# Cooperative Software Verification

## Combination Approaches that Share Information

Dirk Beyer
LMU Munich, Germany

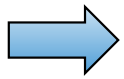
July 7, 2022


# Automatic Software Verification

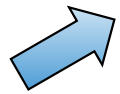
C Program

```
int main() {
    int a = foo();
    int b = bar(a);

    assert(a == b);
}
```



Verification Tool



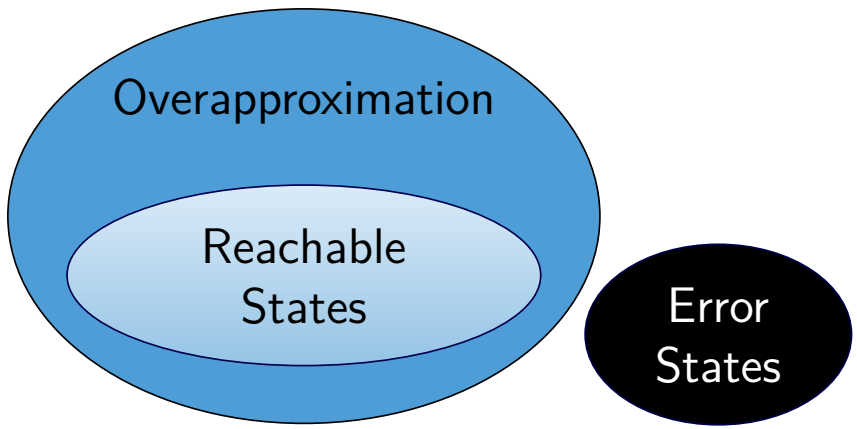
TRUE+witness

i.e., specification is satisfied

FALSE+witness

i.e., bug found

General method:
Create an overapproximation of the program states / compute program invariants



# Two Points

- Open-Source Verification Framework CPAchecker
- Combinations and Cooperation

# CPAchecker History

- 2007: Configurable program analysis [12, 13], CPAchecker was started [17]
- 2009: Large-block encoding [4, FMCAD '09]
- 2010: Adjustable-block encoding [18, FMCAD '10]
- 2012: Conditional model checking [11, FSE '12], PredAbs vs. Impact [23, FMCAD '12]
- 2013: Explicit-state MC [19, FASE '13], BDDs [22, STTT '14], precision reuse [20, FSE '13]
- ...

# CPAchecker— Design Goals

- Unification of several approaches
  $\rightarrow$ reduced to their essential properties
- Allow experimentation with new configurations that we could never think of
- Flexible implementation CPAchecker

# CPAchecker

- Framework for Software Verification
  - Written in Java
  - Open Source: Apache 2.0 License
  - ~80 contributors so far from 15 universities/institutions
  - 470.000 lines of code
    (300.000 without blank lines and comments)
  - Started 2007

https://cpachecker.sosy-lab.org

# CPAchecker: Achievements

- Among world's best software verifiers:
  https://sv-comp.sosy-lab.org/2021/results/
- Continuous success in competition since 2012
  (66 medals: 19x gold, 22x silver, 25x bronze)
- Awarded Gödel medal

  by Kurt Gödel Society


- Used for Linux driver verification
  with dozens of real bugs found and fixed in Linux [29, 21]

# CPAchecker: Concepts

- Included Concepts:
  - CEGAR [24]
  - Interpolation [19, 9]
  - Adjustable-block encoding [18]
  - Conditional model checking [11]
  - Verification witnesses [7, 6]
  - Various abstract domains: predicates, intervals, BDDs, octagons, explicit values
- Available analyses approaches:
  - Predicate abstraction [4, 18, 13, 20]
  - Impact algorithm [30, 23, 9]
  - Bounded model checking [25, 9]
  - k-Induction [8, 9]
  - IC3/Property-directed reachability [5]
  - Explicit-state model checking [19]
  - Interpolation-based model checking

# Status on Verifiers

- From lack of verifiers to plentitude

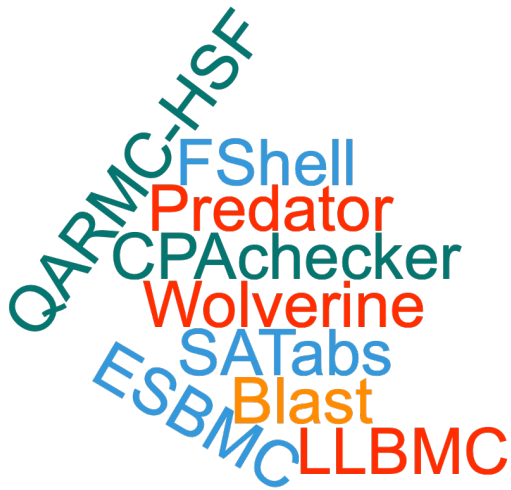
# Competitions in Software Verification and Testing

Mature research area, and there are tool competitions:

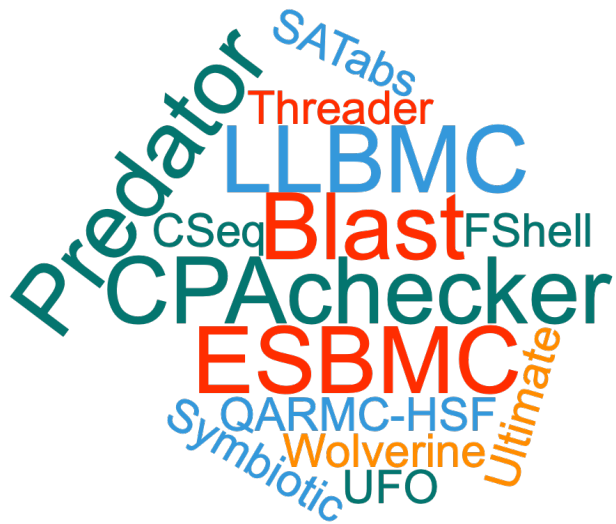
- RERS: off-site, tools, free-style [27]
- SV-COMP: off-site, automatic tools, controlled [1]
- Test-Comp: off-site, automatic tools, controlled [3]
- VerifyThis: on-site, interactive, teams [28]

(alphabetic order)

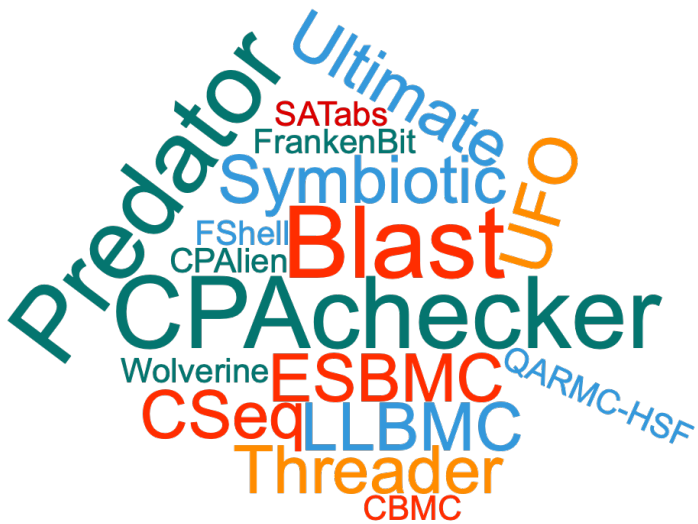
# SV-COMP (Automatic Tools 2012)



