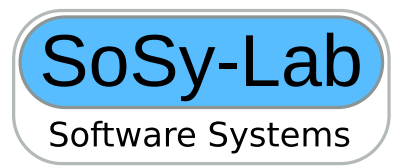
Cooperative Verification

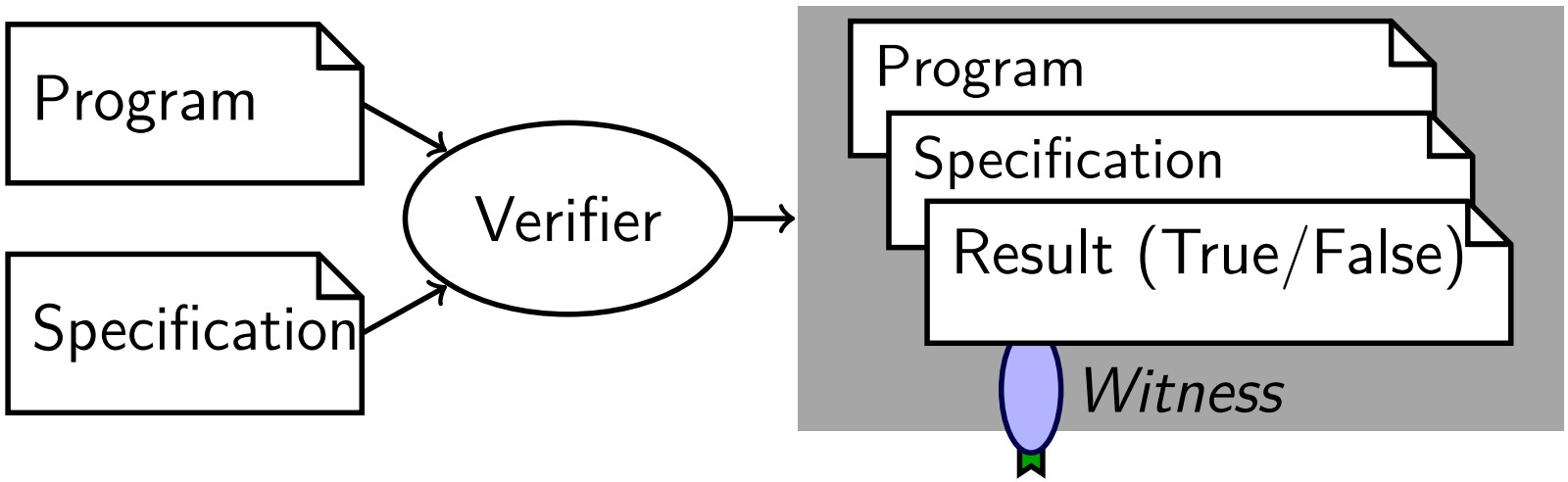
CPAchecker-Workshop 2019, Chiemsee, 2019-10-01

Dirk Beyer

LMU Munich, Germany



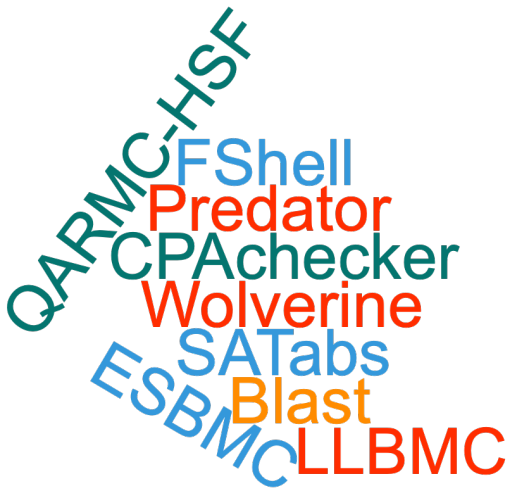
Automatic Software Verification



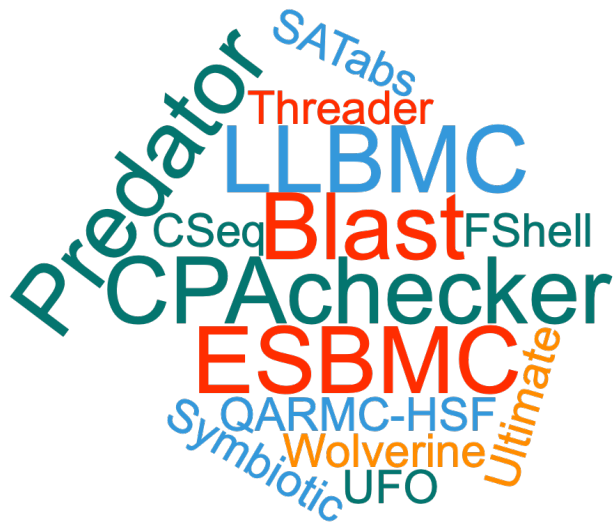
Competitions in Software Verification and Testing

- SV-COMP: off-site, automatic tools, controlled [1]
- Test-Comp: off-site, automatic tools, controlled [3]

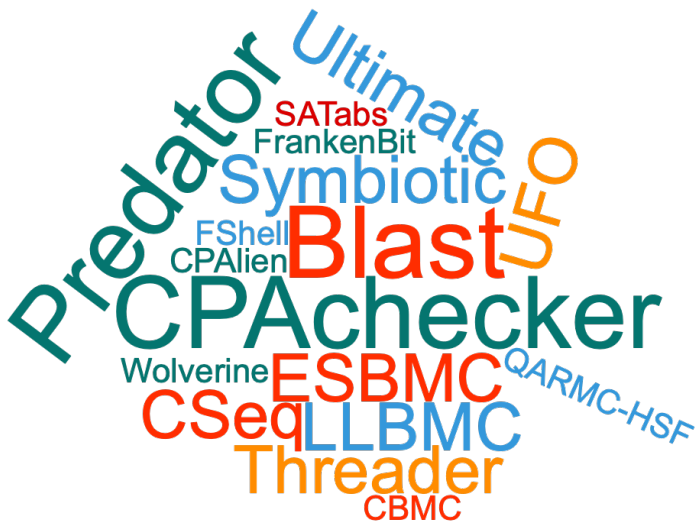
SV-COMP (Automatic Tools 2012)



SV-COMP (Automatic Tools 2013, cumulative)

