Software Verification and Verification Witnesses

Dirk Beyer
LMU Munich, Germany

October 11, 2022, at Huawei PhD Forum 2022

Part 1: SV-COMP

11th Competition on Software Verification

Proc. TACAS 2022,
https://doi.org/10.1007/978-3-030-99527-0_20

Motivation - Goals

1. Community suffers from unreproducible results
   $\rightarrow$ Establish set of benchmarks
2. Publicity for tools that are available
   $\rightarrow$ Provide state-of-the-art overview
3. Support the development of verification tools
   $\rightarrow$ Give credits and visibility to developers
4. Establish standards
   $\rightarrow$ Specification language, Witnesses, Benchmark definitions, Validators
5. Train PhD students on benchmarking and reproducibility
6. Provide computing resources to groups that do not have large clusters

Schedule of Sessions

Session 1:

- Competition Report, by organizer
- System Presentations, 7 min by each team
- Short discussion

Session 2:

- Open Jury Meeting, Community Discussion, moderated by organizer

Procedure – Time Line

Three Steps – Three Deadlines:

- Benchmark submission deadline
- System submission
- Notification of results (approved by teams)

Verification Problem

Input:

- C program $\rightarrow$ GNU/ANSI C standard
- Property
  $\rightarrow$ Reachability of error label, of overflows
  $\rightarrow$ Memory safety (inv-deref, inv-free, memleak)
  $\rightarrow$ Termination

Output:

- TRUE + Witness (property holds)
- FALSE + Witness (property does not hold)
- UNKNOWN (failed to compute result)

Environment

Machines (1000 $ consumer machines):

- CPU: 3.4 GHz 64-bit Quad-Core CPU
- RAM: 33 GB
- OS: GNU/Linux (Ubuntu 20.04)

Resource limits:

- 15 GB memory
- 15 min CPU time (consumed 470 days)

Volume: 309 081 verification runs, 1.43 million validation runs
Incl. preruns: 2.85 million verification runs using 19 years, and 16.3 million validation runs using 11 years

Scoring Schema

Common principles: Ranking measure should be

- easy to understand
- reproducible
- computable in isolation for one tool

SV-COMP:

- Ranking measure is the quality of verification work
- Expressed by a community-agreed score
- Tie-breaker is CPU time

Scoring Schema (2022, unchanged)

Reported result	Points	Description
UNKNOWN	0	Failure, out of ressources
FALSE correct	$+1$	Error found and confirmed
FALSE incorrect	-16	False alarm (imprecise analysis)
TRUE correct	$+2$	Proof found and confirmed
TRUE incorrect	-32	Missed bug (unsound analysis)

Fair and Transparent

Jury:

- Team: one member of each participating candidate
- Term: one year (until next participants are determined)

Systems:

- All systems are available in open GitLab repo
- Configurations and Setup in GitLab repository
  $\rightarrow$ Integrity and reproducibility guaranteed

47 Competition Candidates

Qualification:

- 33 qualified, additional 14 hors concours
- 10 result validators, 1 witness linter
- One person can participate with different tools
- One tool can participate with several configurations (frameworks, no tool-name inflation)

Benchmark quality:

- Community effort, documented on GitLab

Role of organizer:

- Just service: Advice, Technical Help, Executing Runs

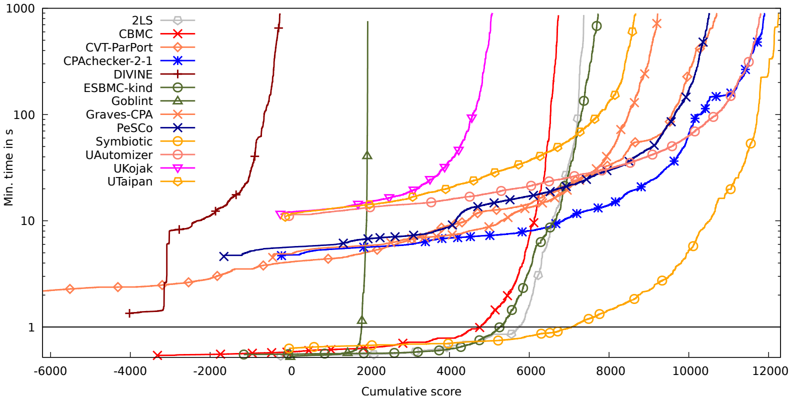
Benchmark Sets

- Everybody can submit benchmarks (conditions apply)
- Eight categories when closed (scores normalized):
  - Reachability: 5400 tasks
  - Memory Safety: 3321 tasks
  - Concurrency: 763 tasks
  - NoOverflows: 454 tasks
  - Termination: 2293 tasks
  - Software Systems: 3417 tasks
  - Overall: 15648 tasks
  - Java: 586 tasks

Replicability

- SV-Benchmarks:
  https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks
- SV-COMP Setup:
  https://gitlab.com/sosy-lab/sv-comp/bench-defs
- Resource Measurement and Process Control:
  https://github.com/sosy-lab/benchexec
- Archives:
  https://gitlab.com/sosy-lab/sv-comp/archives-2022
- Witnesses:
  https://doi.org/10.5281/zenodo.5838498

Results – Example: Overall



Impact / Achievements

- Large benchmark set of verification tasks
  → established and used in many papers
  for experimental evaluation
- Good overview over state-of-the art
  → covers model checking and program analysis
- Participants have an archived track record
  of their achievements
- Infrastructure and technology for
  controlling the benchmark runs (cf. StarExec)

[Competition Report and System Descriptions
are archived in Proceedings TACAS 2022]
https://doi.org/10.1007/978-3-030-99527-0_20

Alternative Rankings — Definitions

- Correct Verifiers — Low Failure Rate:

  $$\frac{\text{number of incorrect results}}{\text{total score}}$$

  with unit E/sp.

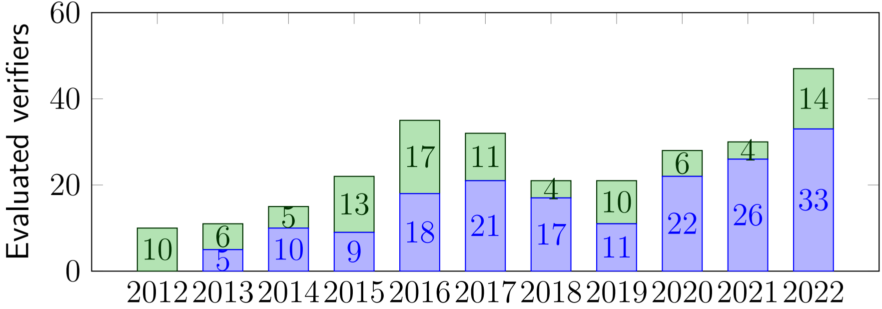
- Green Verifiers — Low Energy Consumption:

  $$\frac{\text{total CPU energy}}{\text{total score}}$$

  with the unit J/sp.

Number of Participants

Number of evaluated verifiers for each year
(first-time participants on top)



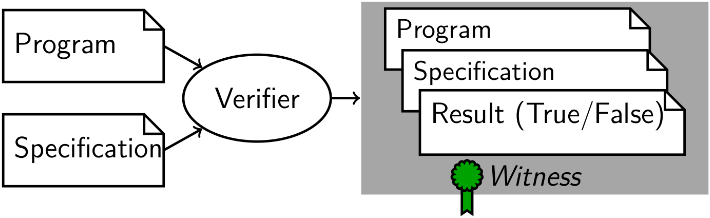
Different Techniques

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	✓	✓									✓		✓	✓	✓			
VeriAbs	✓			✓	✓		✓	✓										
VeriFuzz				✓				✓										✓
VIAP																		
Yogar-CBMC	✓			✓							✓		✓			✓		
Yogar-CBMC-Par.	✓			✓							✓		✓			✓		

Competition Report [1]
https://doi.org/10.1007/978-3-030-17502-3_9

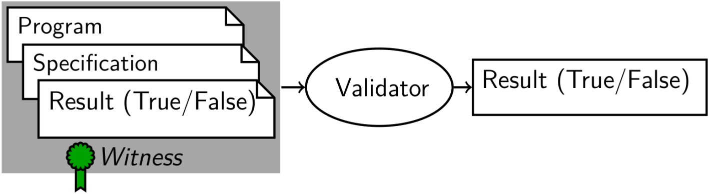
Part 2: Software Verification with Witnesses

Witnesses are an important interface between tools.