# SV-COMP (Automatic Tools 2013, cumulative)

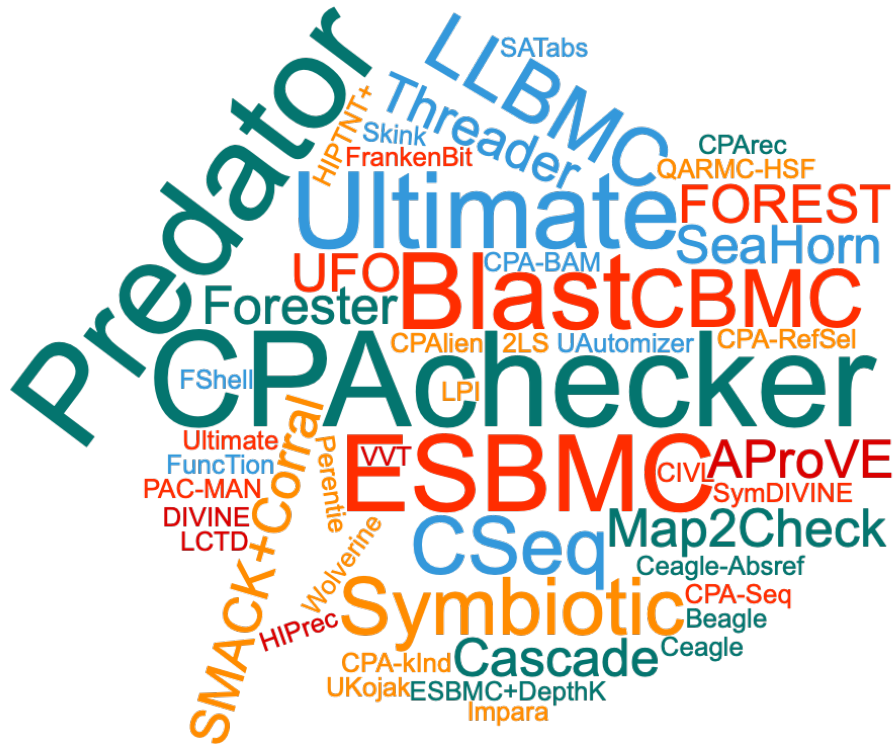


# SV-COMP (Automatic Tools 2014, cumulative)

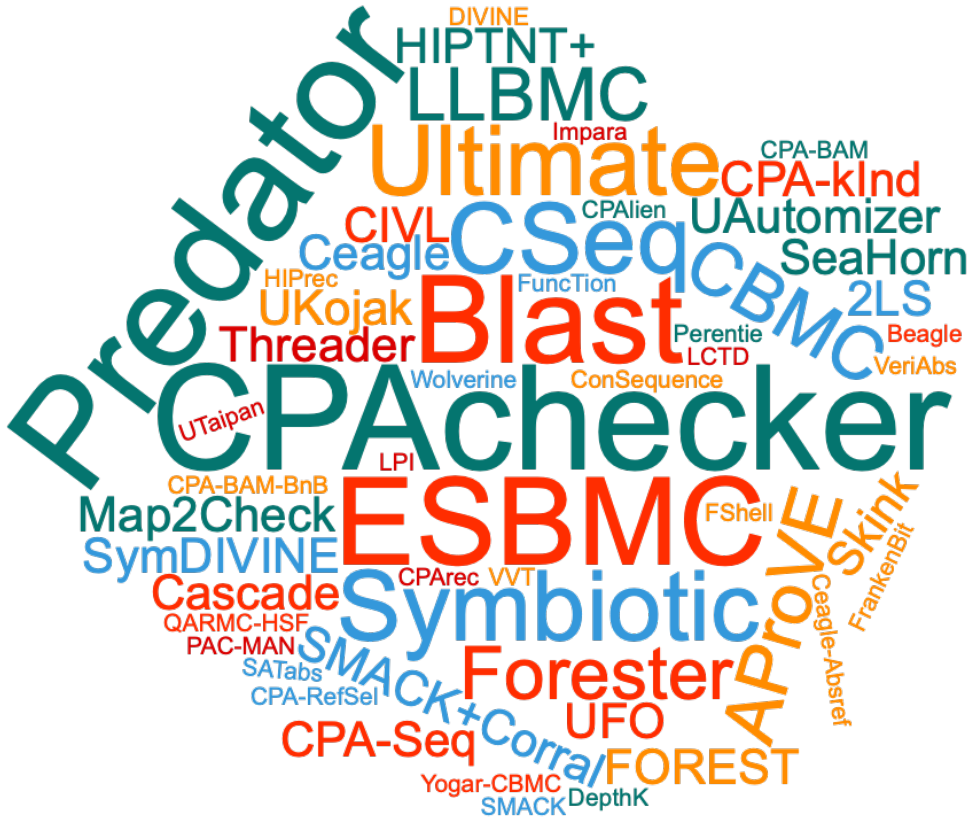


# SV-COMP (Automatic Tools 2015, cumulative)

# SV-COMP (Automatic Tools 2016, cumulative)

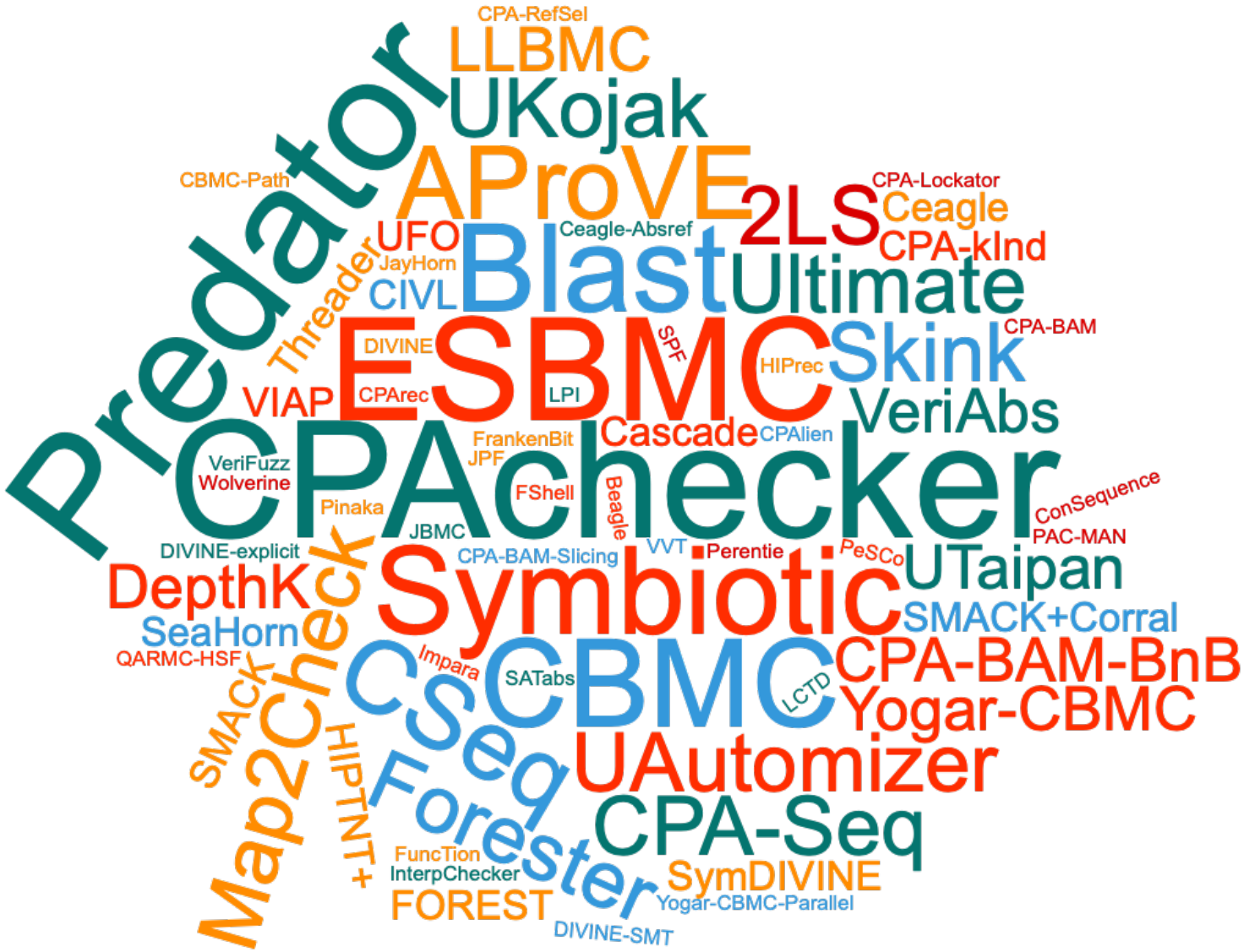


# SV-COMP (Automatic Tools 2017, cumulative)



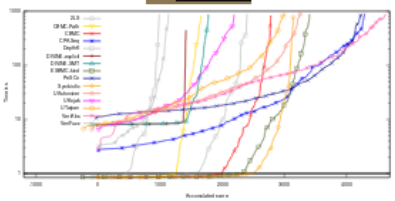
# SV-COMP (Automatic Tools 2018, cumulative)