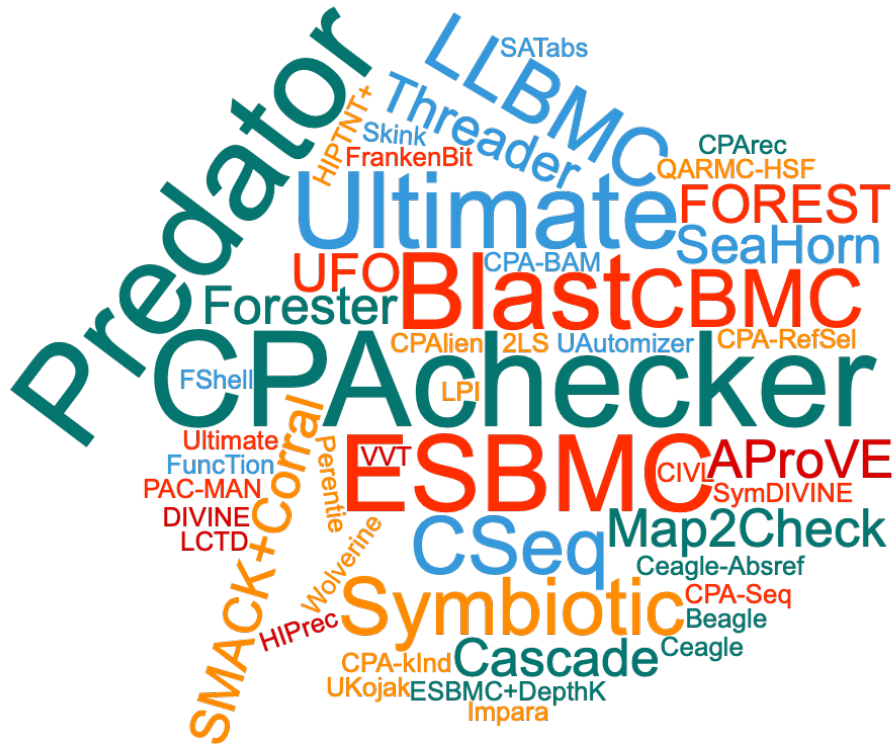


SV-COMP (Automatic Tools 2014, cumulative)

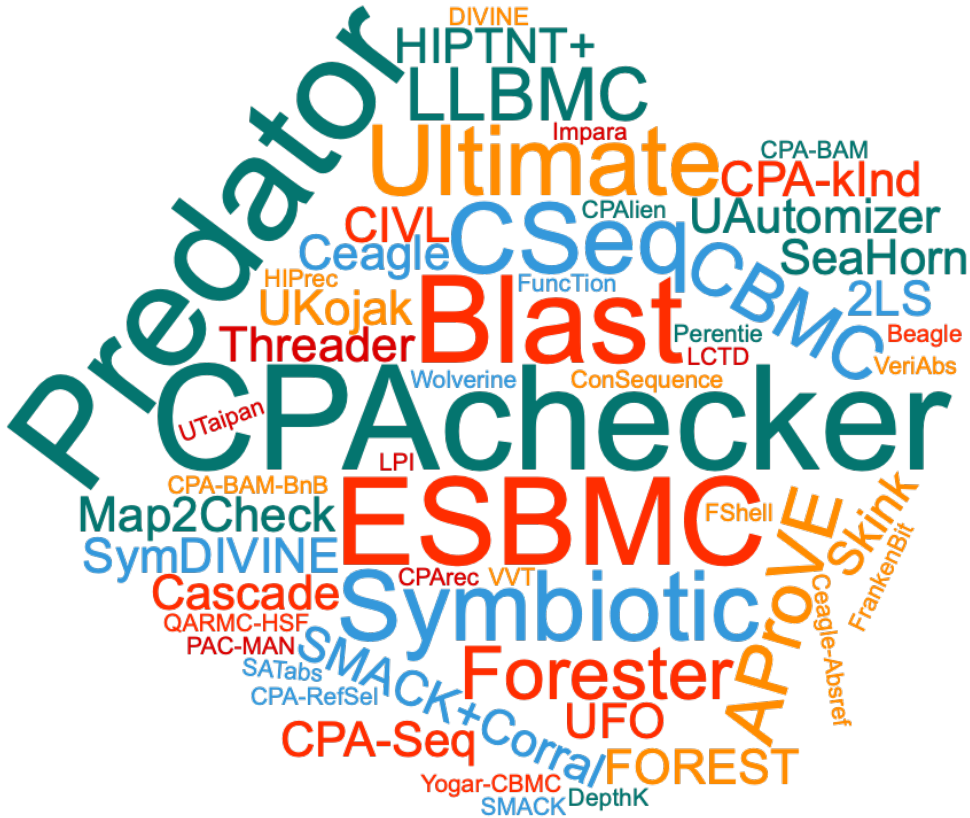


SV-COMP (Automatic Tools 2015, cumulative)

SV-COMP (Automatic Tools 2016, cumulative)