[5, Proc. FSE 2015] [4, Proc. FSE 2016]

Witness Validation



- Validate untrusted results
- Easier than full verification

Example Combination (in DSL CoVeriTeam)

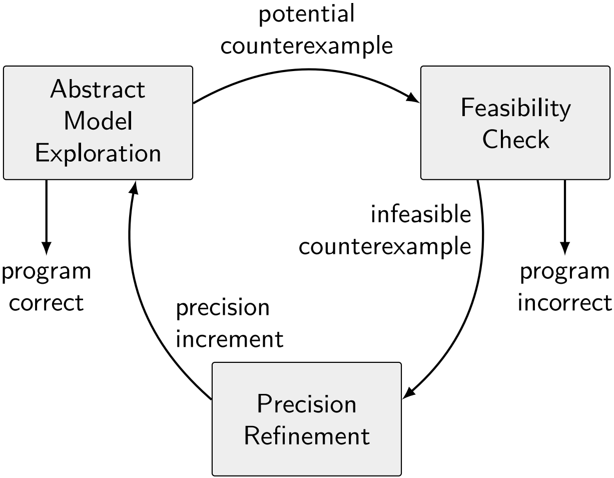
CoVeriTeam: Language and Tool [12, Proc. TACAS 2022]

Algorithm 1 Witness Validation [5, 4]

Input: Program p, Specification s
Output: Verdict

verifier := Verifier("Ultimate Automizer")
validator := Validator("CPAchecker")
result := verifier.verify(p, s)
if result.verdict $\in$ {TRUE, FALSE} **then**
    result = validator.validate (p, s, result.witness)
return (result.verdict, result.witness)

CEGAR

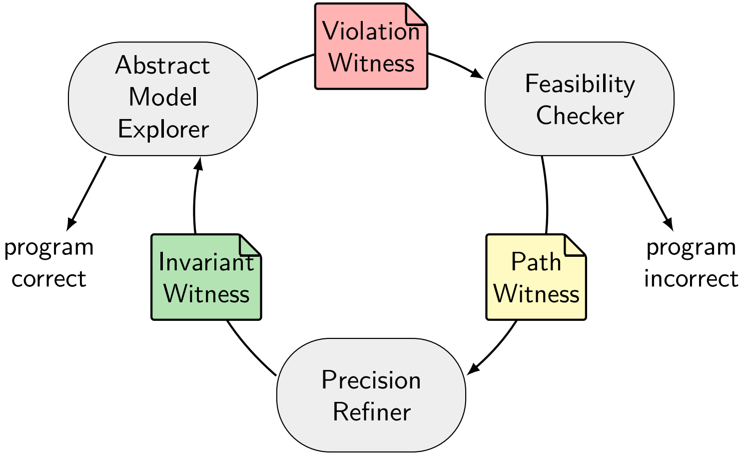

potential
counterexample
Abstract
Model
Exploration
Feasibility
Check
infeasible
counterexample
program
correct
program
incorrect
precision
increment
Precision
Refinement

Modularization of CEGAR

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

$\Rightarrow$ Modular reformulation

Workflow of modular CEGAR



Proc. ICSE 2022 [8]

Interactive and Automatic Methods

- How to achieve cooperation between automatic and interactive verifiers?
- Idea: Try to use existing interfaces for information exchange

```
//@ensures \return==0;
int main() {
  unsigned int x = 0;
  unsigned int y = 0;
  //@loop invariant x==y;
  while (nondet_int()) {
    x++;
    //@assert x==y+1;
    y++;
  }
  assert(x==y);
  return 0;
}
```

