Cooperative Software Verification

Combination Approaches that Share Information

Dirk Beyer
LMU Munich, Germany

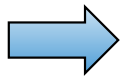
August 16, 2022, at Manchester University

Automatic Software Verification

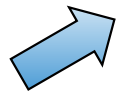
C Program

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

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



Verification Tool



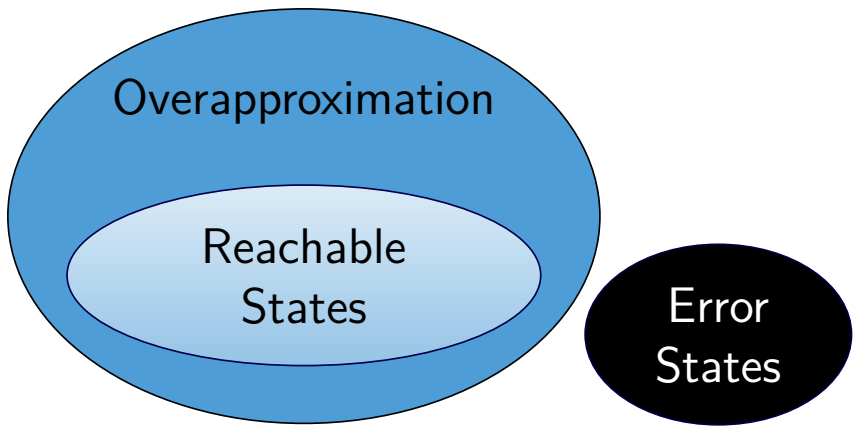
TRUE+witness

i.e., specification is satisfied

FALSE+witness

i.e., bug found

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



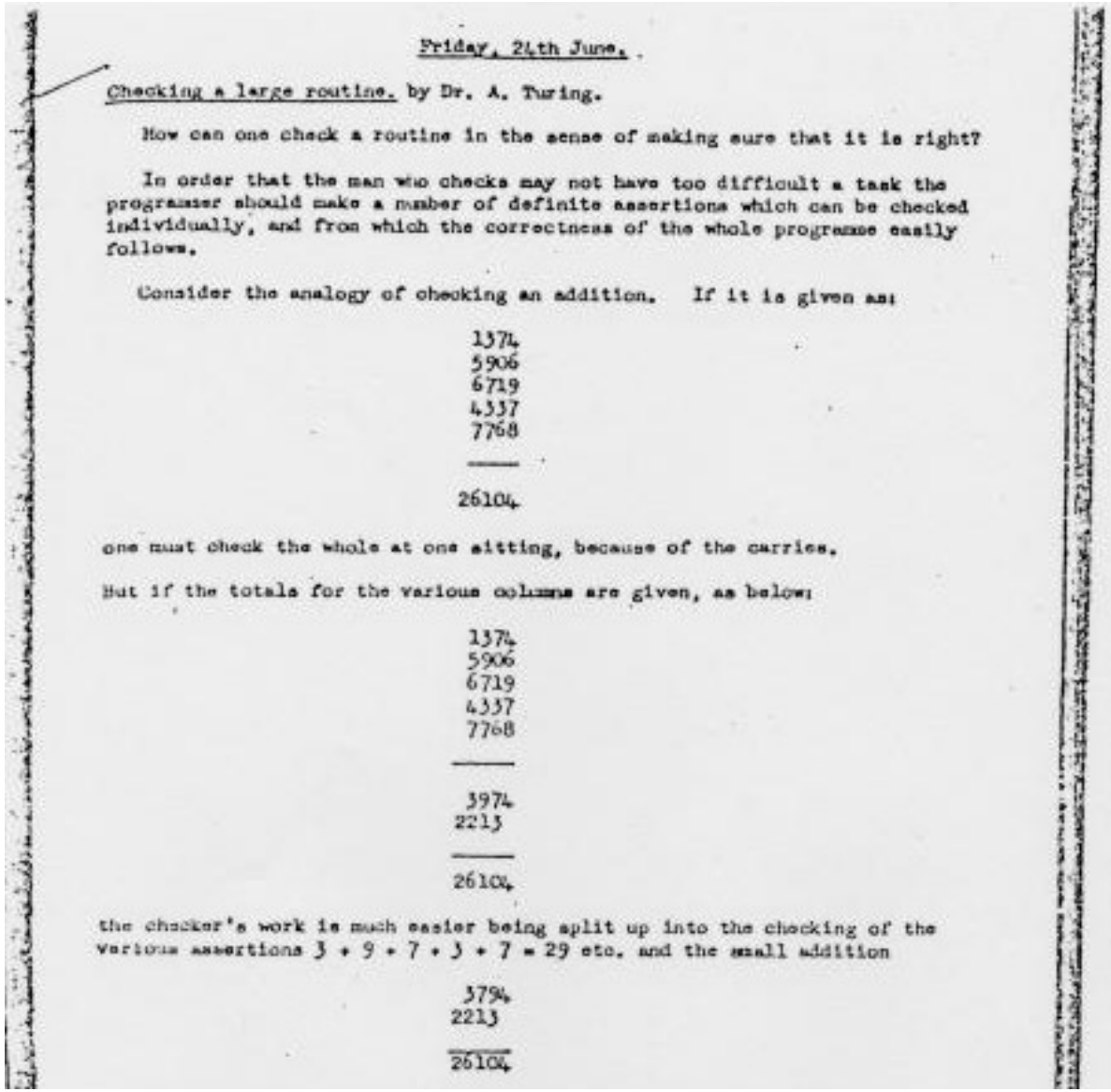
Two Points

- There is quite some history of software verification.
- We need more combinations and cooperation.

Some Historical Landmarks

- **70 years ago: Assertions and Proof Decomposition**, Alan Turing, 1949 [42]
  "In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily follows."

Landmarks, Alan Turing, 1949 [42]



Friday, 24th June.

Checking a large routine. by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
------
26104
```

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
 1374
 5906
 6719
 4337
 7768
------

 3974
 2213
------
26104
```

the checker's work is much easier being split up into the checking of the various assertions 3 + 9 + 7 + 3 + 7 = 29 etc. and the small addition

```
 3974
 2213
------
26104
```

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- **60 years ago: Logic Interpolation**,
  William Craig, J. Symb. Log. 1957 [32]
  "Linear reasoning. A new form of the Herbrand-Gentzen theorem."
  Defines an interpolation for logic formulas
  Made popular by Ken McMillan [39]

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- **50 years ago: Data-Flow Analysis and Abstract States**,
  Gary Kildall, POPL 1973 [38]
  "A Unified Approach to Global Program Optimization"
  Defines algorithm, meet operation, lattice, etc.
  Extended and made popular for program analysis by
  F. Nielson, H. R. Nielson, C. Hankin, P. Cousot, R. Cousot

Landmarks, Gary Kildall, POPL 1973 [38]

A UNIFIED APPROACH TO GLOBAL
PROGRAM OPTIMIZATION

Gary A. Kildall

Computer Science Group
Naval Postgraduate School
Monterey, California

Abstract

A technique is presented for global analysis of program structure in order to perform compile time optimization of object code generated for expressions. The global expression optimization presented includes constant propagation, common subexpression elimination, elimination of redundant register load operations, and live expression analysis. A general purpose program flow analysis algorithm is developed which depends upon the existence of an "optimizing function." The algorithm is defined formally using a directed graph model of program flow structure, and is shown to be correct. Several optimizing functions are defined which, when used in conjunction with the flow analysis algorithm, provide the various forms of code optimization. The flow analysis algorithm is sufficiently general that additional functions can easily be defined for other forms of global code optimization.

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- **40 years ago: LTL and Model Checking**,
  Pnueli, Clarke, Emerson, Sifakis, 1981
  Specification languages, modeling languages, algorithms, theory, tools
  LTL, CTL, automata, Kripke structures, model checking,
  $\rightarrow$ software model checking

"Handbook of Model Checking", Springer, 2018 [28]

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- **25 years ago: Predicate Abstraction**,
  Graf, Saïdi, 1997 [34]
  Enabling idea to project software to
  a (smaller) finite state space

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- **20 years ago: Tools for Software Model Checking**

  "1st generation" of tools:
  - Summer 2000: SLAM [2, 1]
  - Fall 2000: BLAST [35, 14]
  - 2004: SATABS [30]

  SLAM paper received Test-of-Time Award from PLDI, first to apply predicate abstraction + CEGAR to software. BLAST received gold medals in competitions.