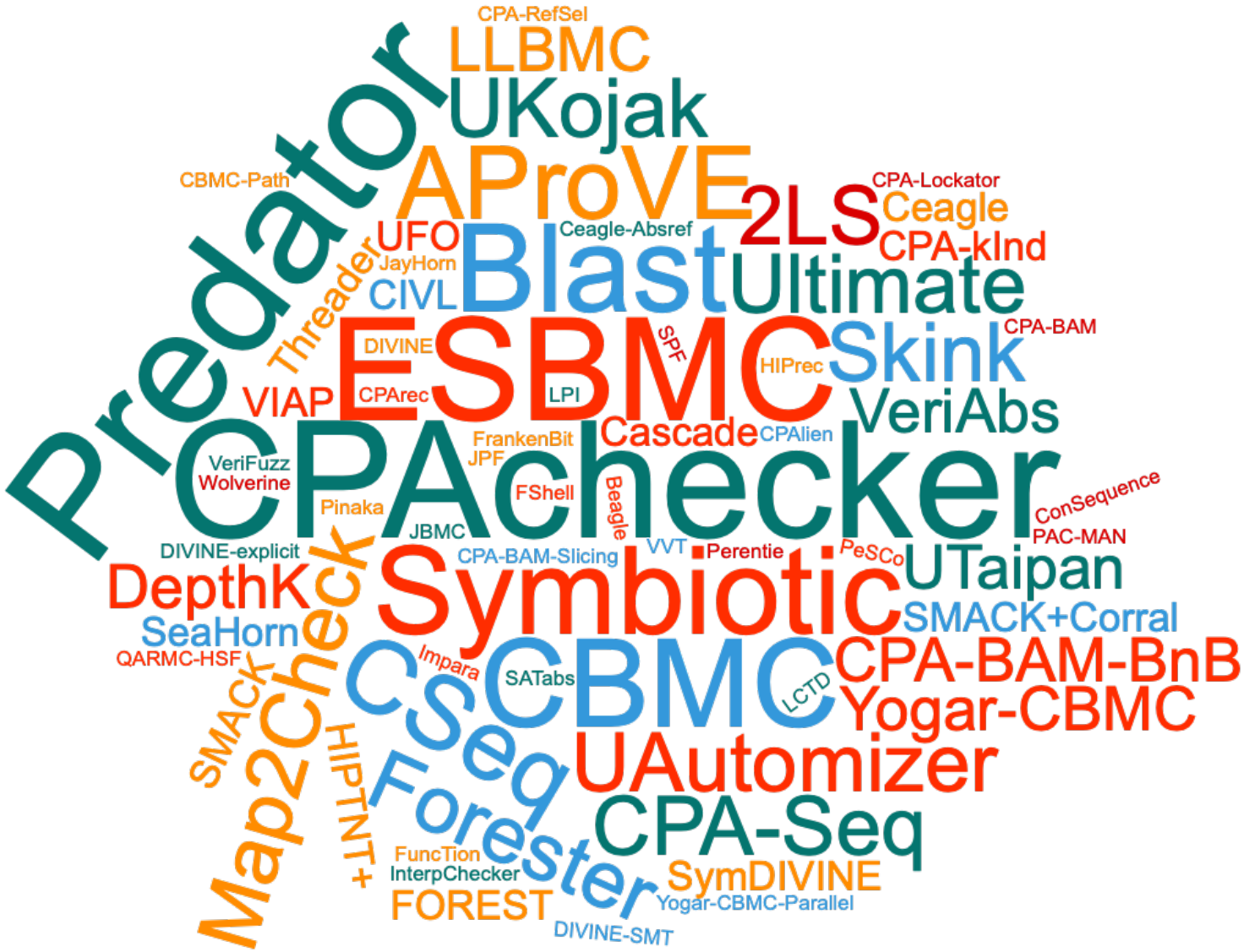


SV-COMP (Automatic Tools 2017, cumulative)



SV-COMP (Automatic Tools 2018, cumulative)

SV-COMP (Automatic Tools 2019, cumulative)



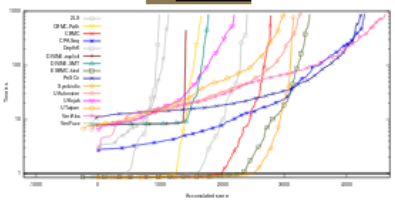
What is the best verifier?

- Many different kinds of programs seem to require many different good tools with different strengths

SV-COMP (Automatic Tools)

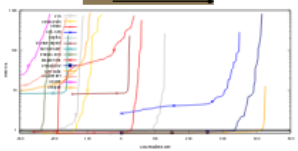
ReachSafety

1. VeriAbs
2. CPA-Seq
3. PeSCo



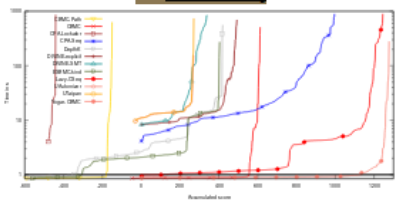
MemSafety

1. Symbiotic
2. PredatorHP
3. CPA-Seq



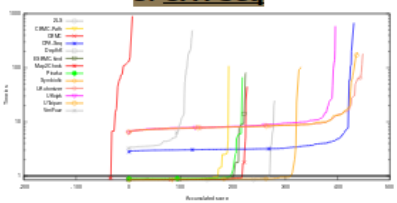
ConcurrencySafety

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



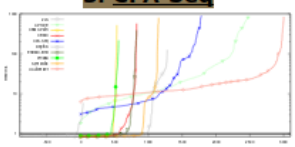
NoOverflows

1. UAutomizer
2. UTaipan
3. CPA-Seq



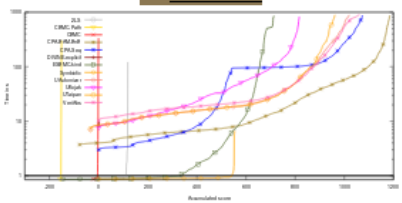
Termination

1. UAutomizer
2. AProVE
3. CPA-Seq



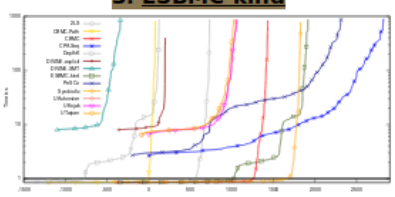
SoftwareSystems

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



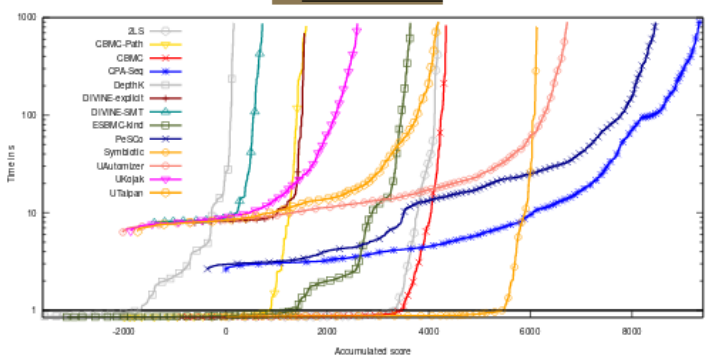
FalsificationOverall

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



Overall

1. CPA-Seq
2. PeSCo
3. UAutomizer



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

Which techniques are used?

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

Algorithms

17	Bounded Model Checking
13	CEGAR
8	Predicate Abstraction
5	k-Induction
4	Symbolic Execution
3	Automata-Based Analysis
2	Property-Directed Reachability (IC3)