# SV-COMP (Automatic Tools 2019, cumulative)

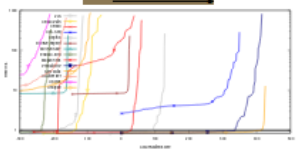


# Different Strengths

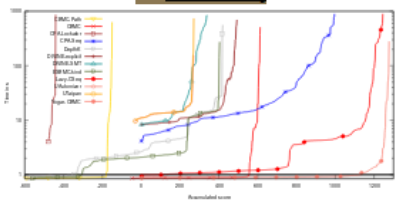
**ReachSafety**
1. VeriAbs
2. CPA-Seq
3. PeSCo



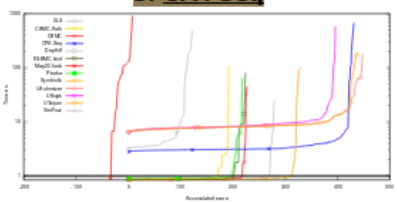
**MemSafety**
1. Symbiotic
2. PredatorHP
3. CPA-Seq



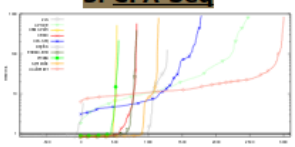
**ConcurrencySafety**
1. Yogar-CBMC
2. Lazy-CSeq
3. CPA-Seq



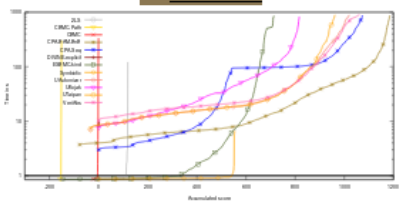
**NoOverflows**
1. UAutomizer
2. UTaipan
3. CPA-Seq



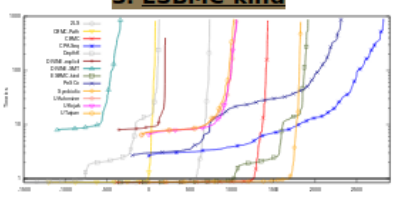
**Termination**
1. UAutomizer
2. AProVE
3. CPA-Seq



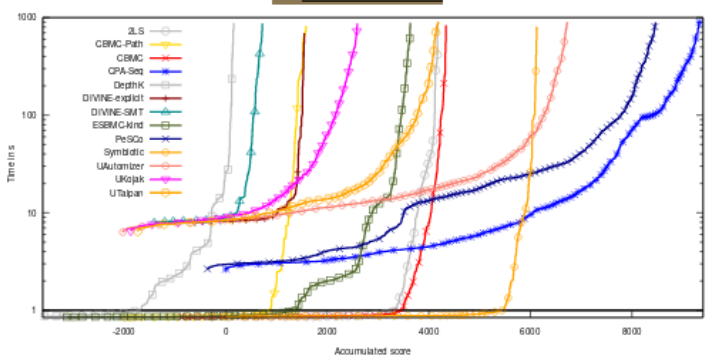
**SoftwareSystems**
1. CPA-BAM-BnB
2. CPA-Seq
3. VeriAbs



**FalsificationOverall**
1. CPA-Seq
2. PeSCo
3. ESBMC-kind



**Overall**
1. CPA-Seq
2. PeSCo
3. UAutomizer




https://sv-comp.sosy-lab.org/2019/results

# Different Techniques

| Participant | CEGAR | Predicate Abstraction | Symbolic Execution | Bounded Model Checking | k-Induction | Property-Directed Reach. | Explicit-Value Analysis | Numeric. Interval Analysis | Shape Analysis | Separation Logic | Bit-Precise Analysis | ARG-Based Analysis | Lazy Abstraction | Interpolation | Automata-Based Analysis | Concurrency Support | Ranking Functions | Evolutionary Algorithms |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2LS | | | | ✓ | ✓ | | | ✓ | | | ✓ | | | | | | ✓ | |
| APROVE | | | ✓ | | | | ✓ | ✓ | | ✓ | ✓ | | | | | | ✓ | |
| CBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| CBMC-PATH | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| CPA-BAM-BNB | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | | | |
| CPA-LOCKATOR | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | | |
| CPA-SEQ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | |
| DEPTHK | | | | ✓ | ✓ | | | | | | ✓ | | | | | ✓ | | |
| DIVINE-EXPLICIT | | | | | | | ✓ | | | | ✓ | | | | | ✓ | | |
| DIVINE-SMT | | | | | | | ✓ | | | | ✓ | | | | | ✓ | | |
| ESBMC-KIND | | | | ✓ | ✓ | | | | | | ✓ | | | | | ✓ | | |
| JAYHORN | ✓ | ✓ | | | | ✓ | | ✓ | | | | | ✓ | ✓ | | | | |
| JBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| JPF | | | | ✓ | | | ✓ | ✓ | | | ✓ | | | | | ✓ | | |
| LAZY-CSEQ | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| MAP2CHECK | | | | ✓ | | | | | | | ✓ | | | | | | | |
| PESCO | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | |
| PINAKA | | | ✓ | ✓ | | | | | | | ✓ | | | | | | | |
| PREDATORHP | | | | | | | | | ✓ | | | | | | | | | |
| SKINK | ✓ | | | | | | ✓ | | | | | | | ✓ | ✓ | | | |
| SMACK | ✓ | | | ✓ | | ✓ | | | | | ✓ | | ✓ | | | ✓ | | |
| SPF | | | ✓ | | | | | | ✓ | | | | | | | ✓ | | |
| SYMBIOTIC | | | ✓ | | | | | ✓ | | | ✓ | | | | | | | |
| UAUTOMIZER | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | | ✓ | |
| UKOJAK | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | | | | |
| UTAIPAN | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | |

Competition Report [2]

https://doi.org/10.1007/978-3-030-17502-3_9

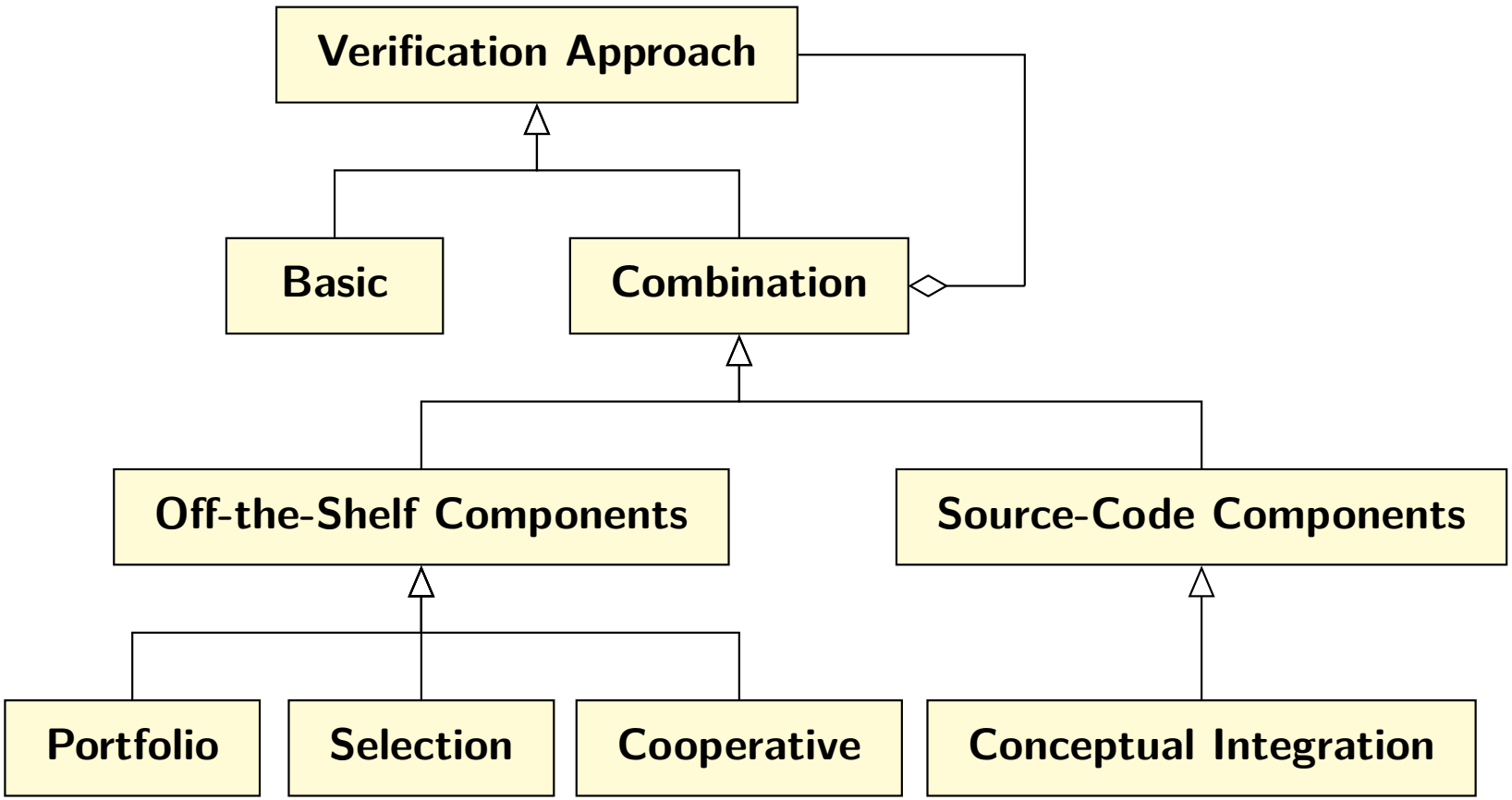
# Insights from Software Model Checking

- Verifiers have different strengths
- There are plenty of tools
- $\Rightarrow$ Cooperative Verification Approaches

