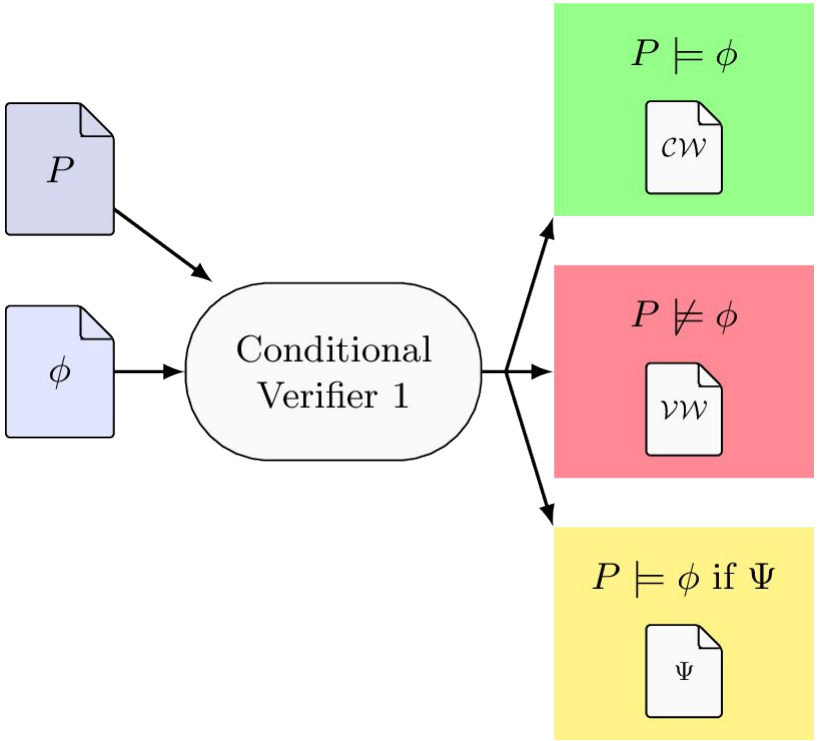
Verifier		Produced witnesses			Produced tests			
		Unref.	Ref.	Total	Count	kLOC	kB	# Inputs (Avg.)
2LS	[45]	992	384	1 376	1 208	89.9	3 999	7.57
BLAST	[47]	778	202	980	327	29.0	938	0.271
CBMC	[34]	831	467	1 298	1 249	67.7	2 991	6.33
CEAGLE		619	426	1 045	540	92.2	262	5.39
CPA-BAM-BNB	[2]	851	175	1 026	158	42.9	1 114	0
CPA-KIND	[10]	263	193	456	656	56.2	2 967	14.9
CPA-SEQ	[23]	883	767	1 650	838	95.5	3 895	1.79
DEPTHK	[43]	1 159	305	1 464	1 302	65.4	3 170	2.96
ESBMC	[37]	653	148	801	478	21.0	1 983	2.53
ESBMC-FALSI	[37]	981	395	1 376	1 133	53.7	1 906	1.81
ESBMC-INCR	[37]	970	392	1 362	1 126	53.5	1 896	1.82
ESBMC-KIND	[24]	847	352	1 199	1 028	48.9	1 774	1.69
FORESTER	[30]	51	0	51	0	0	0	-
PREDATORHP	[33]	86	61	147	80	17.2	434	0
SKINK	[17]	30	25	55	44	0.290	8	0
SMACK	[41]	871	632	1 503	1 576	128	5 654	6.09
SYMBIOTIC	[19]	927	411	1 338	589	38.1	1 375	0
SYMDIVINE	[38]	247	224	471	405	13.4	580	0
UAUTOMIZER	[29]	514	70	584	121	2.24	59	0
UKOJAK	[40]	309	67	376	116	2.15	55	0
UTAIPAN	[26]	338	70	408	121	2.23	59	0
Total		13 200	5 766	18 966	13 095	920	35 119	5.60

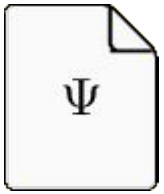
- Make any verifier a test-case generator