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- 20 years ago: Tools for Software Model Checking
- **15 years ago: Satisfiability Modulo Theory**
  Standard formula format SMTLIB [3]
  Enormous breakthrough, many tools, ... [31]

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- 20 years ago: Tools for Software Model Checking
- 15 years ago: Satisfiability Modulo Theory
- **10 years ago: Competition on Software Verification** 2012 [4]
  Competitions create awareness of tools,
  provide comparative evaluations, establish standards
  (input, exchange, comparability, reproducibility)

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- 20 years ago: Tools for Software Model Checking
- 15 years ago: Satisfiability Modulo Theory
- 10 years ago: Competition on Software Verification
- **today:** From **lack** of tools to **abundance** of tools
  Problem: missing standard interfaces, missing cooperation

Status on Verifiers

- From lack of verifiers to plentitude

Example CPAchecker [21]: Many Concepts

- Included Concepts:
  - CEGAR [27] Interpolation [24, 12]
  - Configurable Program Analysis [16, 17]
  - Adjustable-block encoding [22]
  - Conditional model checking [15]
  - Verification witnesses [10, 9]
  - Various abstract domains: predicates, intervals, BDDs, octagons, explicit values
- Available analyses approaches:
  - Predicate abstraction [7, 22, 17, 25]
  - Impact algorithm [40, 26, 12]
  - Bounded model checking [29, 12]
  - k-Induction [11, 12]
  - IC3/Property-directed reachability [8]
  - Explicit-state model checking [24]
  - Interpolation-based model checking [23]

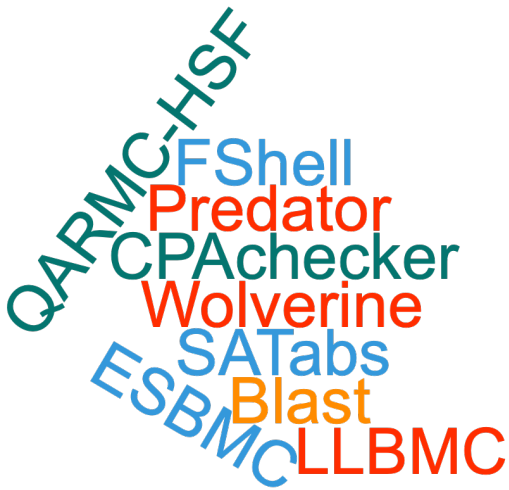
Competitions in Software Verification and Testing

Mature research area, and there are tool competitions:

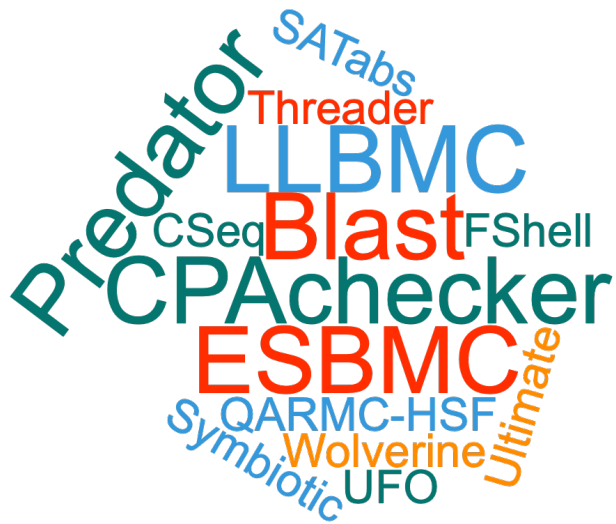
- RERS: off-site, tools, free-style [36]
- SV-COMP: off-site, automatic tools, controlled [4]
- Test-Comp: off-site, automatic tools, controlled [6]
- VerifyThis: on-site, interactive, teams [37]

(alphabetic order)

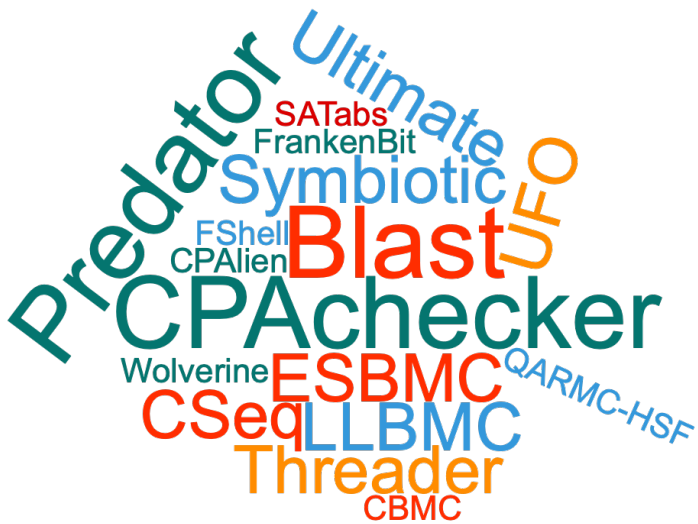
SV-COMP (Automatic Tools 2012)



SV-COMP (Automatic Tools 2013, cumulative)

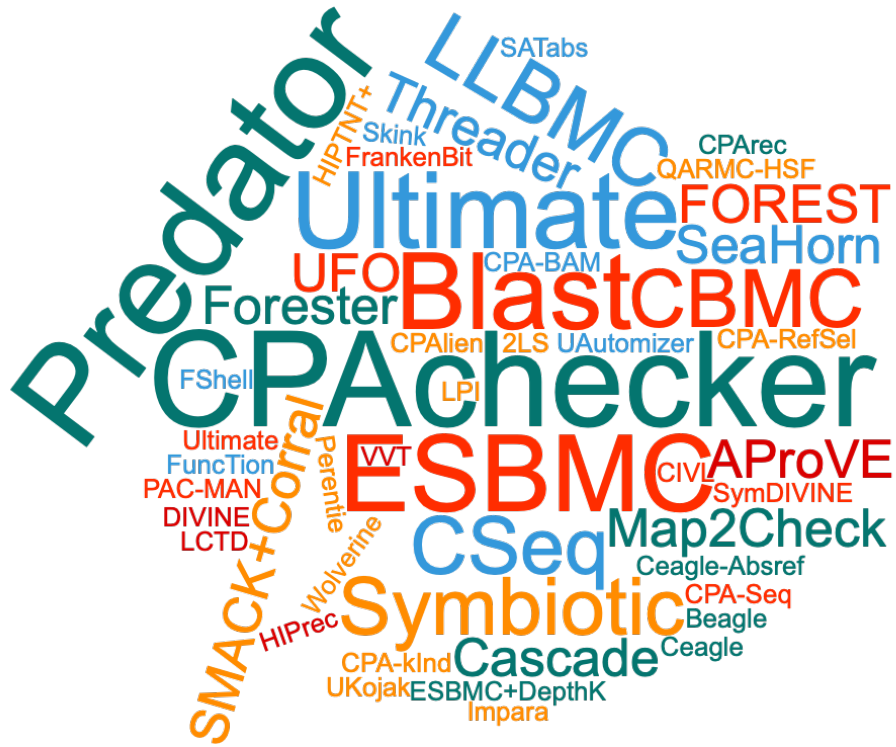


SV-COMP (Automatic Tools 2014, cumulative)