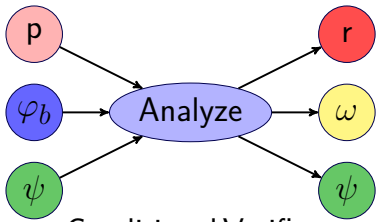
# Cooperative Verification — Think big!

- Introduce a new level!
- Current tools are "low level" components (engines)
- Construct combinations
- Clear Interfaces
  via, e.g., Conditions, Witnesses, Test Suites
- Success: SAT, SMT
- See also: Little Engines [31], Evidential Tool Bus [26]

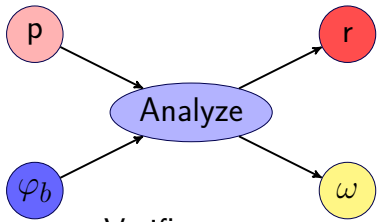
# Approaches for Combinations



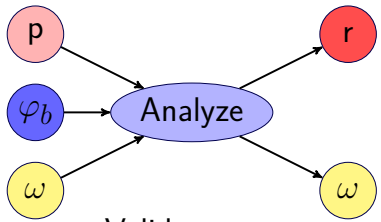
# Graphical Visualization of the Coop Framework



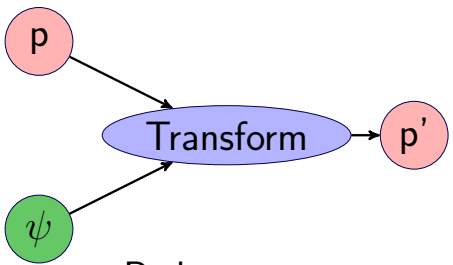

Conditional Verifier



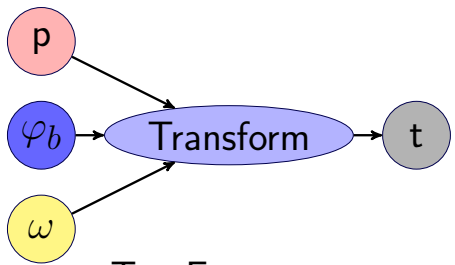

Verifier



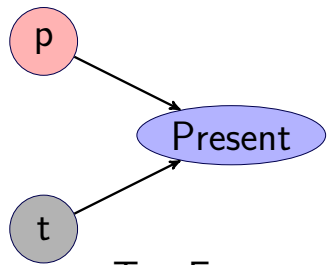

Validator



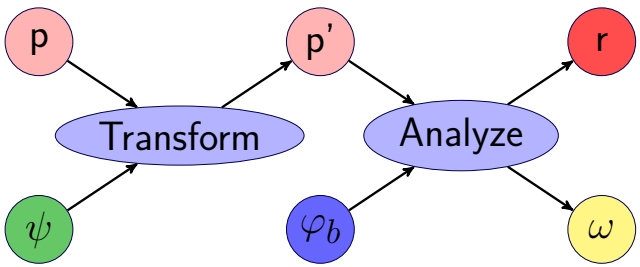

Reducer



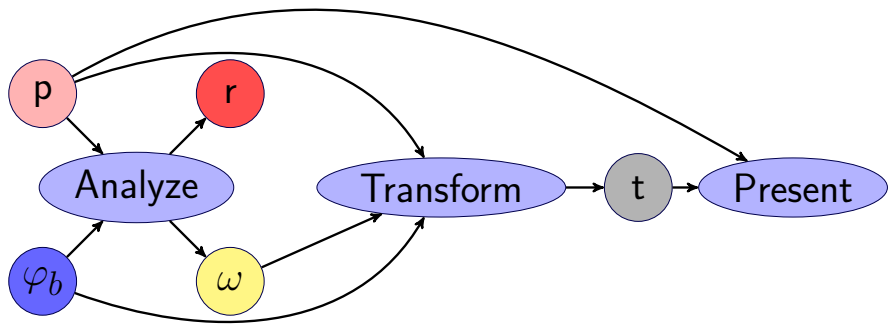

Test Extractor




Test Executor




Conditional Verifier constructed from Reducer and Verifier




Execution-Based Validation constructed from Verifier, Test Extractor, and Test Executor

# Definition of Cooperative Verification

An approach is called **cooperative verification**, if

- identifiable *actors* pass information in form of
- identifiable *artifacts* towards the common objective of
- solving a *verification* problem,

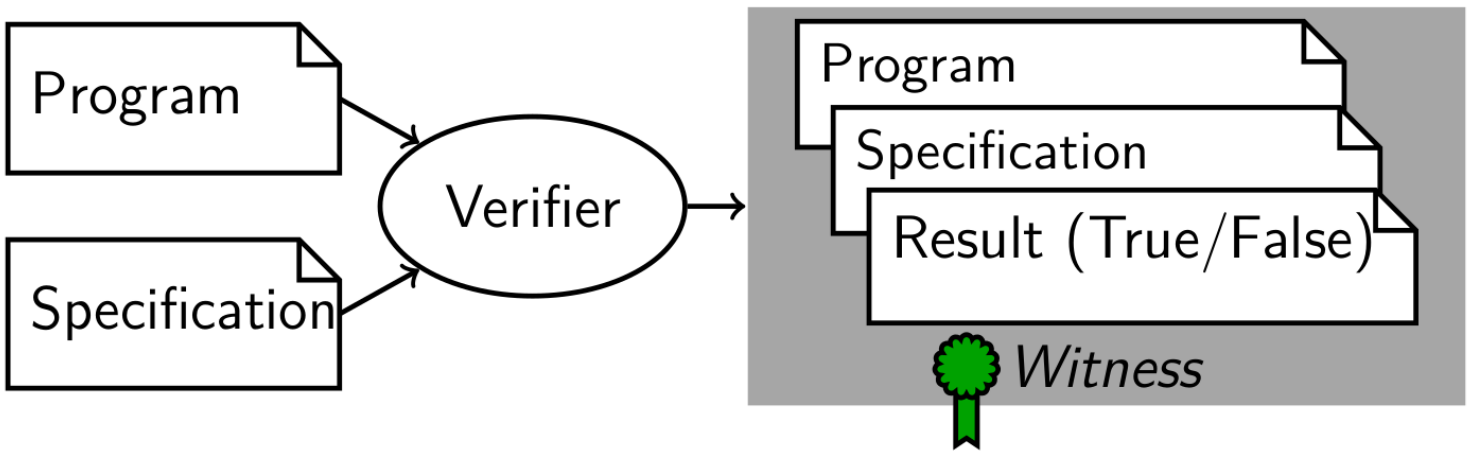
where at least two of these actors are *analyzers*.