- Advantages:
  - Compared to previous work, over XX verifiers can be turned into testers
  - Formal techniques are very good at finding bugs [**HVC paper**]
  - Bugs found by formal verifiers can be examined through execution
- Limitations:
  - Witness must contain concrete input vector
  - At this state, verifier only produces a single witness → only a single test

- Transfers conditional model checking to difference verification
- Turn any verifier into incremental verifier

- Transfers conditional model checking to difference verification
- Turn any verifier into incremental verifier

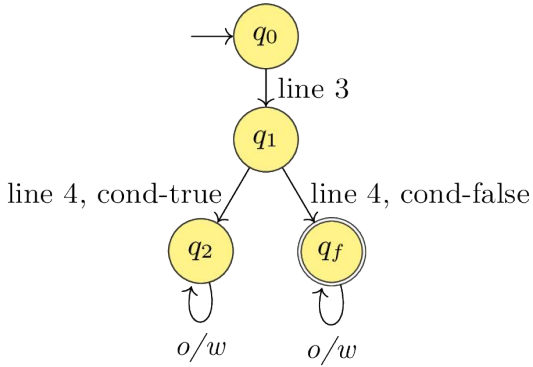




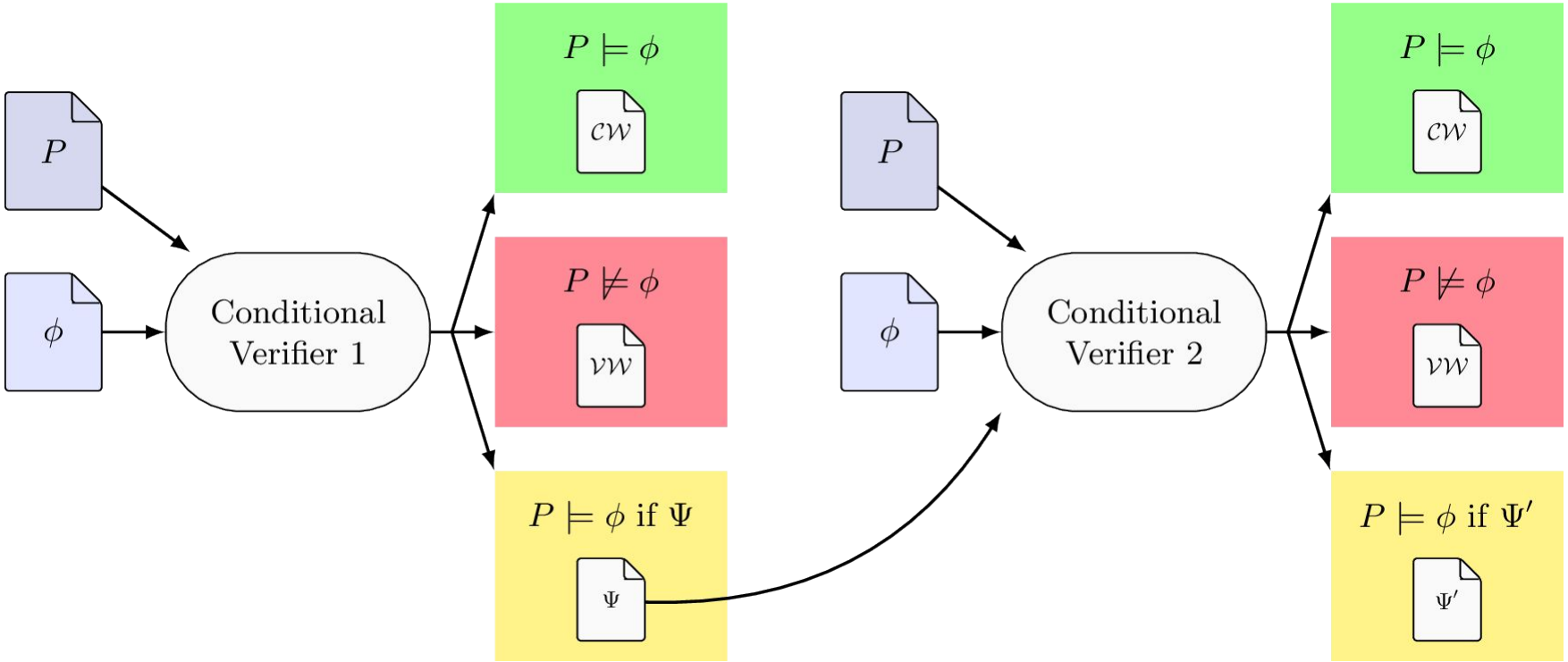
:

Condition. Automaton that describes the already-explored state-space with *source-code guards* and *state-space guards*.

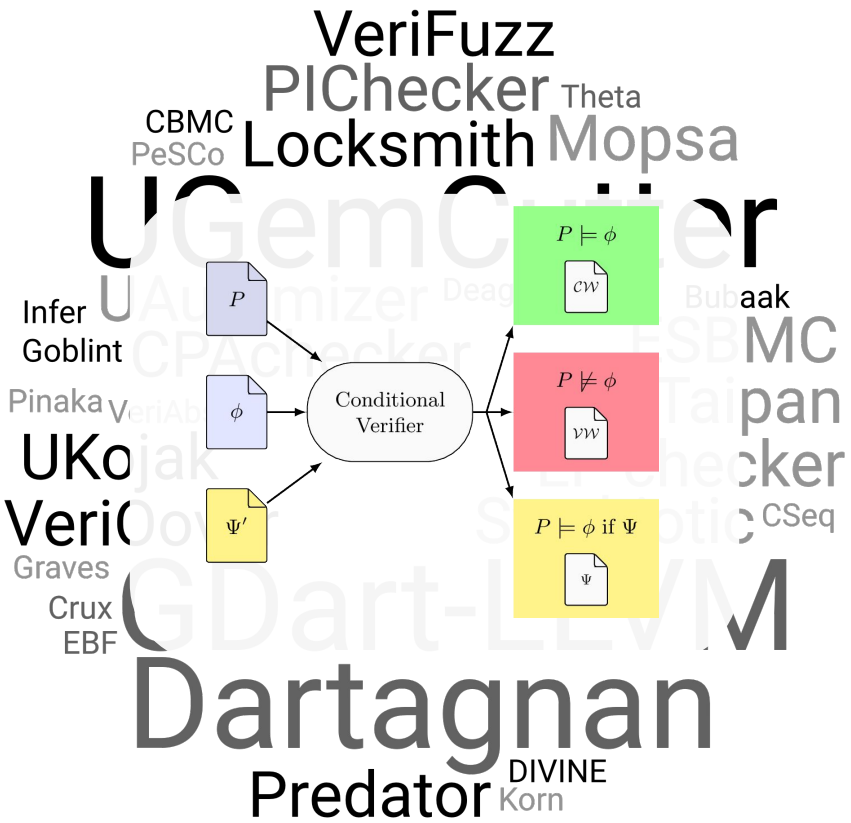
A condition *covers* a program execution if its run leads to an accepting state.



Beyer, Henzinger, Keremoglu, Wendler: Conditional Model Checking: A Technique to Pass Information between Verifiers. *Proc. FSE, 2012.*



Beyer, Henzinger, Keremoglu, Wendler: Conditional Model Checking: A Technique to Pass Information between Verifiers. *Proc. FSE, 2012.*