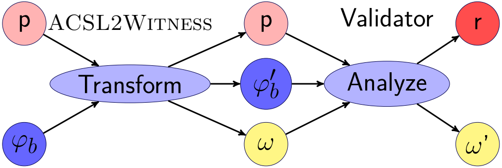
ACSL-annotated program, as used by FRAMA-C

```
...
<node id="q1">
<data key="invariant">( y == x )</data>
<data key="invariant.scope">main</data>
</node>
<edge source="q0" target="q1">
<data key="enterLoopHead">true</data>
<data key="startline">6</data>
<data key="endline">6</data>
<data key="startoffset">157</data>
<data key="endoffset">165</data>
</edge>
...
```

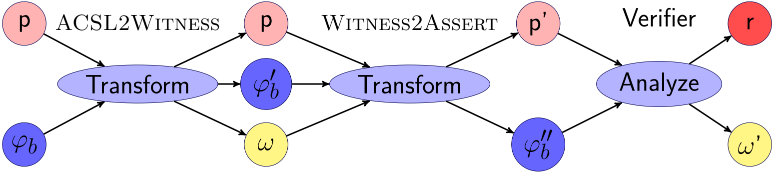
GraphML-based witness automaton generated by automatic verifiers

From Components: Construct Interactive Verifiers

- Turn a witness validator into an interactive verifier:



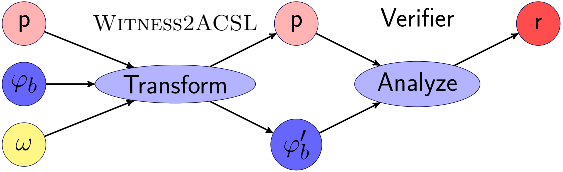
- Turn an automatic verifier into an interactive verifier::



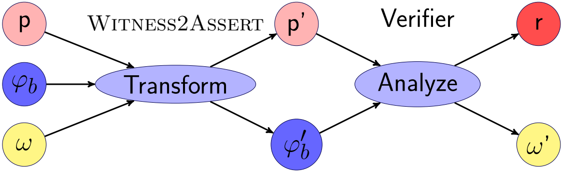
- Annotating in ACSL is more human-readable than witness automata
- Works for a wide range of automatic verifiers/validators

Component Framework: Constructing Validators

- Turn an interactive verifier (Frama-C) into a validator:



- Turn an automatic verifier into a validator:



All Implemented in CPACHECKER [14]

- Included Concepts:
  - CEGAR [20]   Interpolation [17, 7]
  - Configurable Program Analysis [10, 11]
  - Adjustable-block encoding [15]
  - Conditional model checking [9]
  - Verification witnesses [5, 4]
  - Various abstract domains: predicates, intervals, BDDs, octagons, explicit values
- Available analyses approaches:
  - Predicate abstraction [2, 15, 11, 18]
  - IMPACT algorithm [22, 19, 7]
  - Bounded model checking [21, 7]
  - k-Induction [6, 7]
  - IC3/Property-directed reachability [3]
  - Explicit-state model checking [17]
  - Interpolation-based model checking [16]

Simple Combination without Cooperation

Often, even simple combinations help!

Portfolio construction using off-the-shelf verification tools [13, 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)

With Nian-Ze Lee and Po-Chun Chien:

- inject invariants (in k-induction, IMC, ISMC)
- parallel portfolio

A lot of improvements are (trivially) possible.

Conclusion

- Mature research area with competition SV-COMP
- Verification Witnesses as Interfaces
- Combinations and Cooperation

References I

[1] 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

[2] 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

[3] 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

[4] 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

[5] 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

[6] 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

References II

[7] 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

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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

References III

[13] 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

[14] 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

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

[16] 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

[17] 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

[18] 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

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

References IV

[20] 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

[21] 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

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