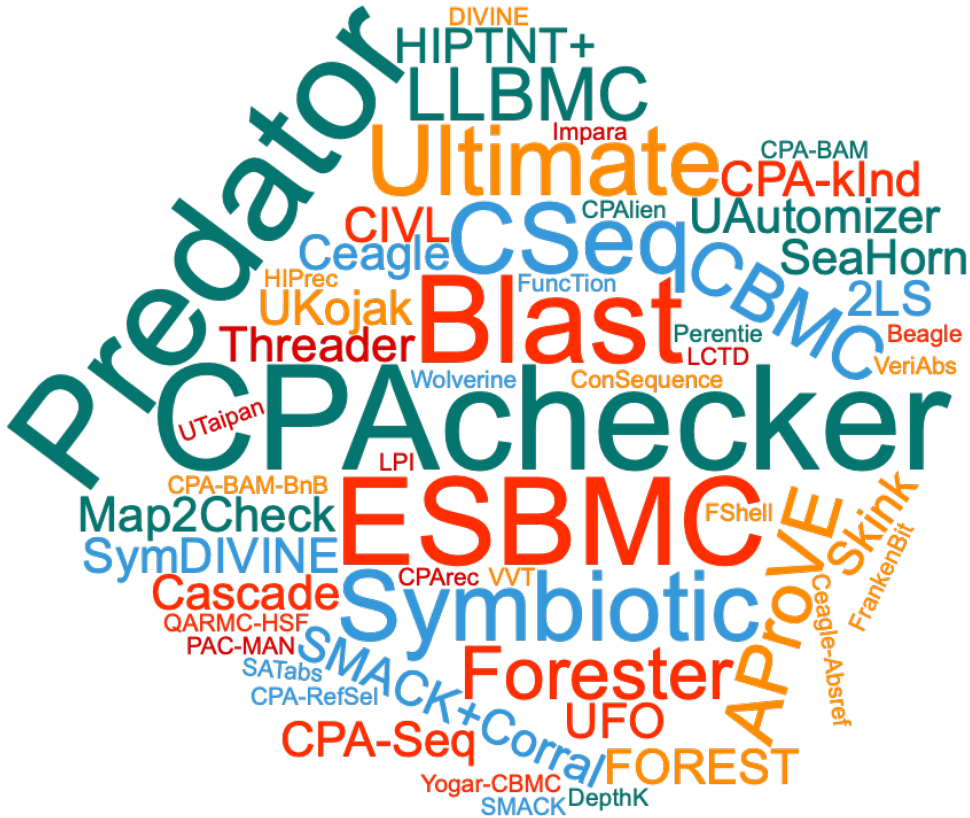


SV-COMP (Automatic Tools 2015, cumulative)

SV-COMP (Automatic Tools 2016, cumulative)

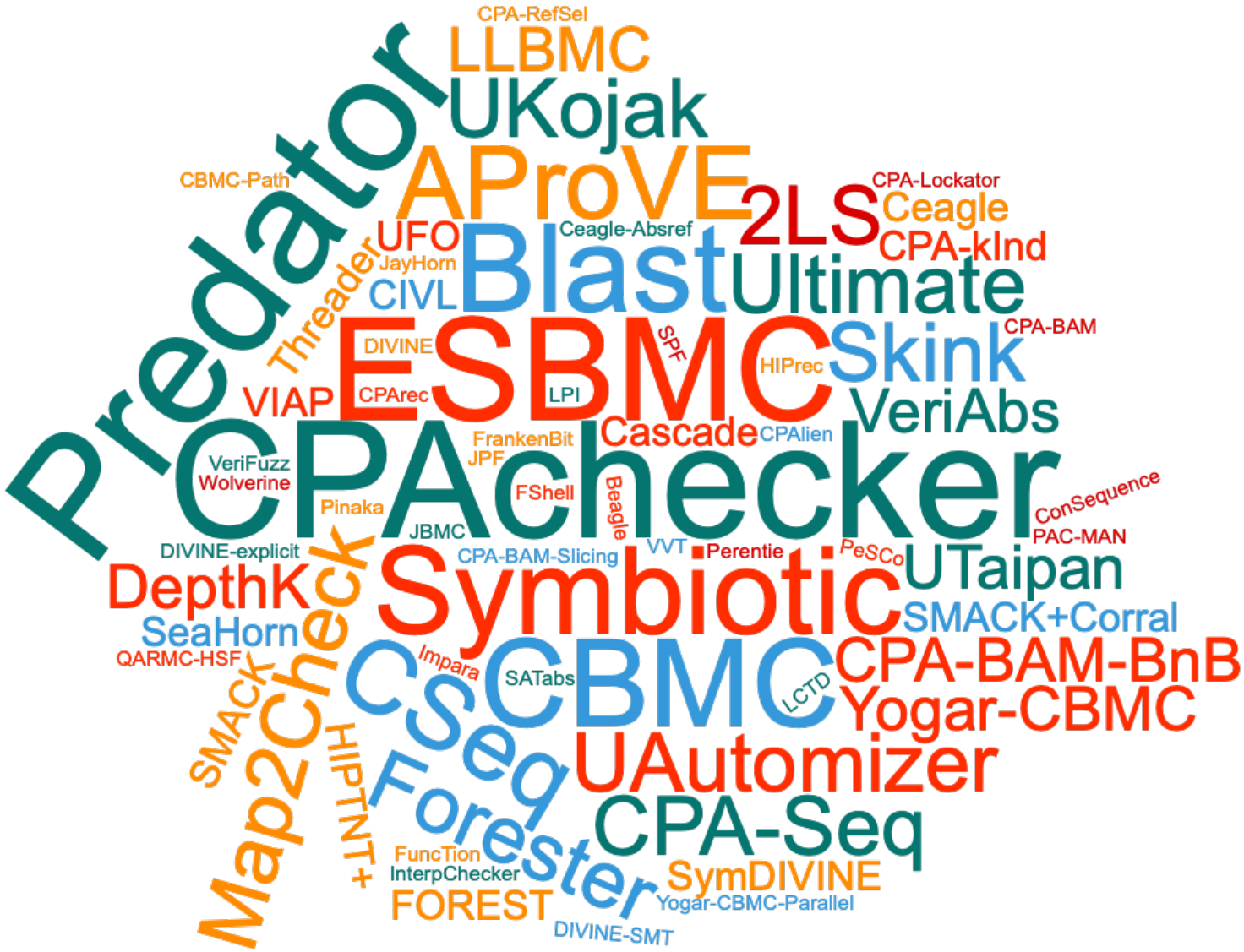


SV-COMP (Automatic Tools 2017, cumulative)



SV-COMP (Automatic Tools 2018, cumulative)

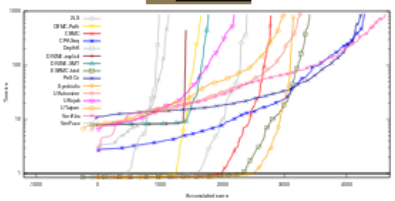
SV-COMP (Automatic Tools 2019, cumulative)



Different Strengths

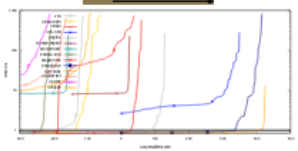
ReachSafety

1. VeriAbs
2. CPA-Seq
3. PeSCo



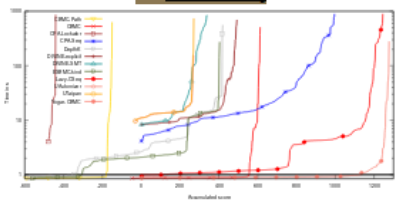
MemSafety

1. Symbiotic
2. PredatorHP
3. CPA-Seq



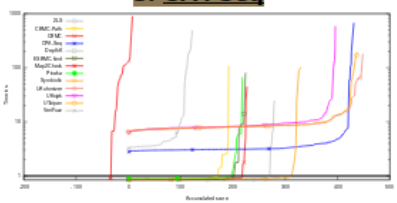
ConcurrencySafety

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



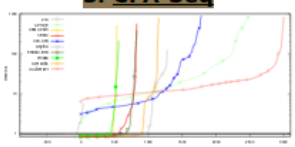
NoOverflows

1. UAutomizer
2. UTaipan
3. CPA-Seq



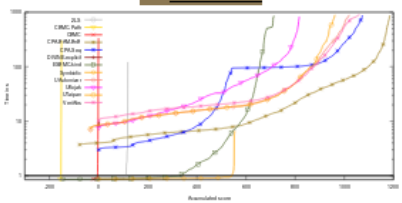
Termination

1. UAutomizer
2. AProVE
3. CPA-Seq



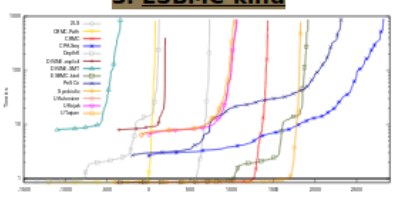
SoftwareSystems

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



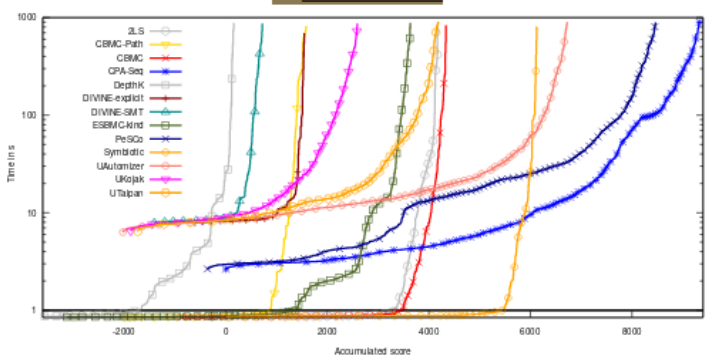
FalsificationOverall

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 [5]