# Definition of Cooperative Verification

Examples for notions:

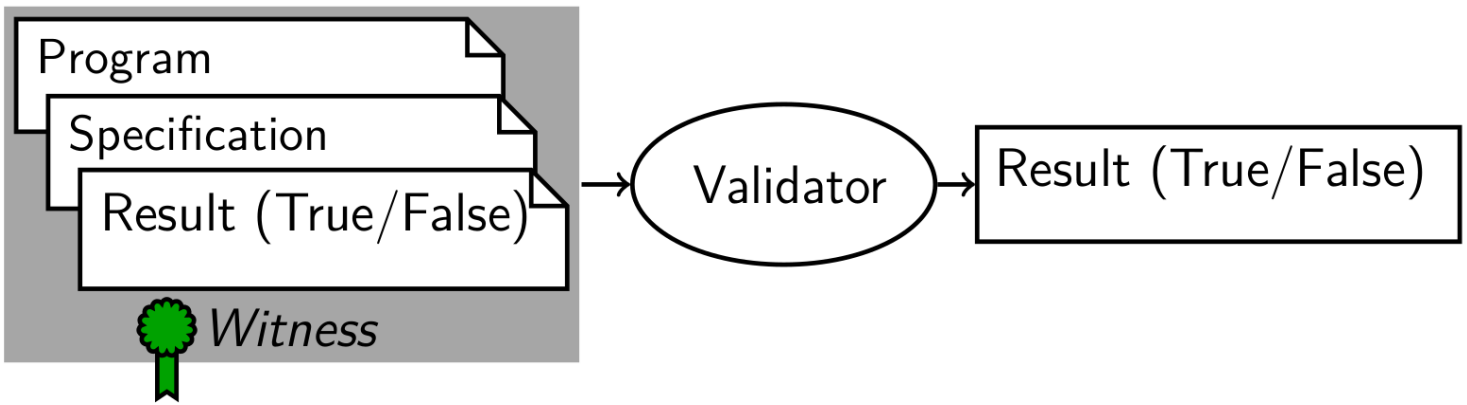
- Identifiable actor:
  off-the-shelf components, binaries, agents, web services
- Identifiable artifacts:
  programs, witnesses, ARGs, test suites
- Verification problem:
  verification task, test task, feasibility check, refinement

# Software Verification with Witnesses




[7, Proc. FSE 2015] [6, Proc. FSE 2016]

# Witness Validation




- Validate untrusted results
- Easier than full verification

# Example Combination (in DSL CoVeriTeam)

CoVeriTeam: Language and Tool [15, Proc. TACAS 2022]

**Algorithm 1** Witness Validation [7, 6]

**Input:** Program p, Specification s
**Output:** Verdict

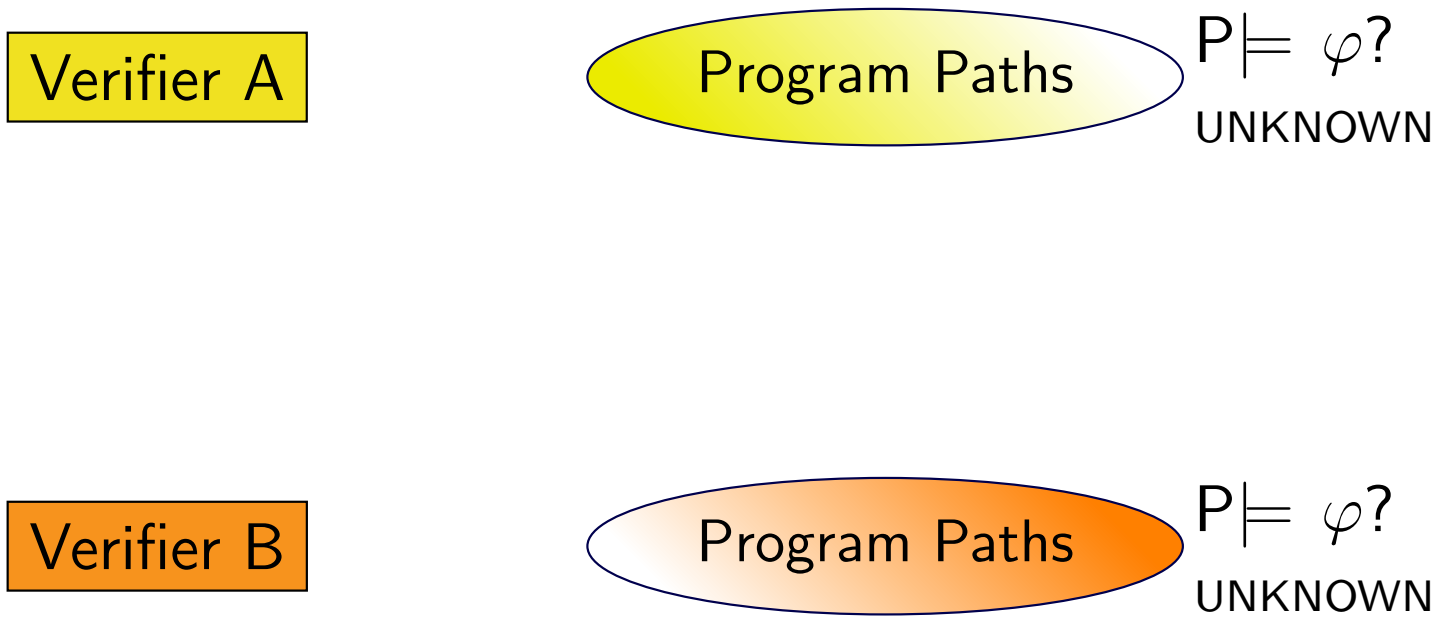
```
1: verifier := Verifier("Ultimate Automizer")
2: validator := Validator("CPAchecker")
3: result := verifier.verify(p, s)
4: if result.verdict ∈ {TRUE, FALSE} then
5:     result = validator.validate (p, s, result.witness)
6: return  (result.verdict, result.witness)
```

# Simple Combination without Cooperation

Often, even simple combinations help!
Portfolio construction using off-the-shelf verification tools [16, Proc. FASE 2022]
Consider AWS category (177 tasks) in SV-COMP 2022:
CBMC: 69 (8 wrong)
CoVeriTeam-Parallel-Portfolio: 147 (3 wrong)
(improvement did not require any change in a verification tool)

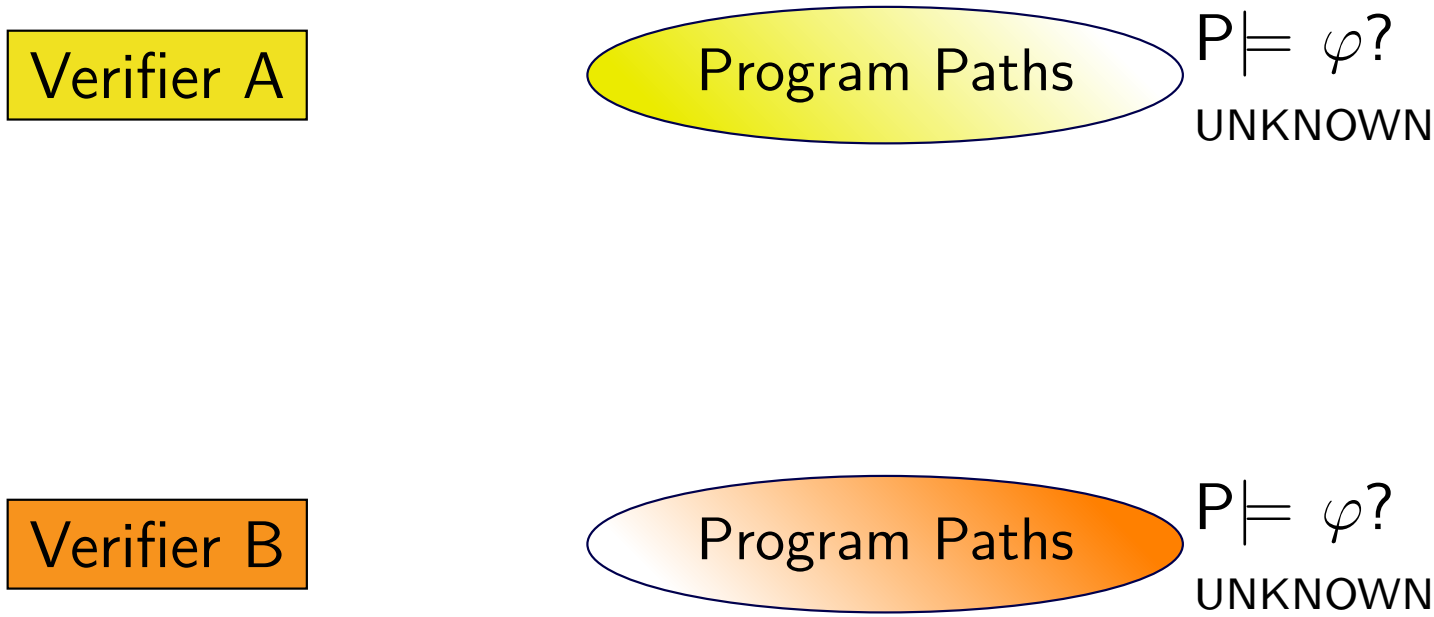
# Facing Hard Verification Tasks

Given: Program P$\models \varphi$?