Abstract Domains

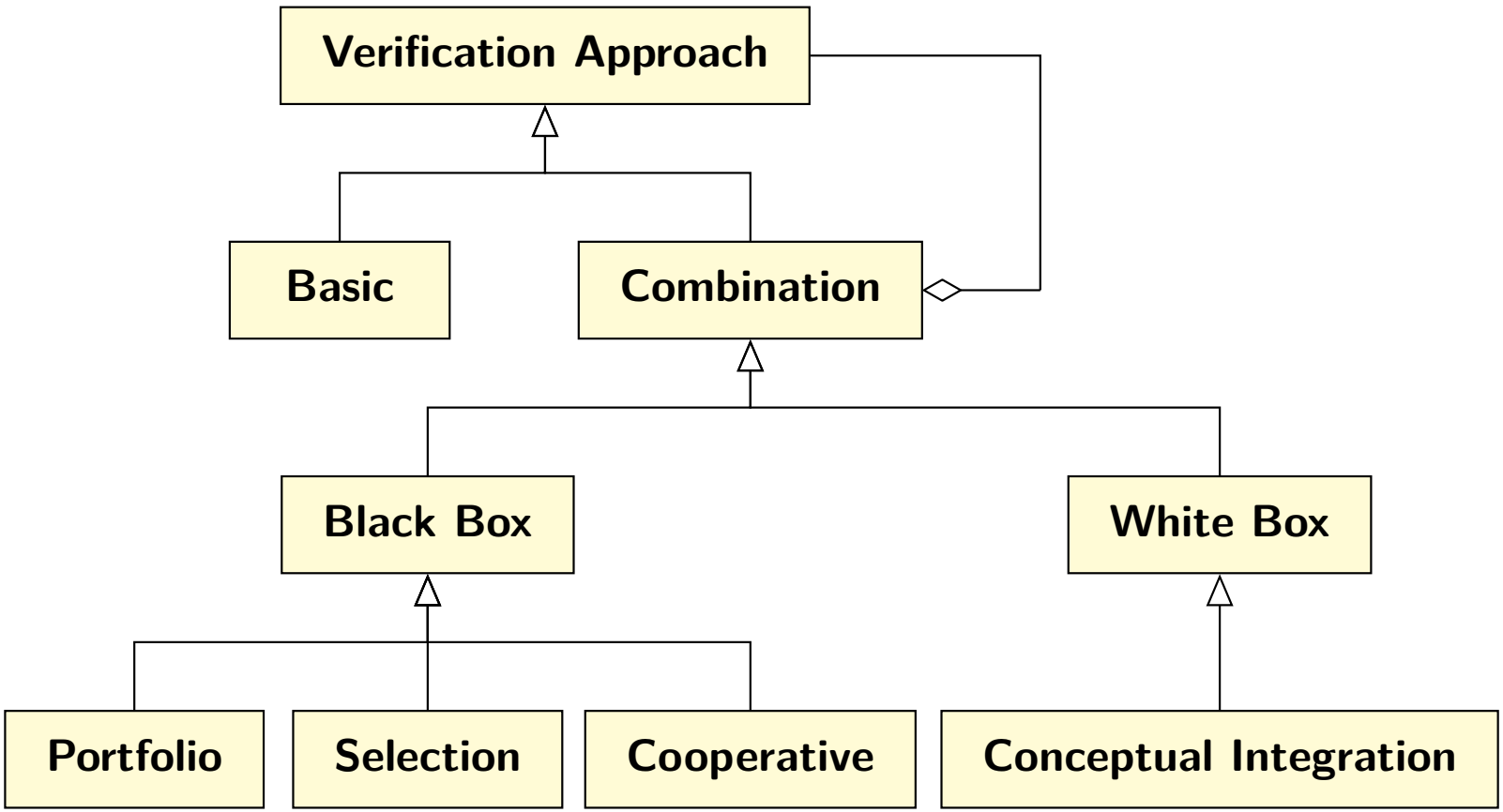
24	Bit-Precise Analysis
10	Explicit-Value Analysis
9	Numerical Interval Analysis
4	Shape Analysis
1	Separation Logic

Testing

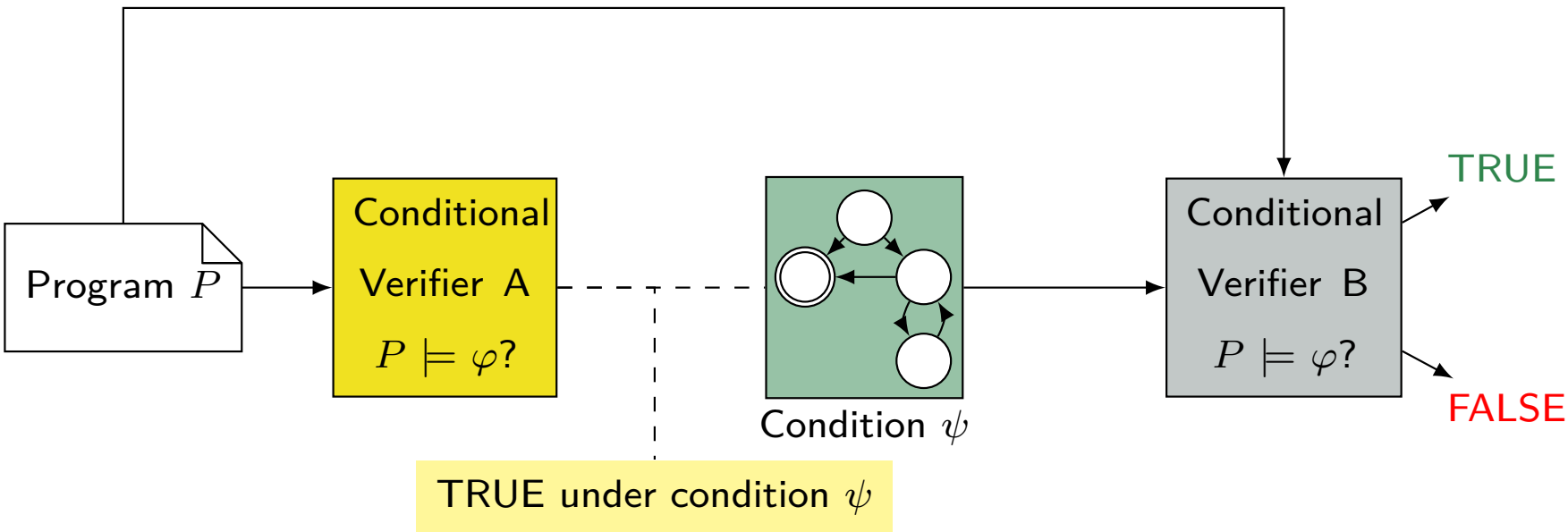
- Fuzzing (VeriFuzz [12], based on AFL)
- Symbolic execution (KLEE [11])
- Software model checking (CoVeriTest [9], → Poster)