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

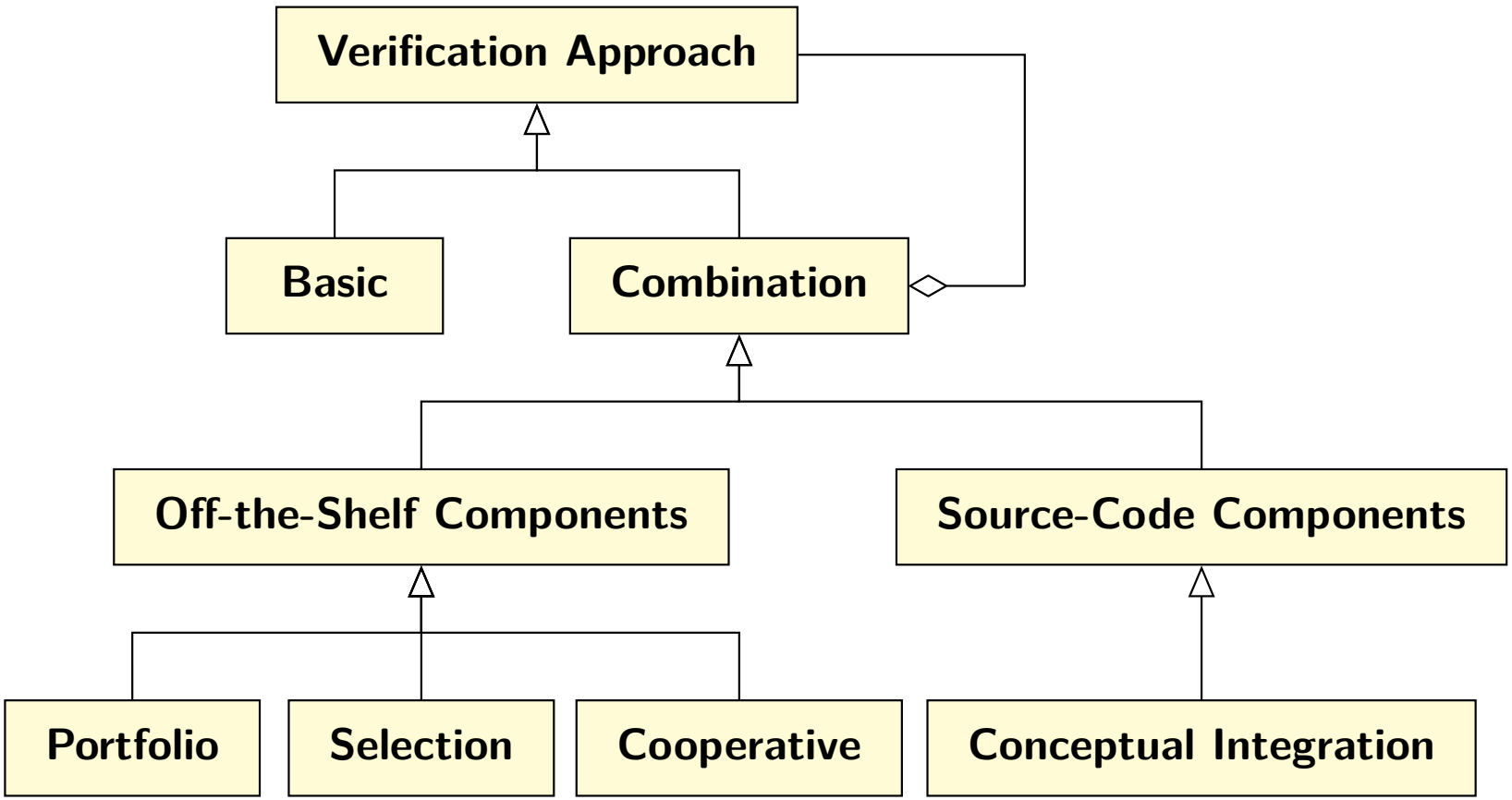
Insights from Software Model Checking

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

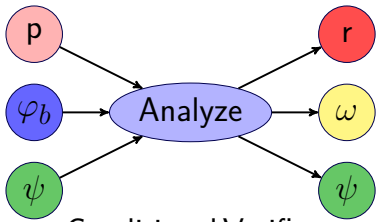
Cooperative Verification — Think big!

- 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 [41], Evidential Tool Bus [33]

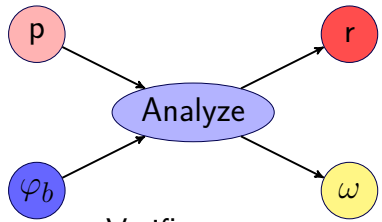
Approaches for Combinations



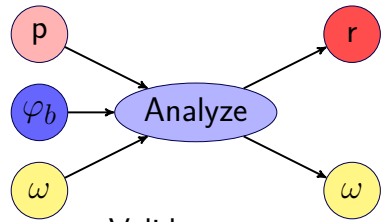
Graphical Visualization of the Coop Framework



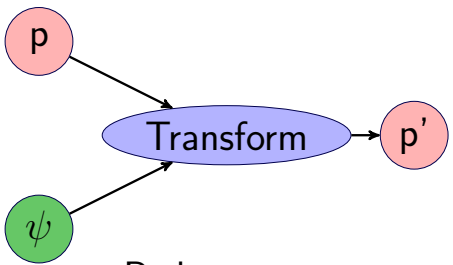
Conditional Verifier



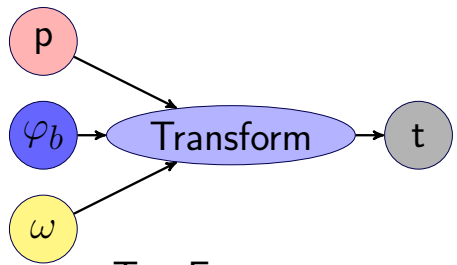
Verifier



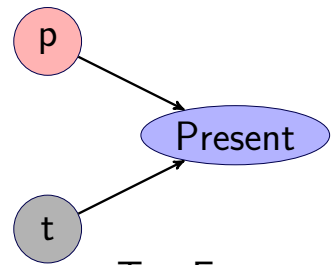
Validator



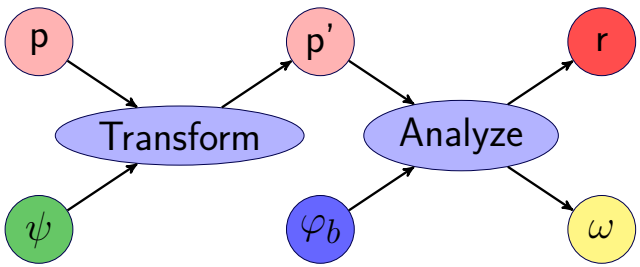
Reducer



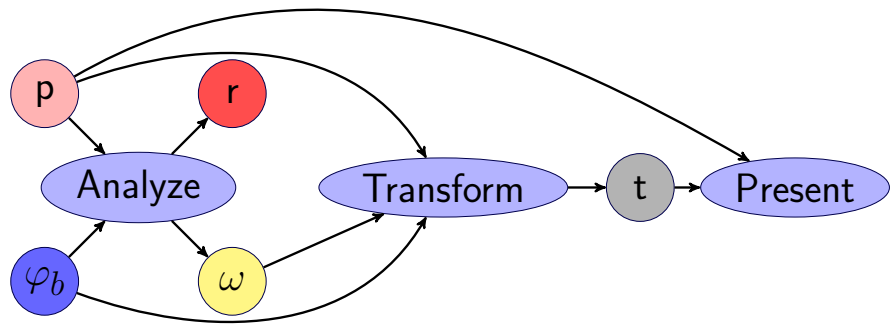
Test Extractor



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*.