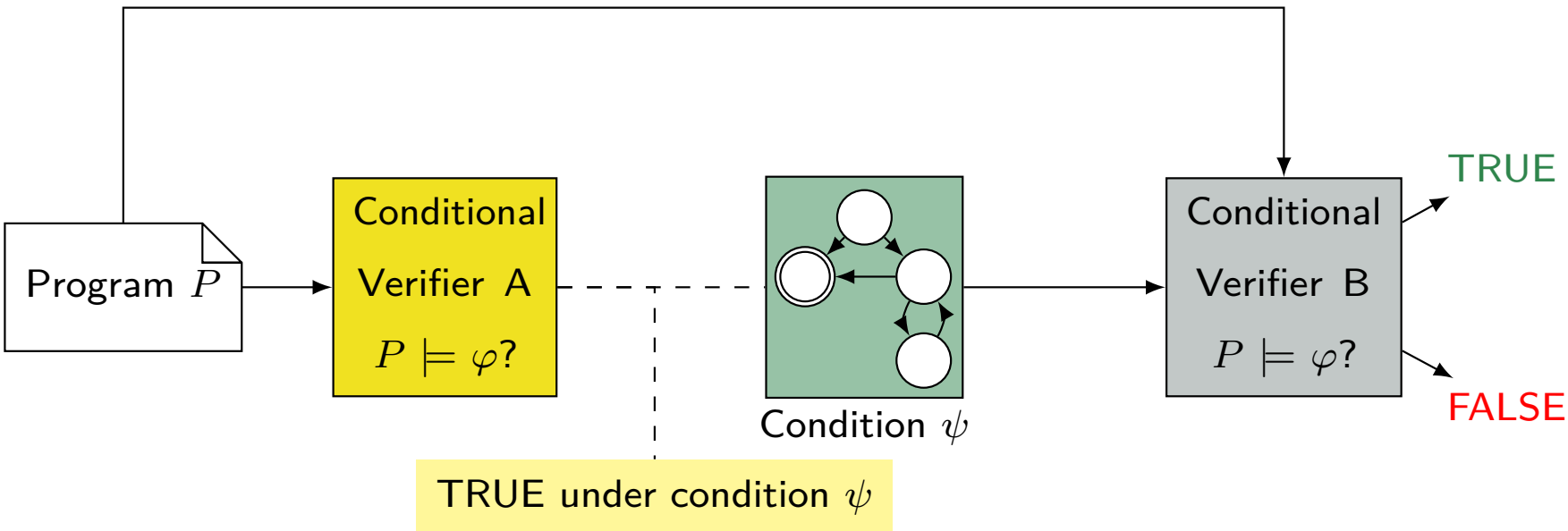


# Facing Hard Verification Tasks

Given: Program P$\models \varphi$?

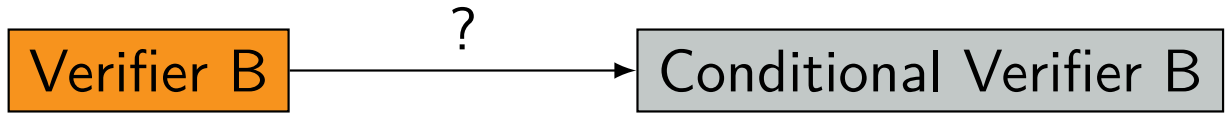


# Conditional Model Checking



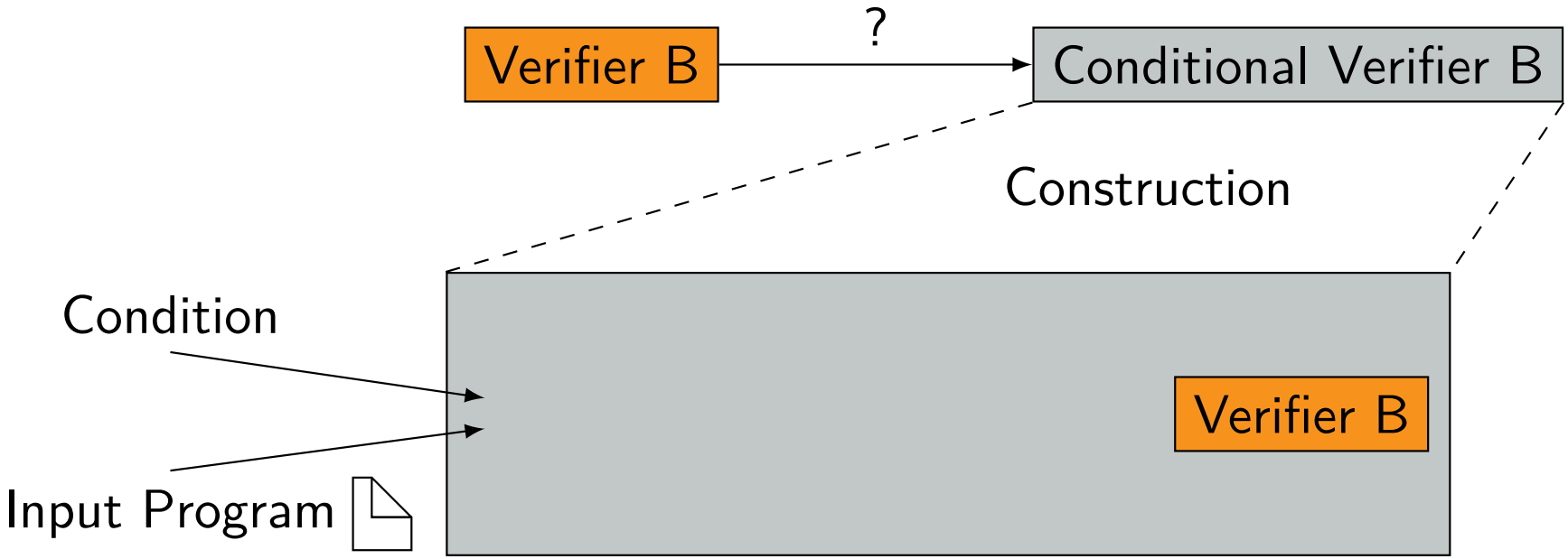

Proc. FSE 2012 [11]

# Reducer-Based Construction



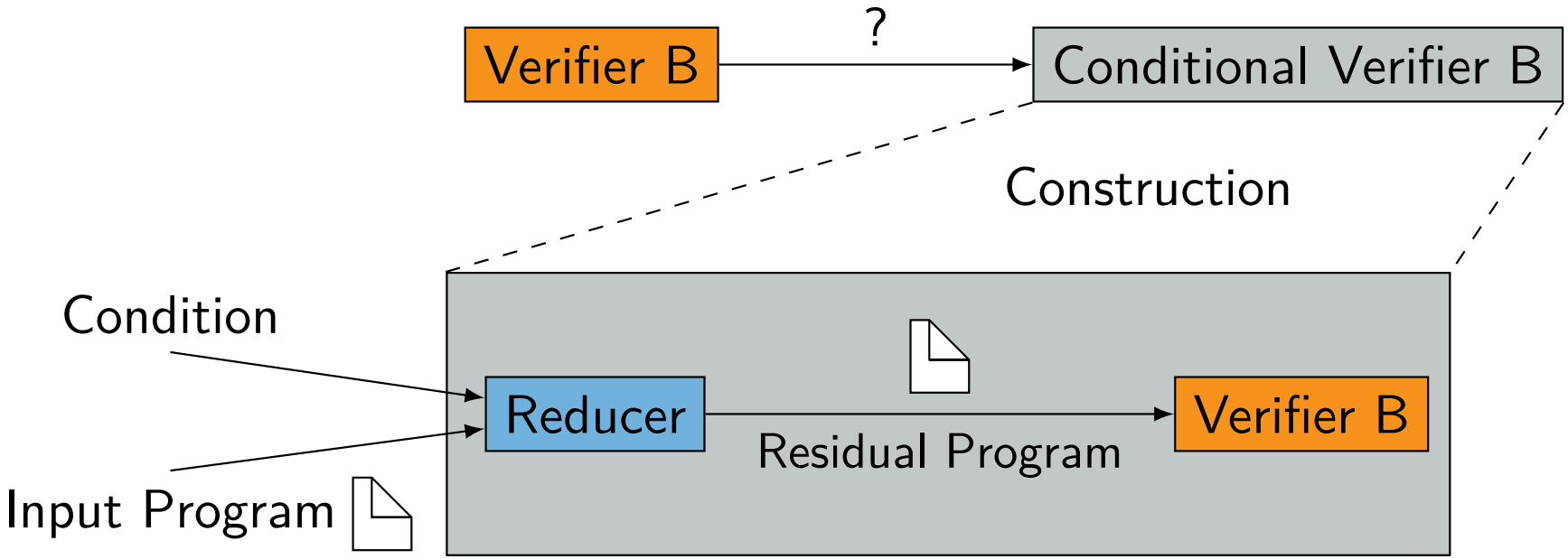

Proc. ICSE 2018 [14]