Cooperative Verification

Approaches for Combinations

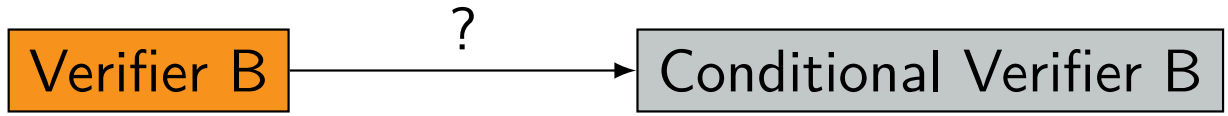


Conditional Model Checking



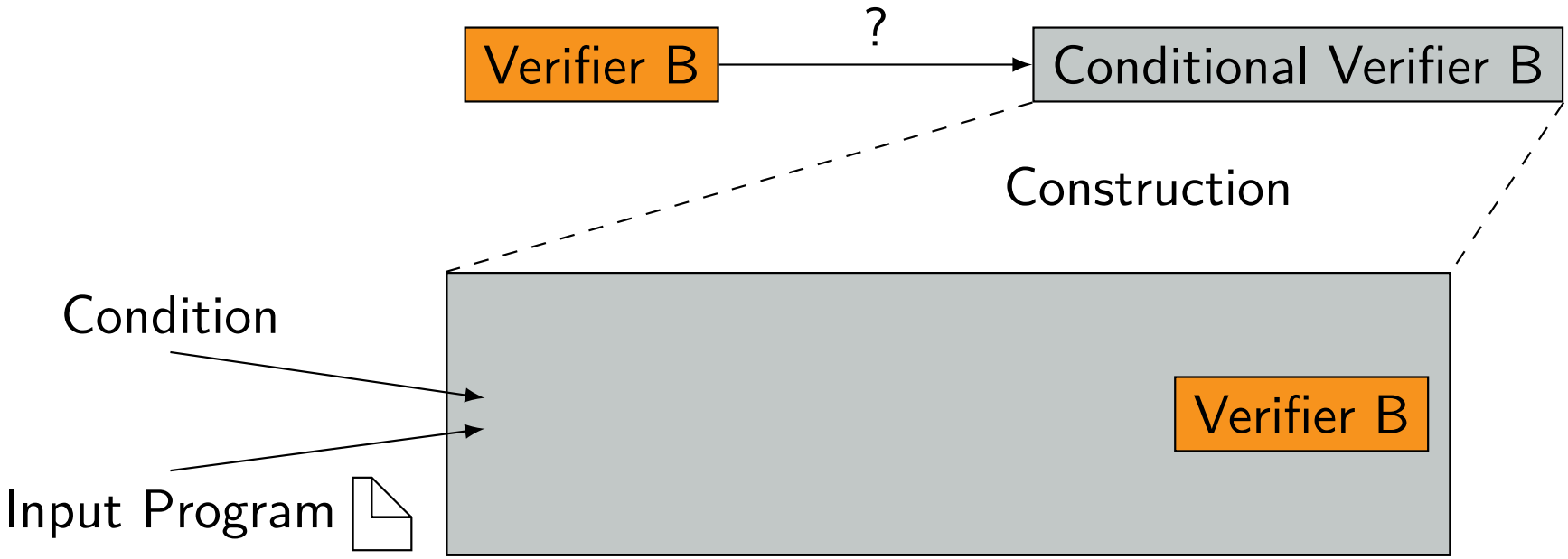
Proc. FSE 2012 [8]

Reducer-Based Construction



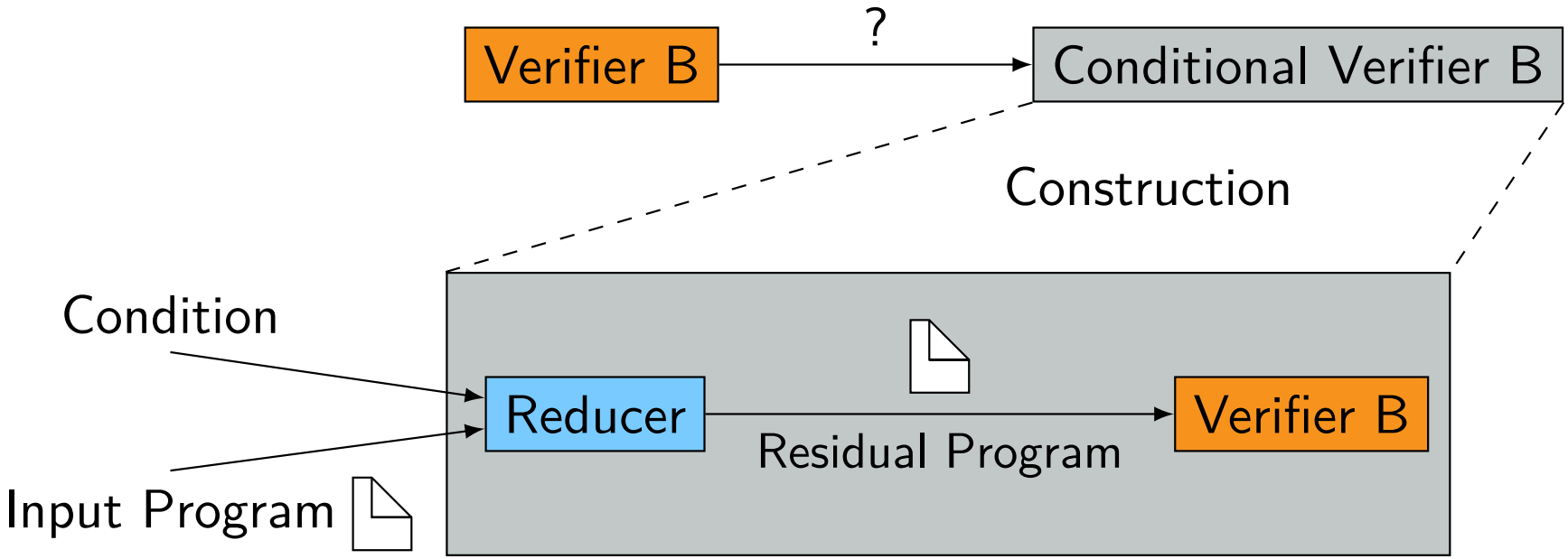
Proc. ICSE 2018 [10]

Reducer-Based Construction



Proc. ICSE 2018 [10]

Reducer-Based Construction

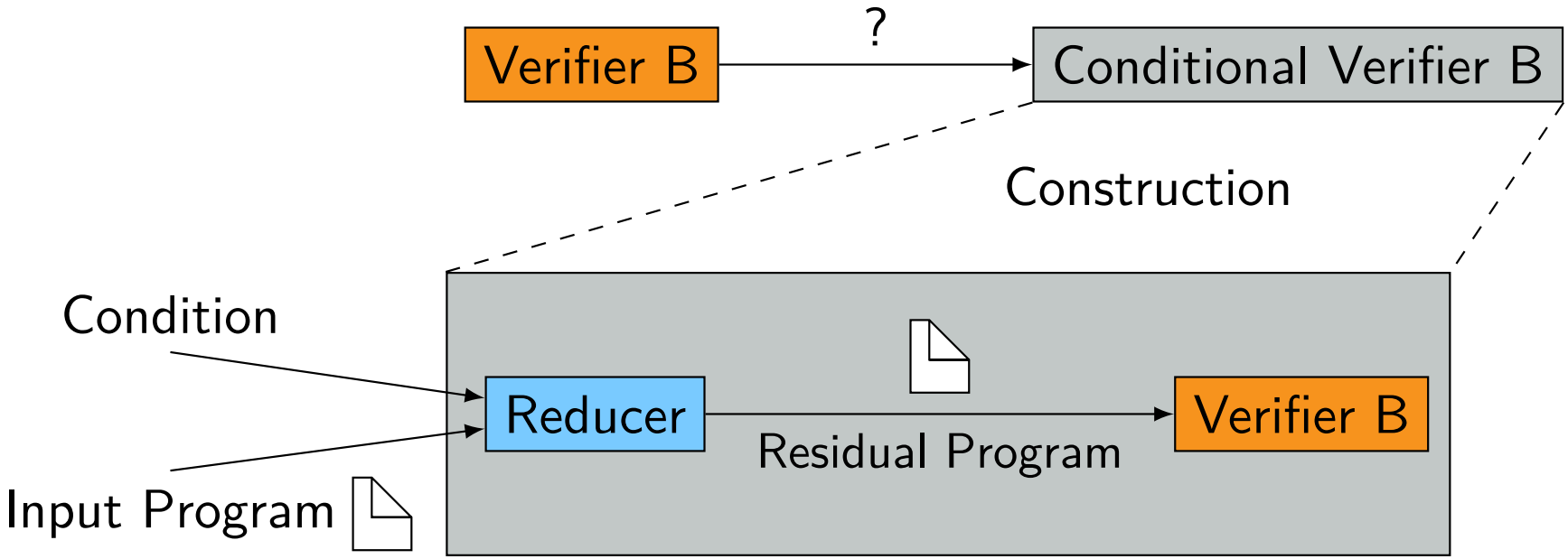


Reducer (preprocessor)