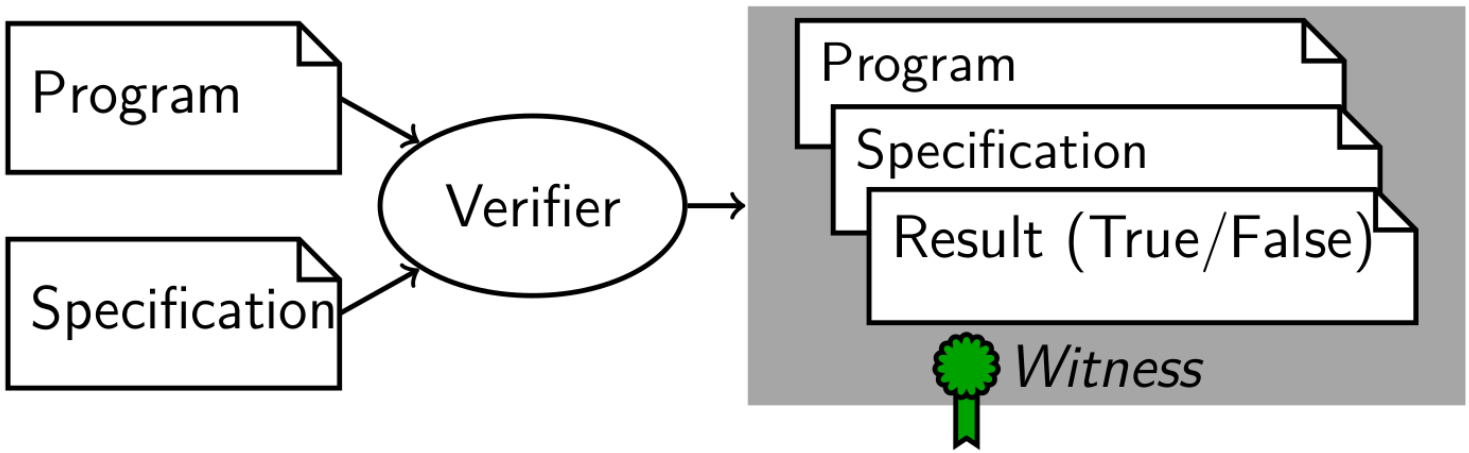
Definition of Cooperative Verification

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

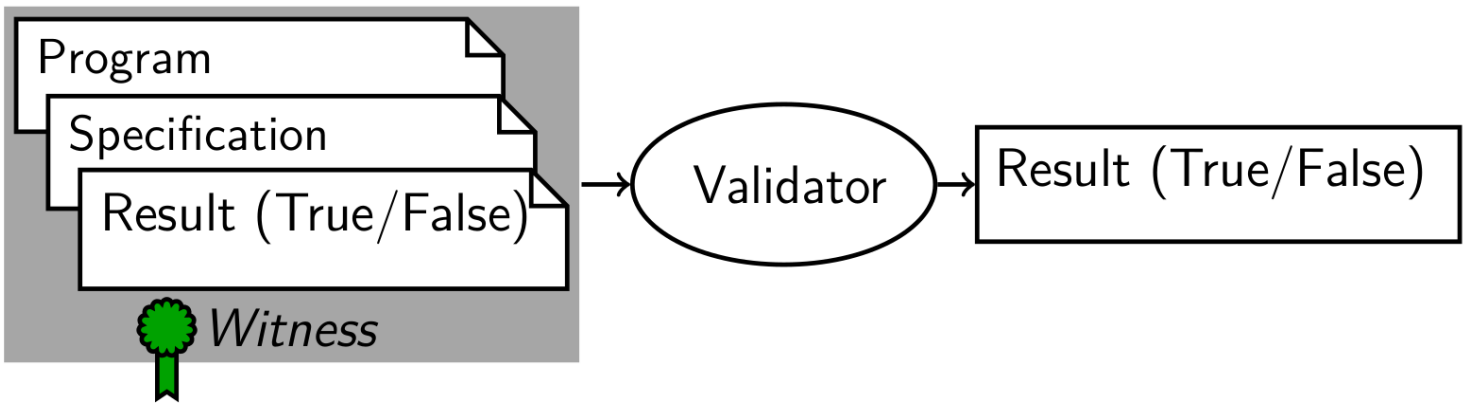
Software Verification with Witnesses

Witnesses are an important interface between tools.



[10, Proc. FSE 2015] [9, Proc. FSE 2016]

Witness Validation



- Validate untrusted results
- Easier than full verification

Example Combination (in DSL CoVeriTeam)

CoVeriTeam: Language and Tool [19, Proc. TACAS 2022]

Algorithm 1 Witness Validation [10, 9]

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

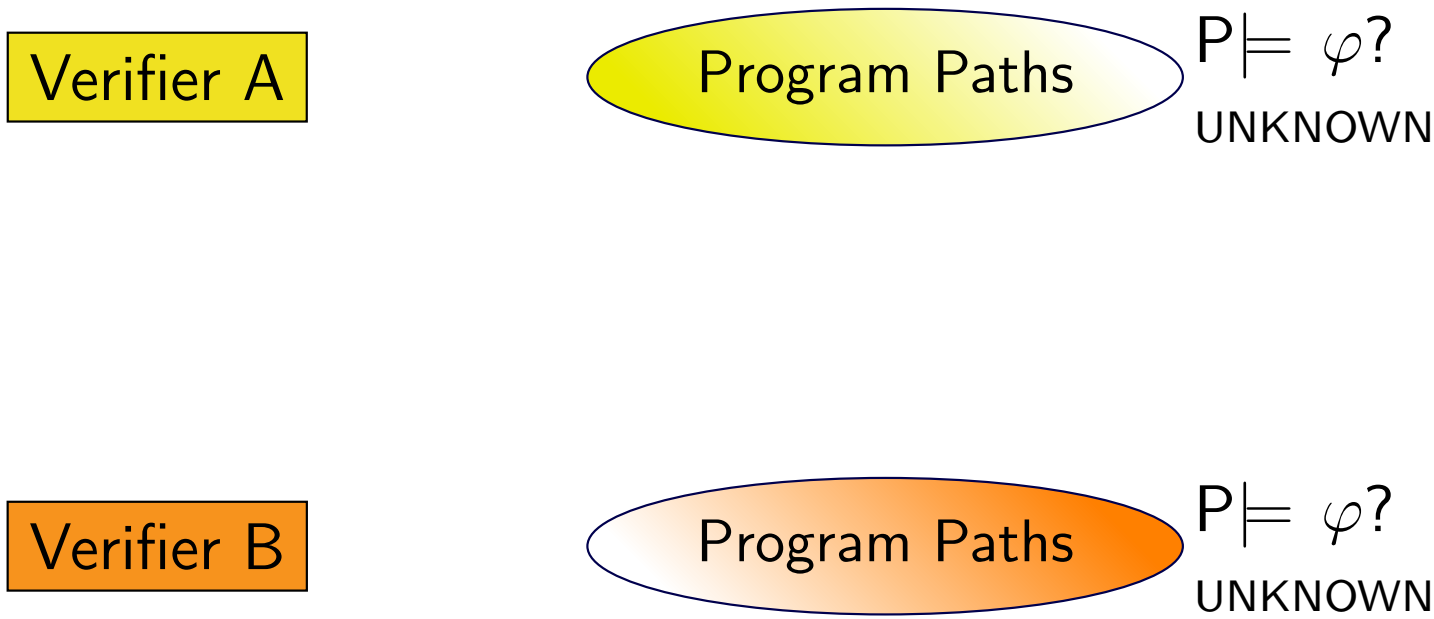
Often, even simple combinations help!