# Reducer-Based Construction




Proc. ICSE 2018 [14]

# Reducer-Based Construction

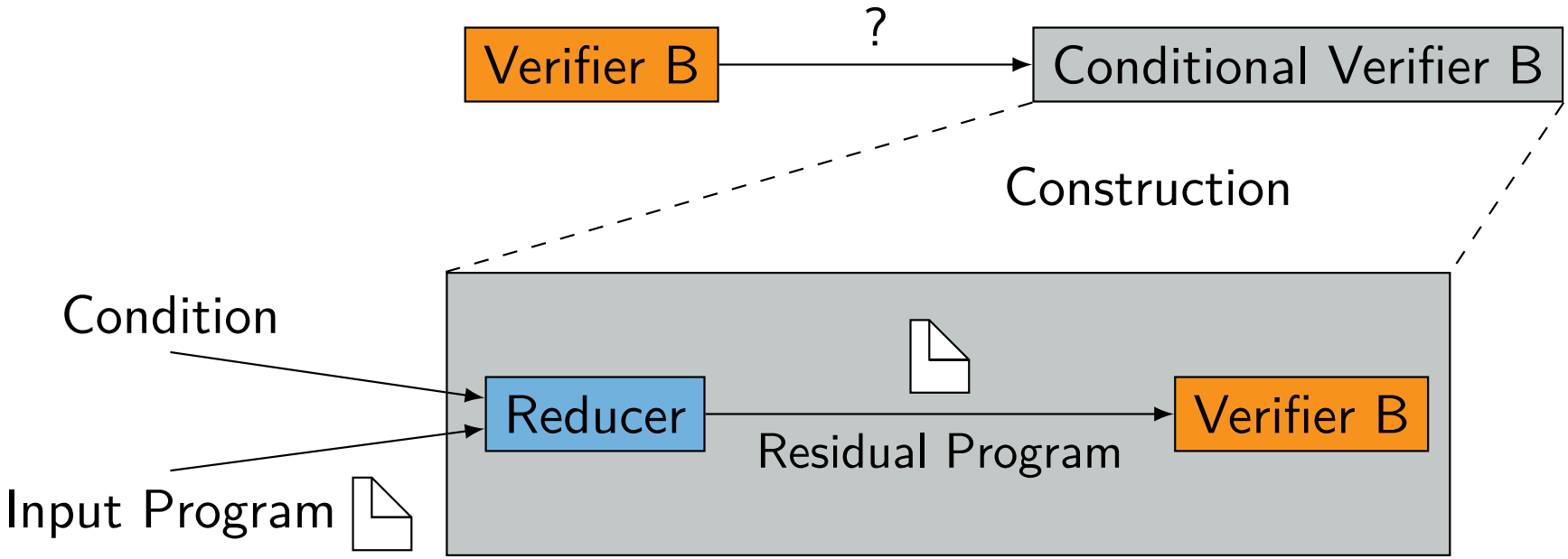



Reducer (preprocessor)

- Builds standard input (C program)
- Representing a subset of paths
- Contains at least all non-verified paths

Proc. ICSE 2018 [14]

# Reducer-Based Construction




Reducer (preprocessor)

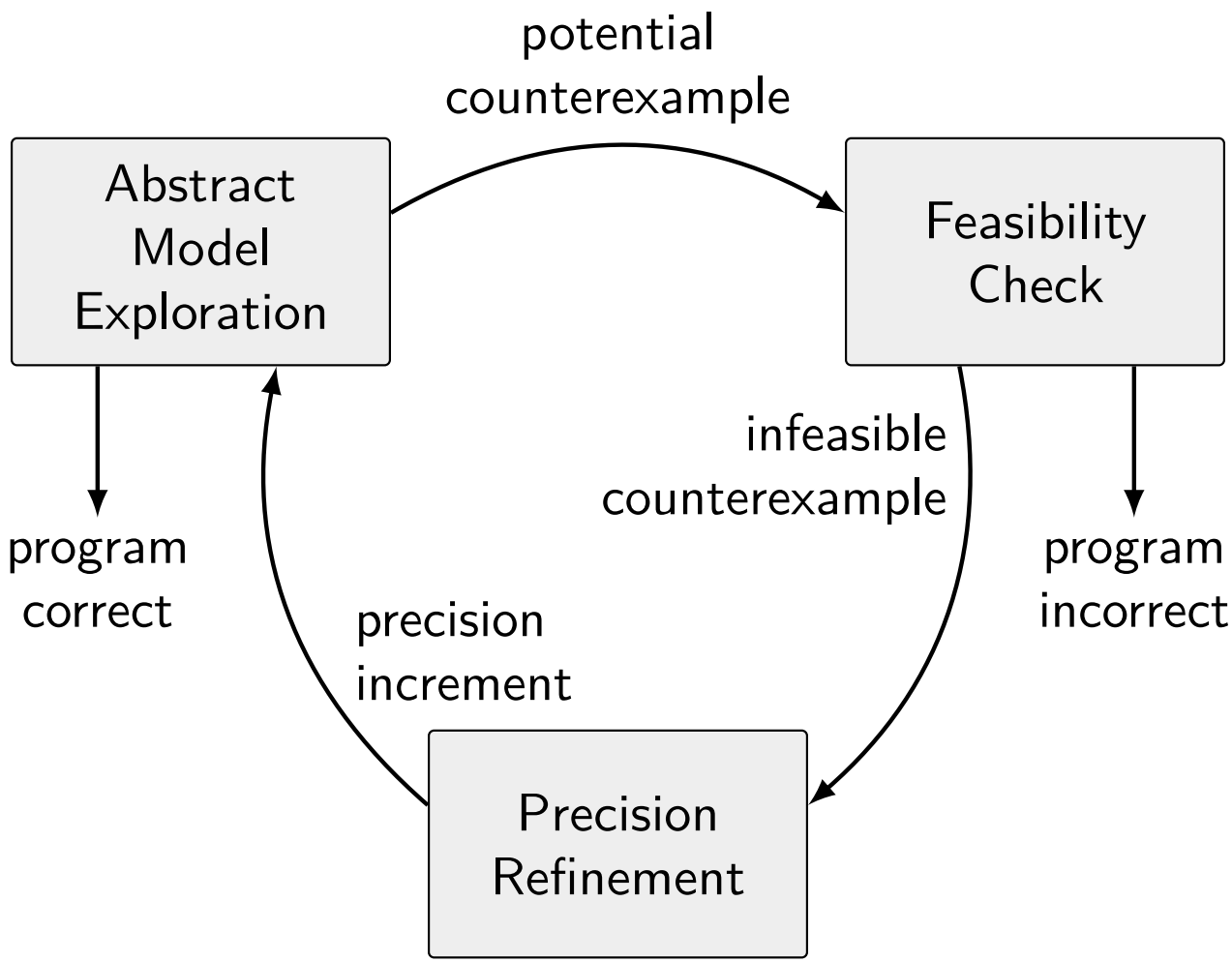
- Builds standard input (C program)
- Representing a subset of paths
- Contains at least all non-verified paths

\+ Verifier-unspecific approach

\+ Many conditional verifiers possible

Proc. ICSE 2018 [14]