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

Proc. ICSE 2018 [10]

Reducer-Based Construction



Reducer (preprocessor)

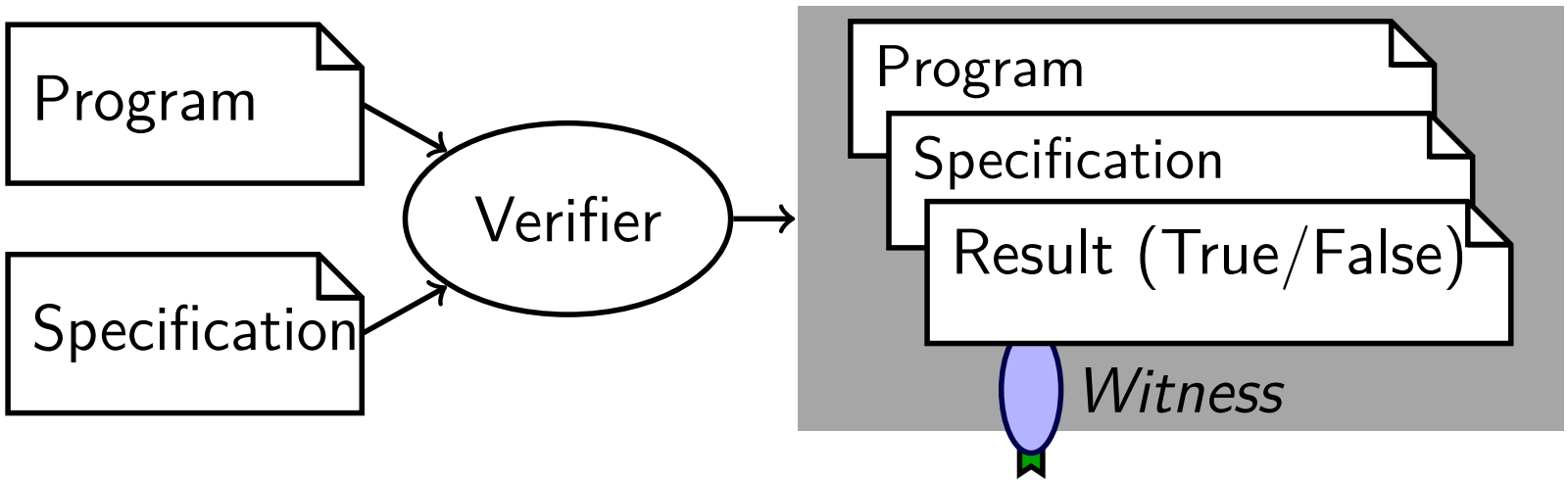
- Builds standard input (C program)
- Representing a subset of paths
- Contains at least all non-verified paths

\+ Verifier-unspecific approach

\+ Many conditional verifiers possible

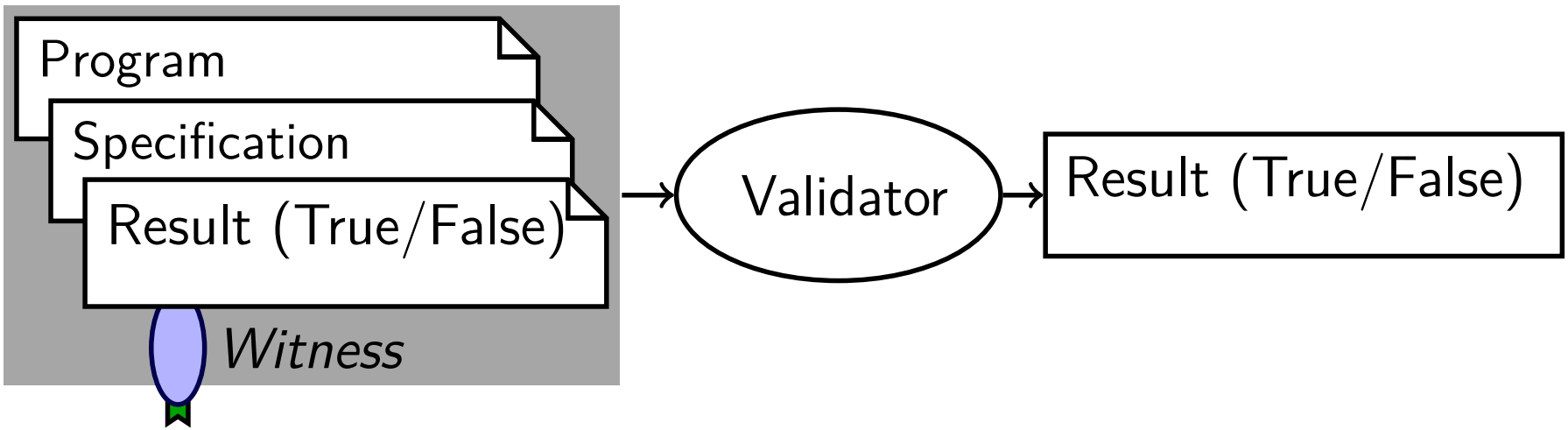
Proc. ICSE 2018 [10]