Portfolio construction using off-the-shelf
verification tools [20, 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)

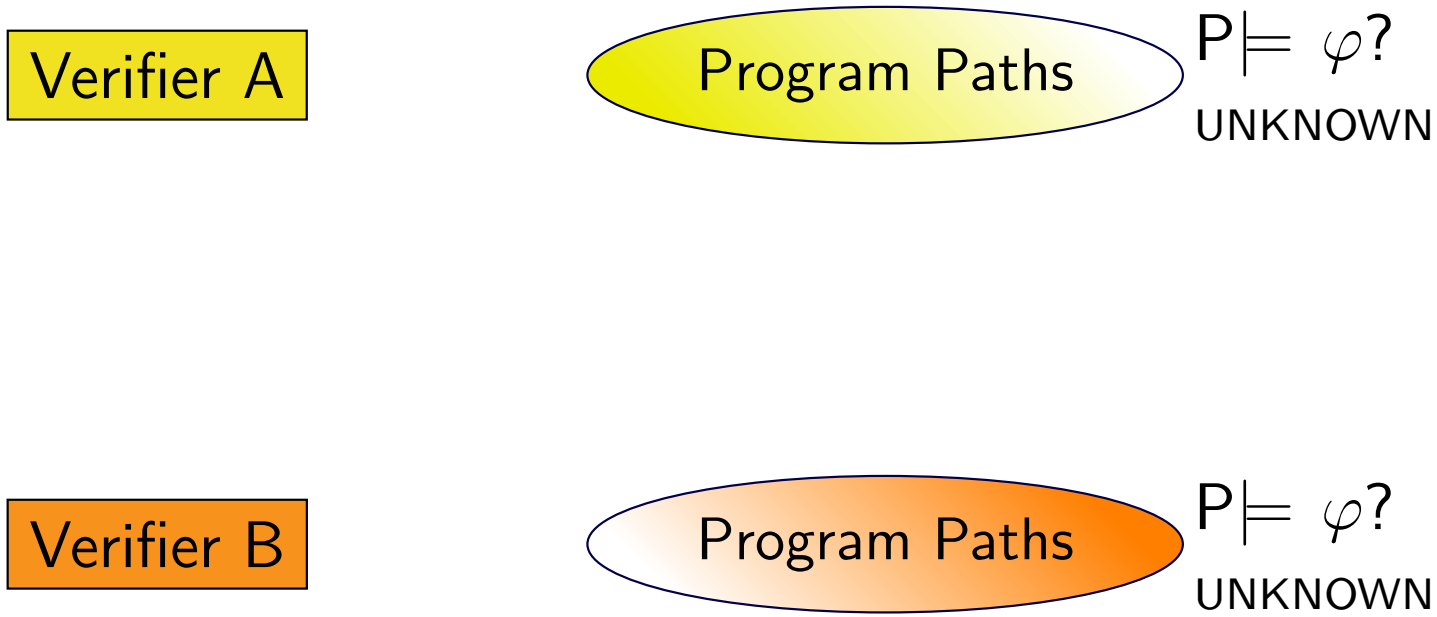
Facing Hard Verification Tasks

Given: Program P$\models \varphi$?

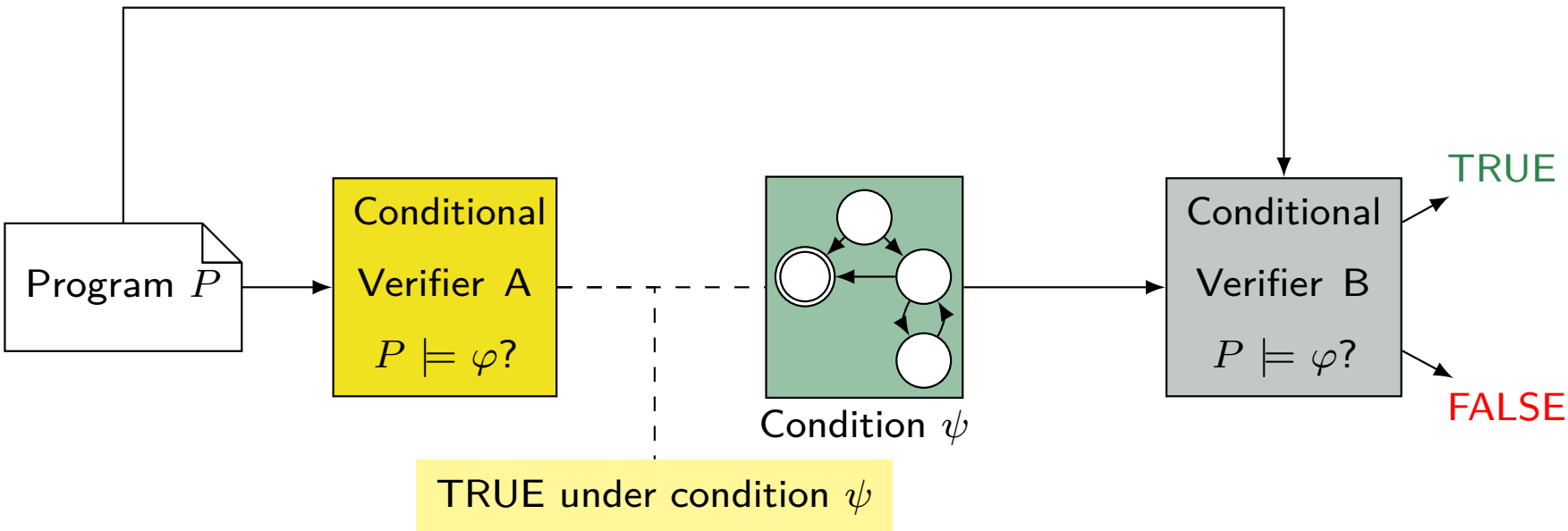


Facing Hard Verification Tasks

Given: Program P$\models \varphi$?

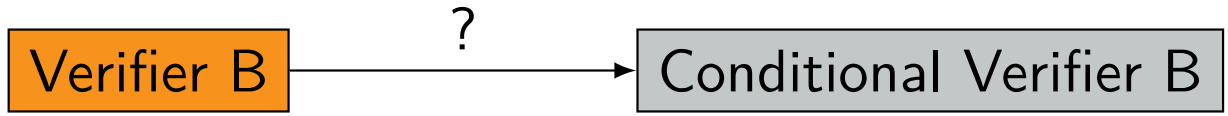


Conditional Model Checking



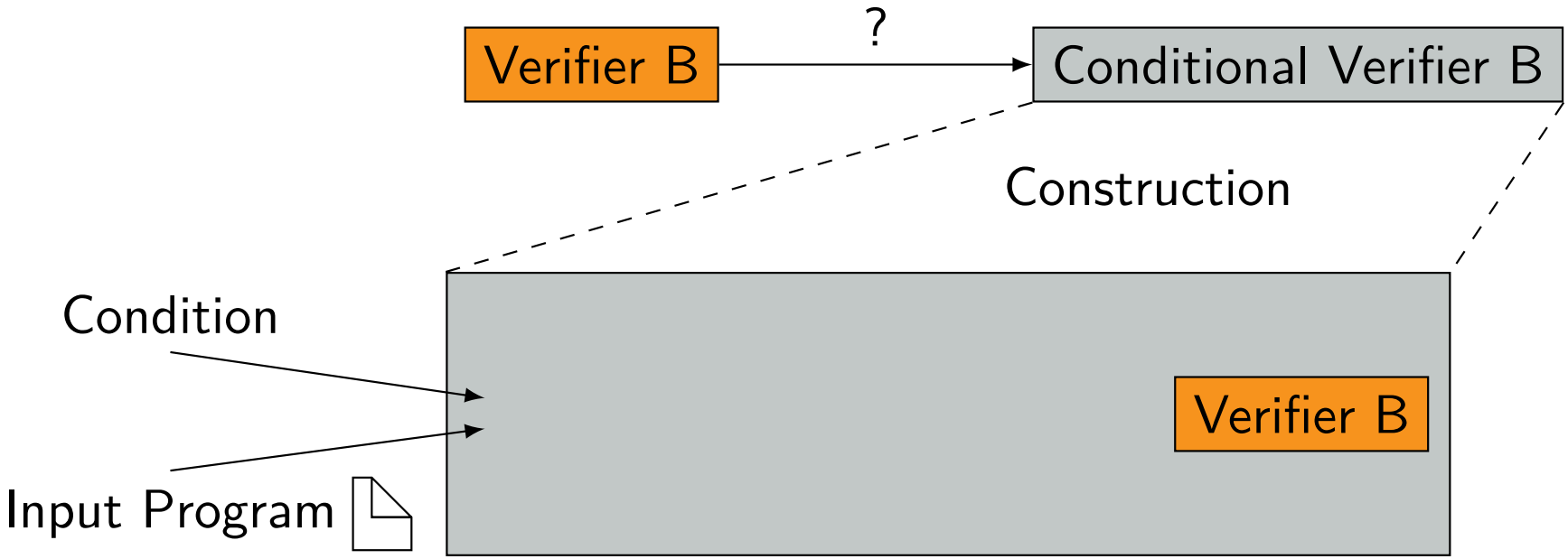
Proc. FSE 2012 [15]