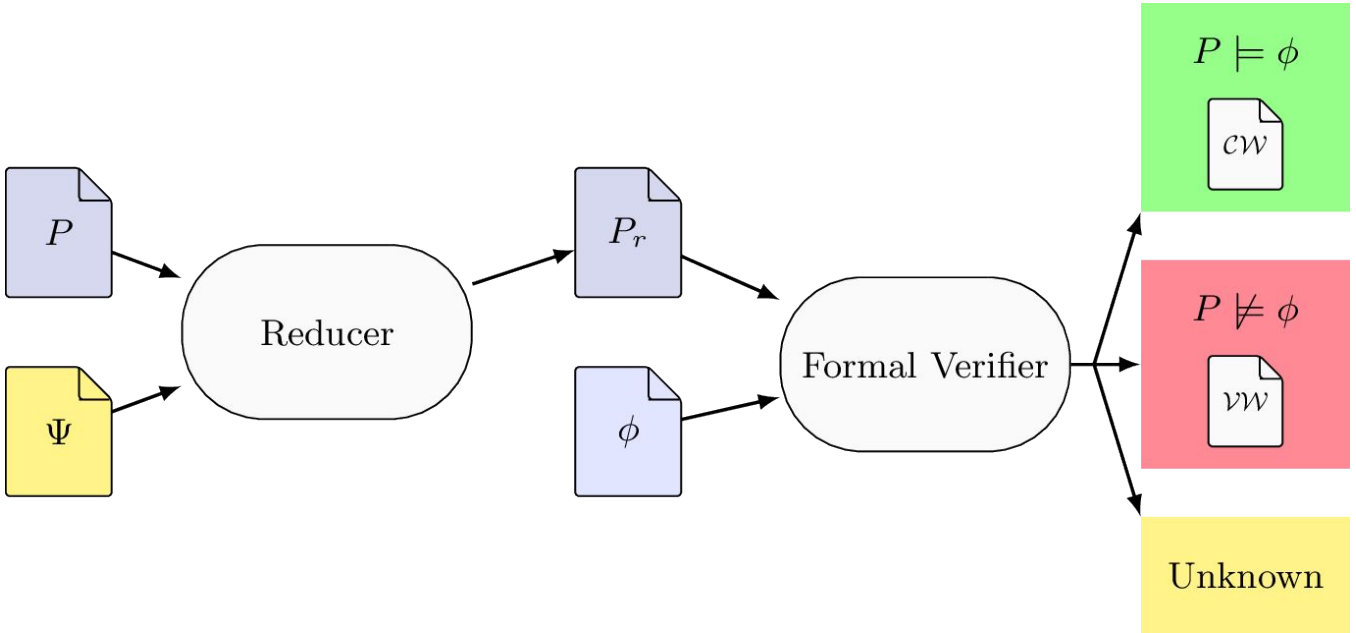


- Conditional Verification is a great idea!
- But there is only one conditional verifier: CPAchecker.

Create providers of conditions?

Create consumers of conditions?

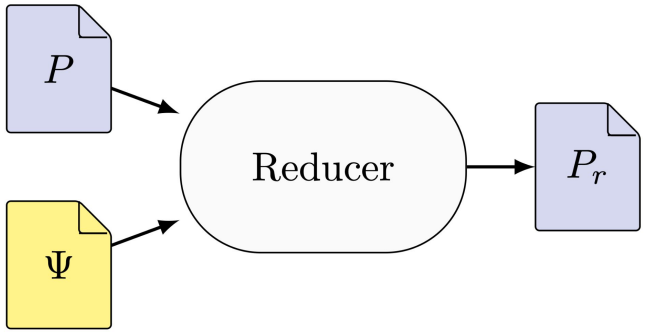
Reducer-Based Construction of Conditional Verifiers



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

A mapping from program *P* and condition \psi to residual program P_r is a reducer, iff:

> *The state space of P_r is a superset of the state space of P that is not covered by \psi.*

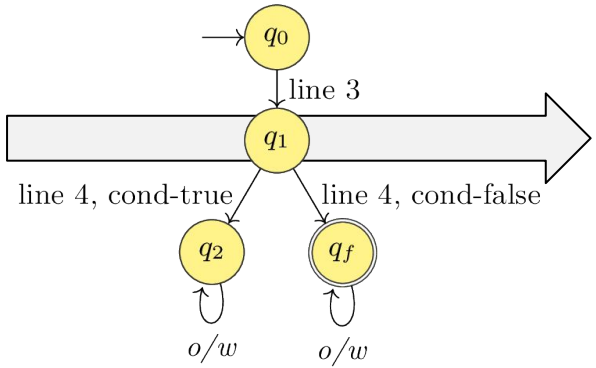


Example Reducers:

- Identity
- Parallel Composition

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

```
 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 }
```




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

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

- Two ideas:
  1. Encode proprietary exchange format in source code
     → Idea transferred to verification witnesses by MetaVal
  2. Make Conditional Model Checking broader applicable
     → Used for Difference Verification

```
 1 int main() {
 2   int i =
__VERIFIER_nondet_int();
 3   if (i != 1017) {
 4     while (i > 1017) {
 5       // branch 1.1
 6       i--;
 7     }
 8     // branch 1.2
 9     // .. snip ..
10   } else {
11     // branch 2
12     // .. snip ..
13   }
14 }
```

Random Tester

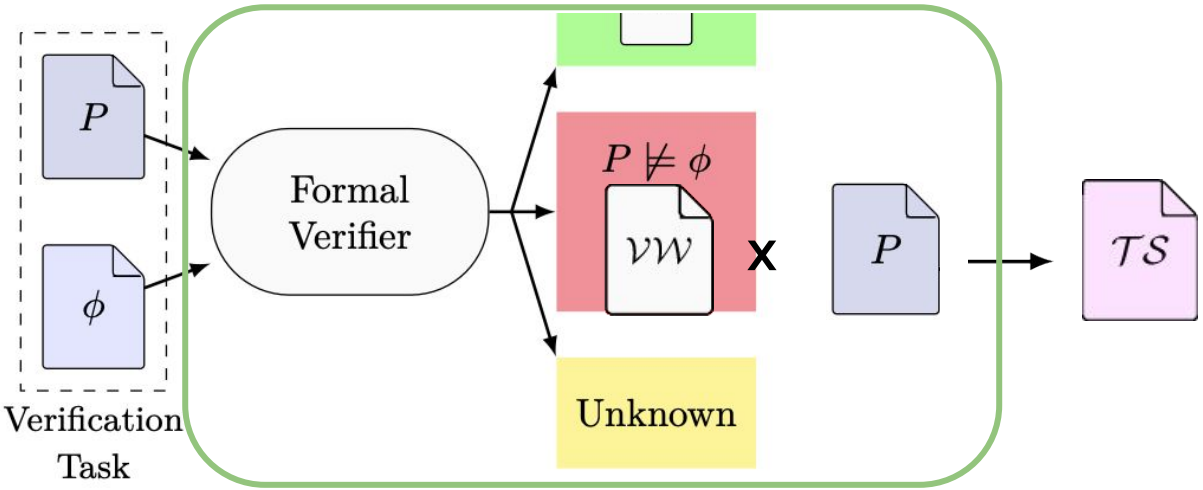
Beyer, Lemberger. **Conditional Testing - Off-the-Shelf Combination of Test-Case Generators.** *Proc. ATVA, 2019.*

```
 1 int main() {
 2   int i =
__VERIFIER_nondet_int();
 3   if (i != 1017) {
 4     while (i > 1017) {
 5       // branch 1.1
 6       i--;
 7     }
 8     // branch 1.2
 9     // .. snip ..
10   } else {
11     // branch 2
12     // .. snip ..
13   }
14 }
```

Symbolic Execution

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

- Exists for proprietary formats[1]
- But we turn any verifier into a test generator[2]



[1]*Beyer, Chlipala, Henzinger, Jhala, Majumdar:* **Generating Tests from Counterexamples.** *Proc. ICSE, 2004.*
Visser, Pasareanu, Kurshid. **Test Input Generation with Java PathFinder.** *Proc. ISSTA, 2004.*
[2]*Beyer, Dangl, Lemberger, Tautschnig.* **Tests from Witnesses: Execution-Based Validation of Verification Results.** *Proc. TAP, 2018.*

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: ***TestCov: Robust Test-Suite Execution and Coverage Measurement****. Proc. ASE, 2019.*

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

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.*