Software Verification with Witnesses



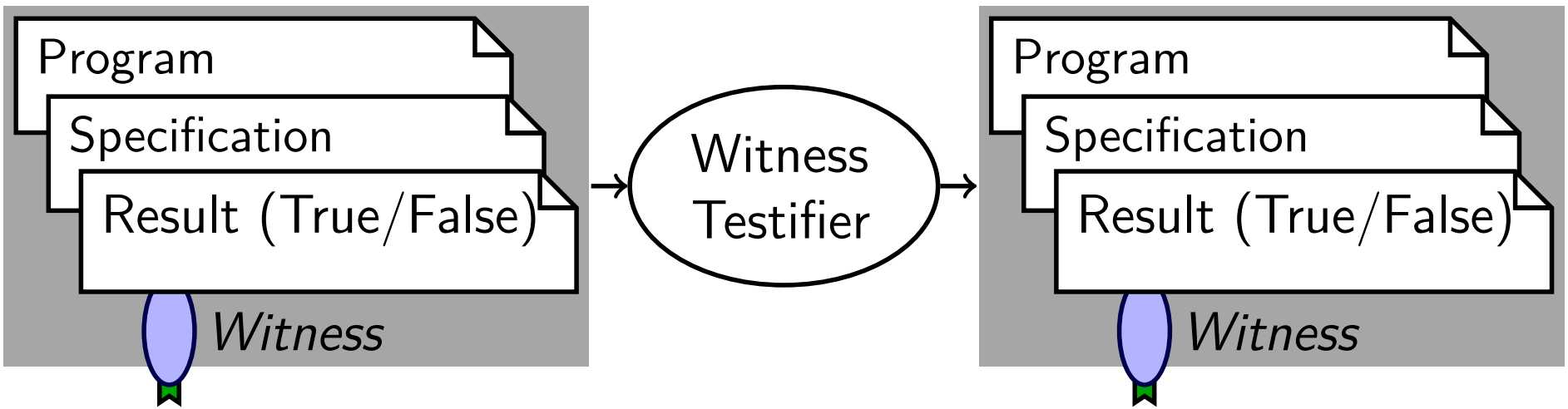
Proc. FSE 2015, 2016 [6, 5]