Reducer-Based Construction



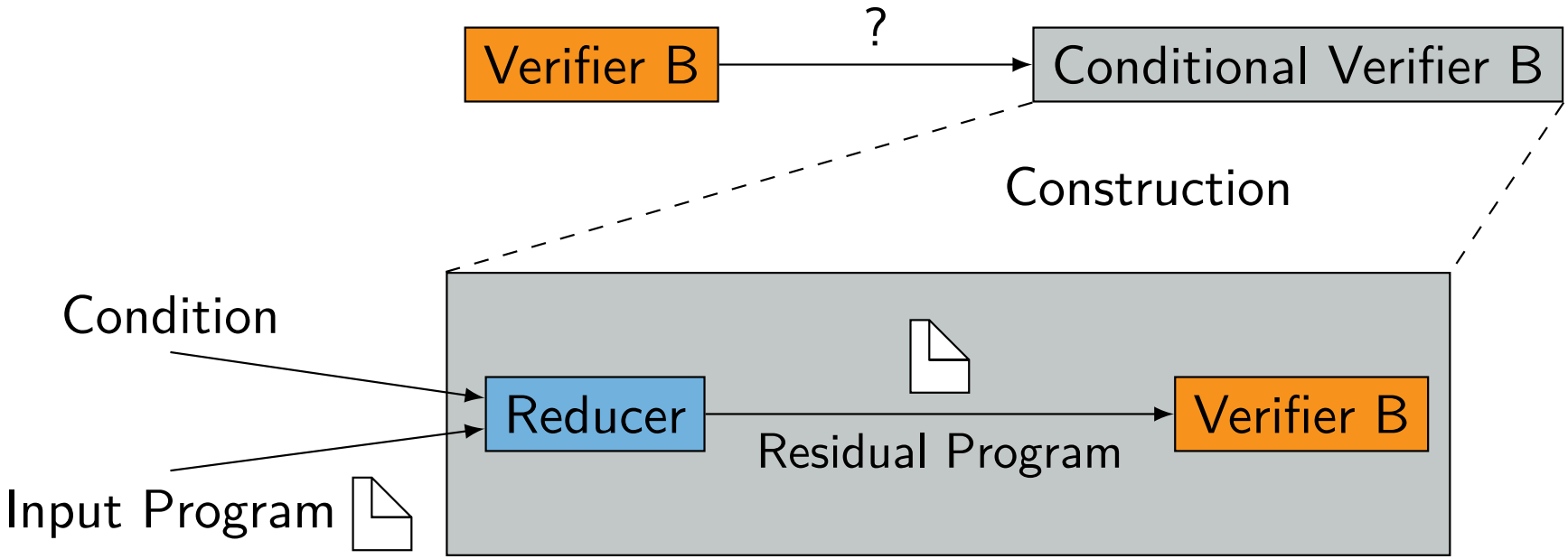
Proc. ICSE 2018 [18]

Reducer-Based Construction



Proc. ICSE 2018 [18]

Reducer-Based Construction

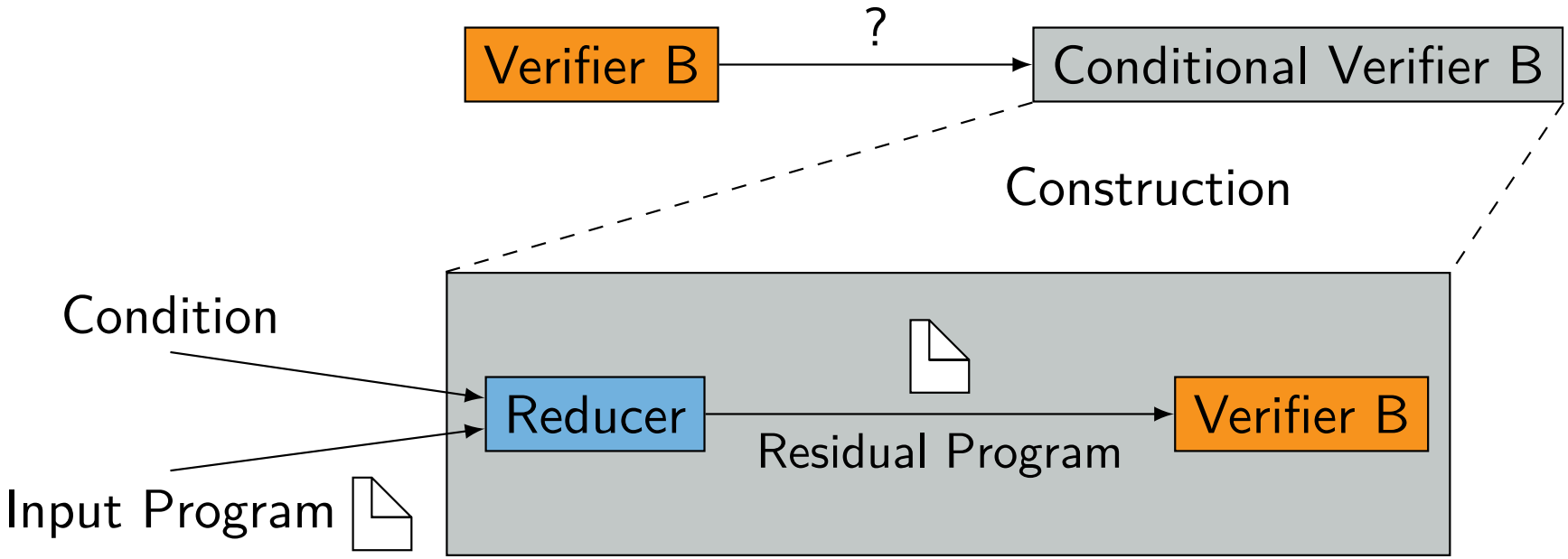


Reducer (preprocessor)

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

Proc. ICSE 2018 [18]

Reducer-Based Construction



Reducer (preprocessor)

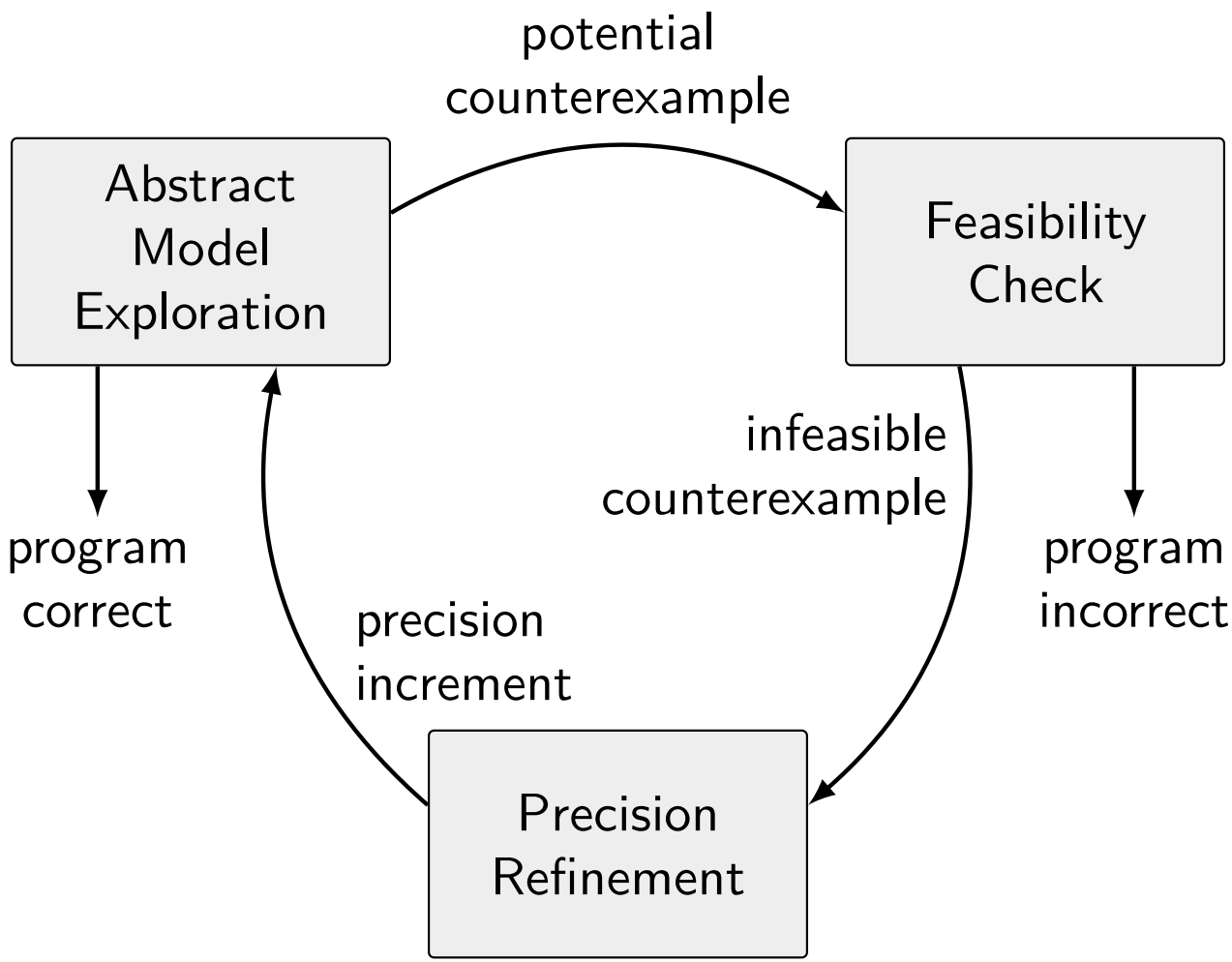
- Builds standard input (C program)
- Representing a subset of paths
- Contains at least all non-verified paths