# CEGAR

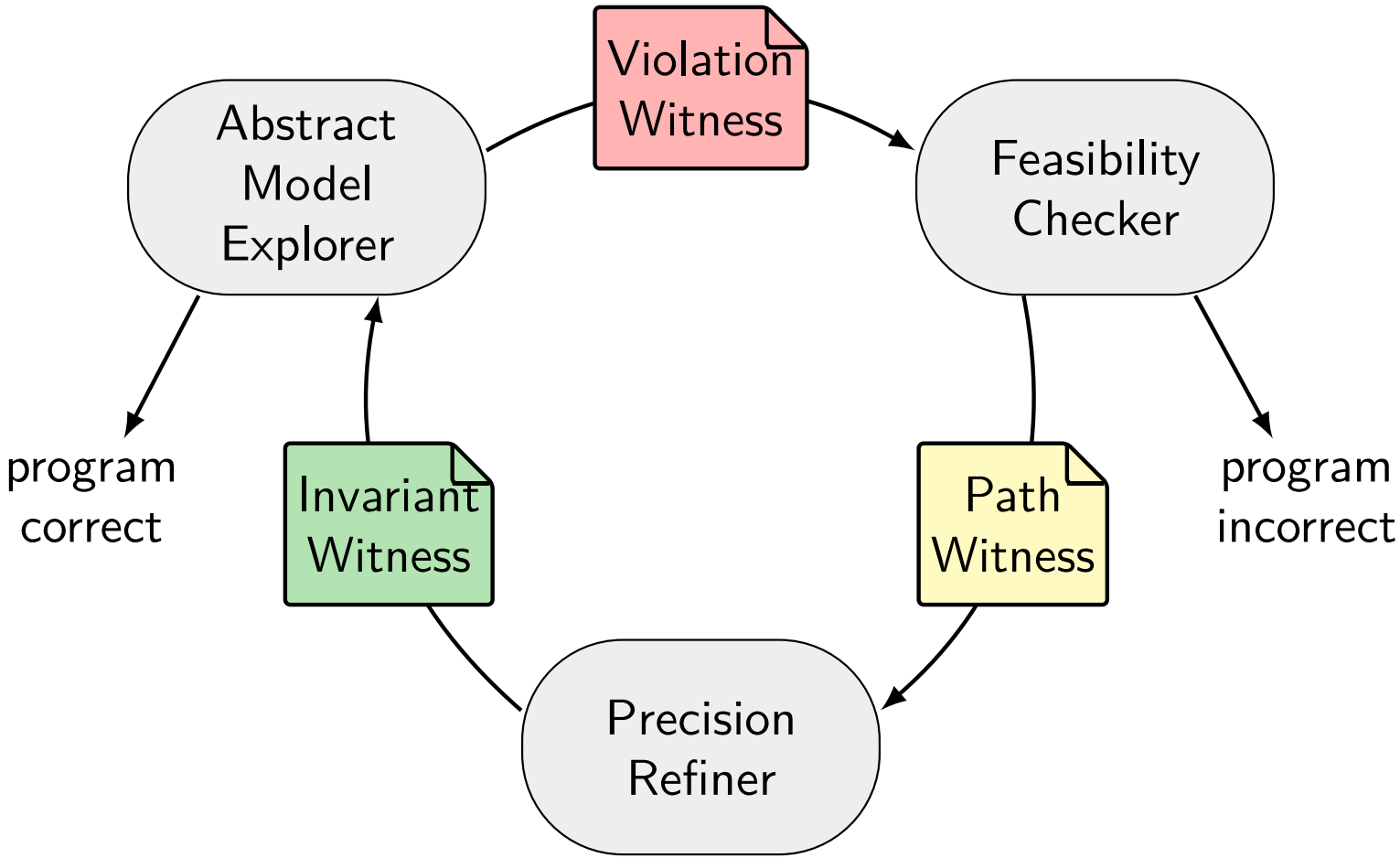

potential
counterexample
Abstract
Model
Exploration
Feasibility
Check
infeasible
counterexample
program
correct
program
incorrect
precision
increment
Precision
Refinement

# Modularization of CEGAR

- CEGARdefines I/O interfaces
- But instances not exchangeable
- Aim: generalize CEGAR, allow exchange of components

$\Rightarrow$ Modular reformulation

# Workflow of modular CEGAR




Proc. ICSE 2022 [10]

# Conclusion

- CPAchecker as framework
- Combinations and Cooperation

# References I


[1] Beyer, D.: Competition on software verification (SV-COMP). In: Proc. TACAS. pp. 504–524. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_38

[2] Beyer, D.: Automatic verification of C and Java programs: SV-COMP 2019. In: Proc. TACAS (3). pp. 133–155. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_9

[3] Beyer, D.: Competition on software testing (Test-Comp). In: Proc. TACAS (3). pp. 167–175. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_11

[4] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147

[5] Beyer, D., Dangl, M.: Software verification with PDR: An implementation of the state of the art. In: Proc. TACAS (1). pp. 3–21. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_1

[6] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: Proc. FSE. pp. 326–337. ACM (2016). https://doi.org/10.1145/2950290.2950351

# References II


[7] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A.: Witness validation and stepwise testification across software verifiers. In: Proc. FSE. pp. 721–733. ACM (2015). https://doi.org/10.1145/2786805.2786867

[8] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_42

[9] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning **60**(3), 299–335 (2018). https://doi.org/10.1007/s10817-017-9432-6

[10] Beyer, D., Haltermann, J., Lemberger, T., Wehrheim, H.: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. In: Proc. ICSE. ACM (2022)

[11] Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Proc. FSE. ACM (2012). https://doi.org/10.1145/2393596.2393664

[12] Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable software verification: Concretizing the convergence of model checking and program analysis. In: Proc. CAV. pp. 504–518. LNCS 4590, Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_51

# References III


[13] Beyer, D., Henzinger, T.A., Théoduloz, G.: Program analysis with dynamic precision adjustment. In: Proc. ASE. pp. 29–38. IEEE (2008). https://doi.org/10.1109/ASE.2008.13

[14] Beyer, D., Jakobs, M.C., Lemberger, T., Wehrheim, H.: Reducer-based construction of conditional verifiers. In: Proc. ICSE. pp. 1182–1193. ACM (2018). https://doi.org/10.1145/3180155.3180259

[15] Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31

[16] Beyer, D., Kanav, S., Richter, C.: Construction of Verifier Combinations Based on Off-the-Shelf Verifiers. In: Proc. FASE. pp. 49–70. Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_3

[17] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16

[18] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010)

# References IV


[19] Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Proc. FASE. pp. 146–162. LNCS 7793, Springer (2013). https://doi.org/10.1007/978-3-642-37057-1_11

[20] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proc. FSE. pp. 389–399. ACM (2013). https://doi.org/10.1145/2491411.2491429

[21] Beyer, D., Petrenko, A.K.: Linux driver verification. In: Proc. ISoLA. pp. 1–6. LNCS 7610, Springer (2012). https://doi.org/10.1007/978-3-642-34032-1_1

[22] Beyer, D., Stahlbauer, A.: BDD-based software verification: Applications to event-condition-action systems. Int. J. Softw. Tools Technol. Transfer **16**(5), 507–518 (2014). https://doi.org/10.1007/s10009-014-0334-1

[23] Beyer, D., Wendler, P.: Algorithms for software model checking: Predicate abstraction vs. IMPACT. In: Proc. FMCAD. pp. 106–113. FMCAD (2012)

[24] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

# References V


[25] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15

[26] Cruanes, S., Hamon, G., Owre, S., Shankar, N.: Tool integration with the Evidential Tool Bus. In: Proc. VMCAI. pp. 275–294. LNCS 7737, Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_18

[27] Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS grey-box challenge 2012: Analysis of event-condition-action systems. In: Proc. ISoLA. pp. 608–614. LNCS 7609, Springer (2012). https://doi.org/10.1007/978-3-642-34026-0_45

[28] Huisman, M., Klebanov, V., Monahan, R.: VerifyThis 2012: A program verification competition. STTT **17**(6), 647–657 (2015). https://doi.org/10.1007/s10009-015-0396-8

[29] Khoroshilov, A.V., Mutilin, V.S., Petrenko, A.K., Zakharov, V.: Establishing Linux driver verification process. In: Proc. Ershov Memorial Conference. pp. 165–176. LNCS 5947, Springer (2009). https://doi.org/10.1007/978-3-642-11486-1_14

[30] McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. CAV. pp. 123–136. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_14

# References VI

[31] Shankar, N.: Little engines of proof. In: Proc. FME. pp. 1–20. LNCS 2391, Springer (2002). https://doi.org/10.1007/3-540-45614-7_1