Witness-Based Result Validation

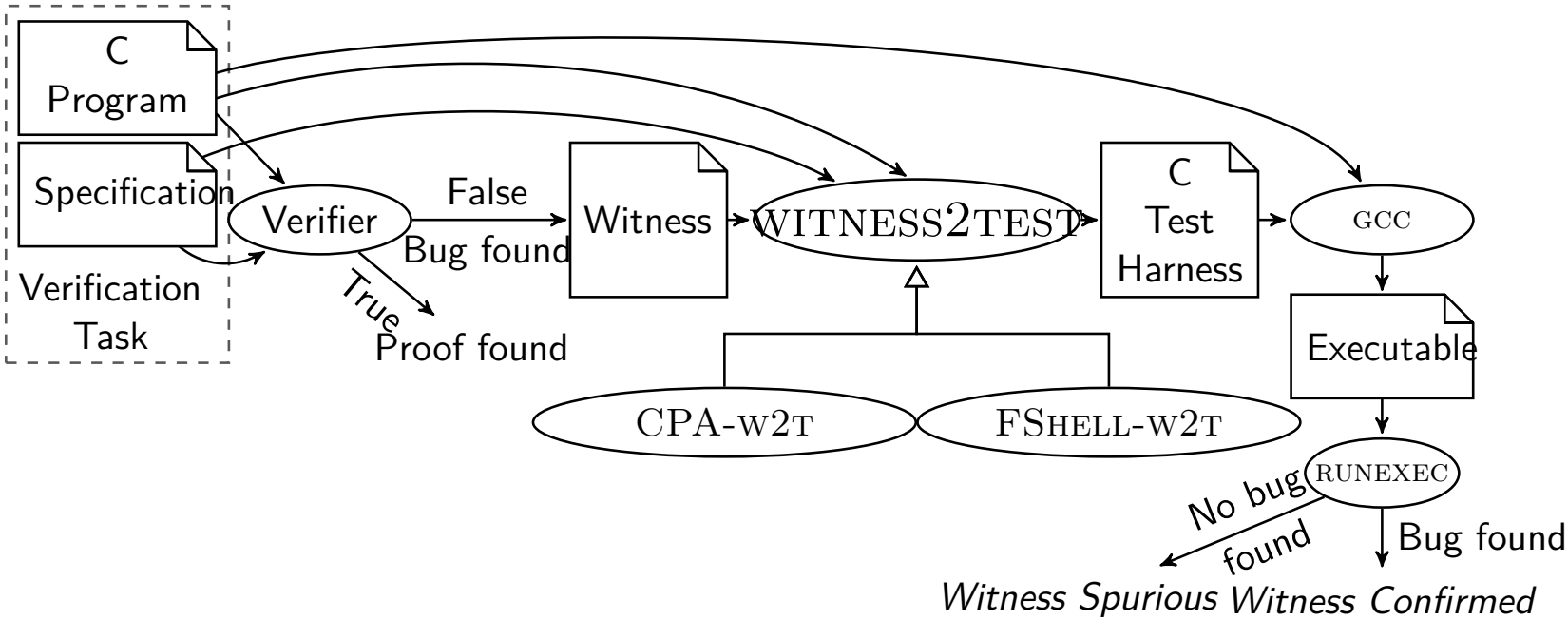


- Validate untrusted results
- Easier than full verification

Stepwise Refinement

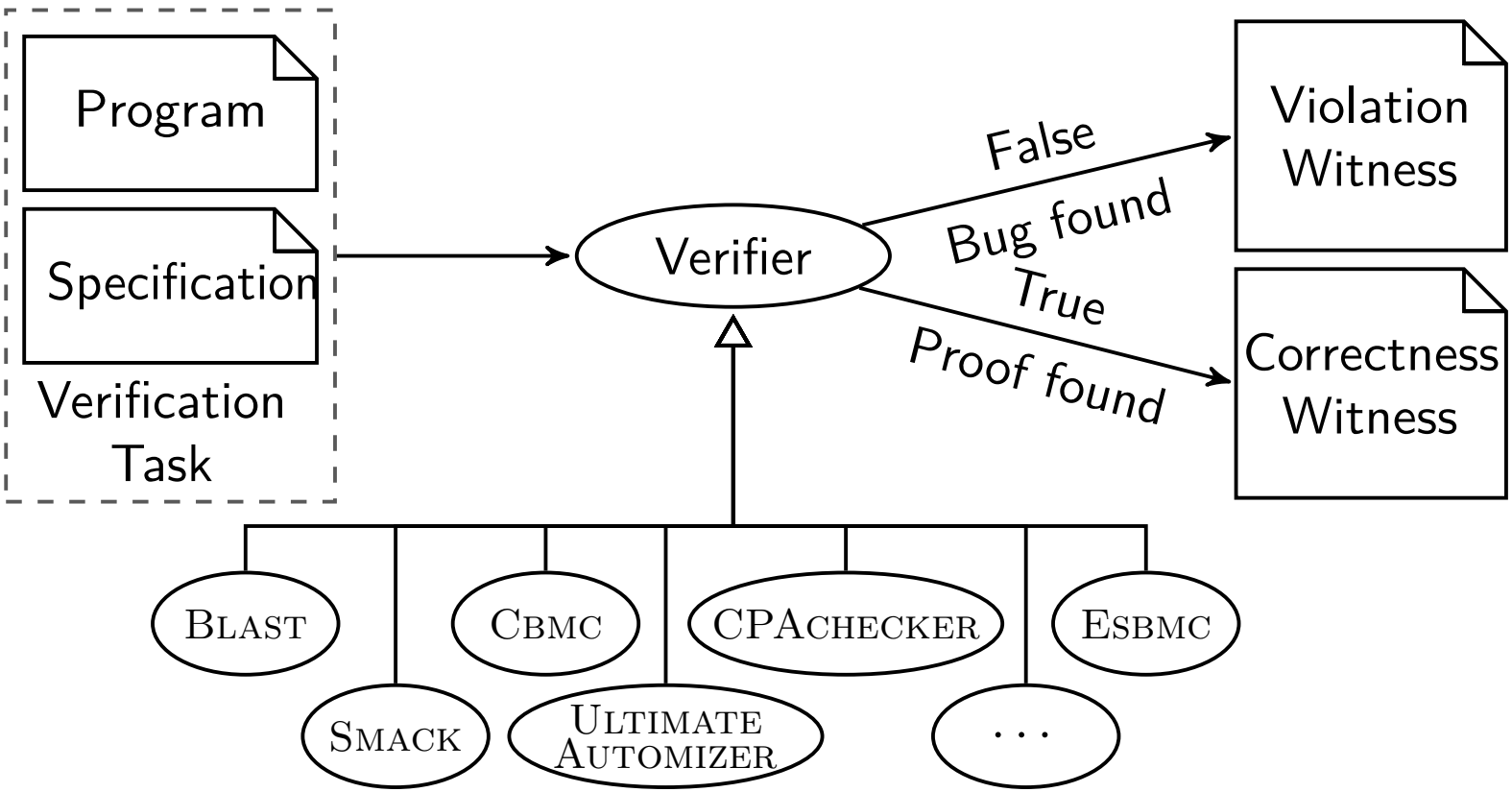


Execution-based Witness Validation

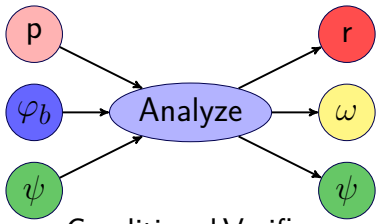


Proc. TAP 2018 [7]
Made "Generating Tests from Counterexamples" more practical
(Proc. ICSE 2004, [4])

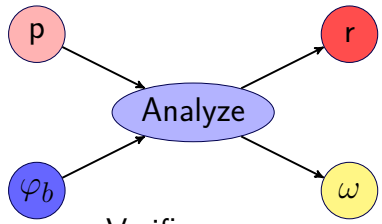
Witness Creation



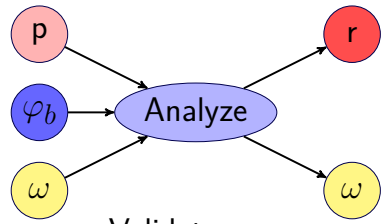
Graphical Visualization of the Coop Framework



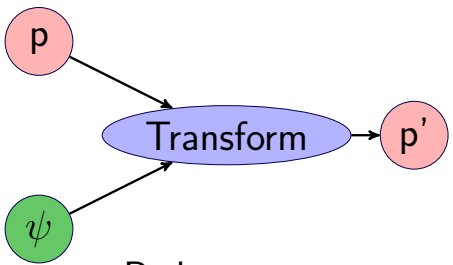
Conditional Verifier



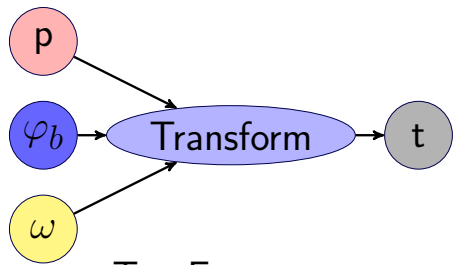
Verifier



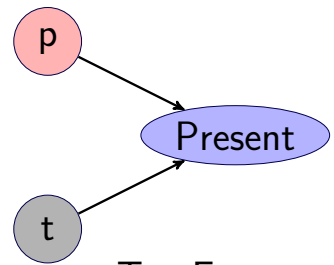
Validator



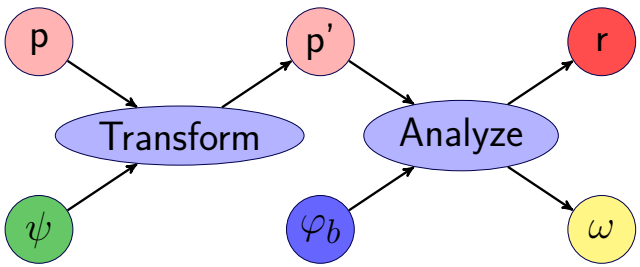
Reducer



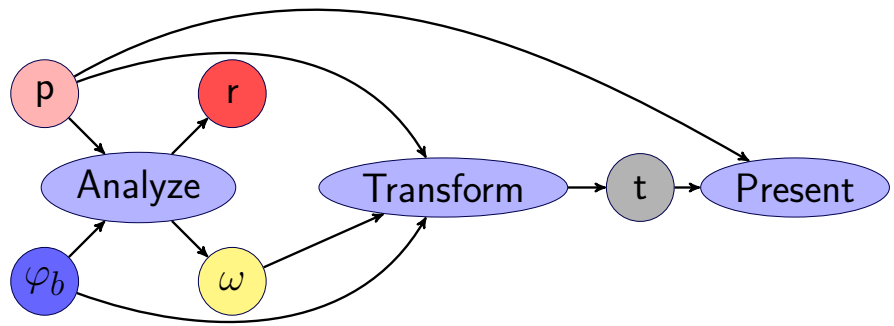
Test Extractor



Test Executor



Conditional Verifier constructed from Reducer and Verifier



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

Conclusion

- Software verification: successful past, bright future
- Competitions solve several problems
- Cooperating combinations are the future

References I


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


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


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


Beyer, D., Chlipala, A.J., Henzinger, T.A., Jhala, R., Majumdar, R.: Generating tests from counterexamples. In: Proc. ICSE. pp. 326–335. IEEE (2004). https://doi.org/10.1109/ICSE.2004.1317455


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


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

References II


Beyer, D., Dangl, M., Lemberger, T., Tautschnig, M.: Tests from witnesses: Execution-based validation of verification results. In: Proc. TAP. pp. 3–23. LNCS 10889, Springer (2018). https://doi.org/10.1007/978-3-319-92994-1_1


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


Beyer, D., Jakobs, M.C.: Coveritest: Cooperative verifier-based testing. In: Proc. FASE. pp. 389–408. LNCS 11424, Springer (2019). https://doi.org/10.1007/978-3-030-16722-6_23


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


Cadar, C., Dunbar, D., Engler, D.R.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. OSDI. pp. 209–224. USENIX Association (2008)


Chowdhury, A.B., Medicherla, R.K., Venkatesh, R.: VeriFuzz: Program aware fuzzing (competition contribution). In: Proc. TACAS. LNCS 11429, Springer (2019)