\+ Verifier-unspecific approach

\+ Many conditional verifiers possible

Proc. ICSE 2018 [18]

CEGAR

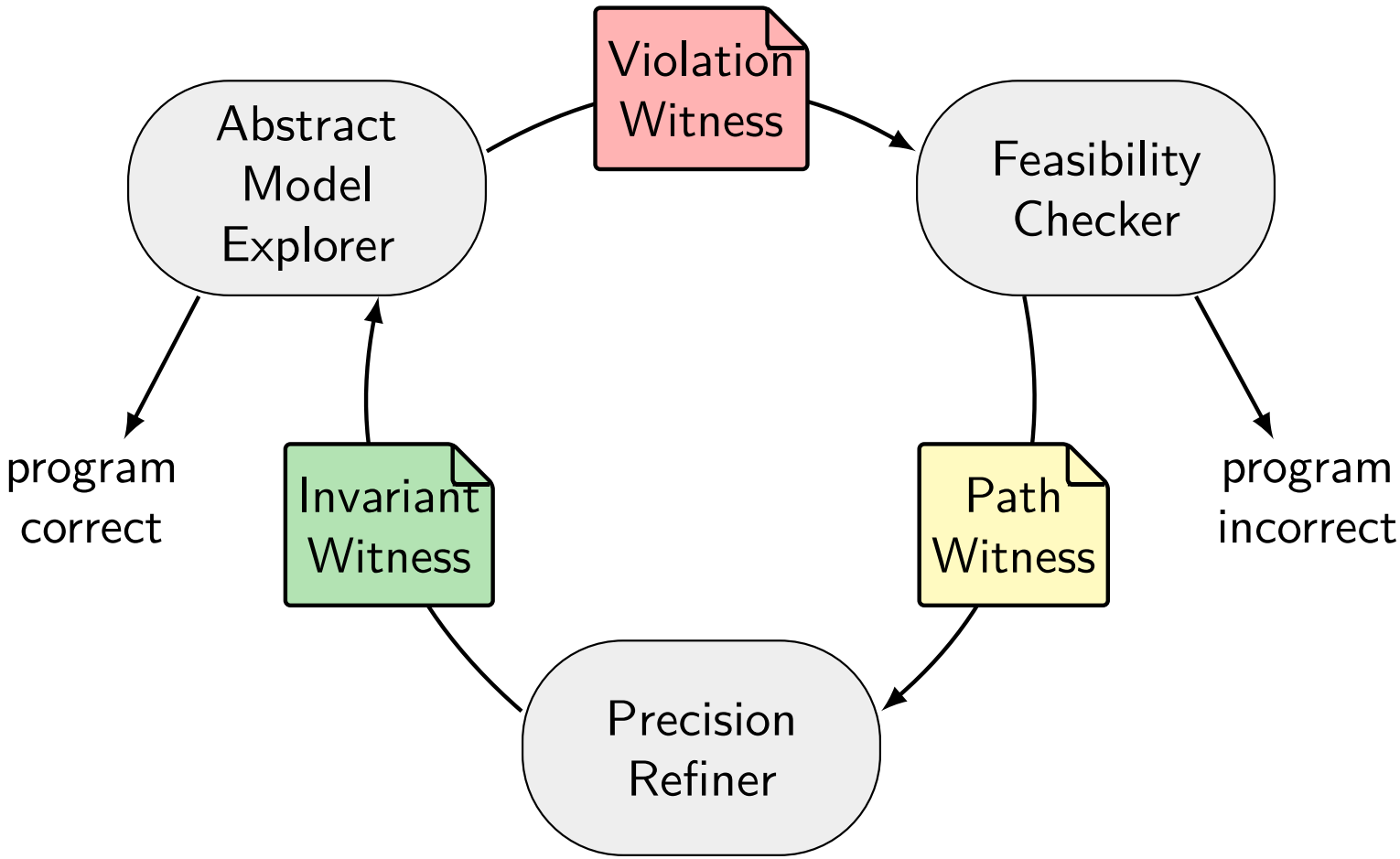

potential
counterexample
Abstract
Model
Exploration
Feasibility
Check
infeasible
counterexample
program
correct
program
incorrect
precision
increment
Precision
Refinement

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 [13]

Conclusion

- Software Verification as a mature research area
- Combinations and Cooperation

References I

[1] Ball, T., Levin, V., Rajamani, S.K.: A decade of software model checking with SLAM. Commun. ACM **54**(7), 68–76 (2011).
https://doi.org/10.1145/1965724.1965743

[2] Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Proc. PLDI. pp. 203–213. ACM (2001).
https://doi.org/10.1145/378795.378846

[3] Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.5. Tech. rep., University of Iowa (2015), available at https://smtlib.cs.uiowa.edu/

[4] 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

[5] 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

[6] 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

References II

[7] 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

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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

References III

[13] Beyer, D., Haltermann, J., Lemberger, T., Wehrheim, H.: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. In: Proc. ICSE. pp. 536–548. ACM (2022). https://doi.org/10.1145/3510003.3510064

[14] Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. Int. J. Softw. Tools Technol. Transfer **9**(5-6), 505–525 (2007). https://doi.org/10.1007/s10009-007-0044-z

[15] 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

[16] 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

[17] 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

[18] 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

References IV

[19] 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

[20] 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

[21] 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

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

[23] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. arXiv/CoRR **2208**(05046) (July 2022). https://doi.org/10.48550/arXiv.2208.05046

[24] 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

References V

[25] 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

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

[27] 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

[28] Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of Model Checking. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8

[29] 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

[30] Clarke, E.M., Kröning, D., Sharygina, N., Yorav, K.: SATABS: SAT-based predicate abstraction for ANSI-C. In: Proc. TACAS. pp. 570–574. LNCS 3440, Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_40

[31] Cok, D.R., Déharbe, D., Weber, T.: The 2014 SMT competition. JSAT **9**, 207–242 (2016)

References VI

[32] Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957). https://doi.org/10.2307/2963593

[33] 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

[34] Graf, S., Saïdi, H.: Construction of abstract state graphs with Pvs. In: Proc. CAV. pp. 72–83. LNCS 1254, Springer (1997). https://doi.org/10.1007/3-540-63166-6_10

[35] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. POPL. pp. 58–70. ACM (2002). https://doi.org/10.1145/503272.503279

[36] 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

[37] 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

[38] Kildall, G.A.: A unified approach to global program optimization. In: Proc. POPL. pp. 194–206. ACM (1973). https://doi.org/10.1145/512927.512945

References VII

[39] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1

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

[41] 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

[42] Turing, A.: Checking a large routine. In: Report on a Conference on High Speed Automatic Calculating Machines. pp. 67–69. Cambridge Univ. Math. Lab. (1949)