Certifying Software Verification

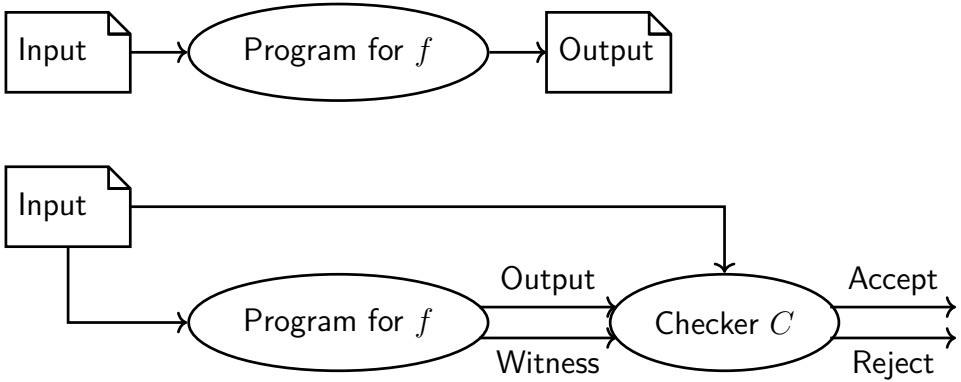
Dirk Beyer
LMU Munich, Germany

SoSy-Lab @ LMU Munich

June 3, 2025, at Dagstuhl on Certifying Algorithms

Certifying Algorithms

Proposed in 2011 by R. M. McConnell, K. Mehlhorn, S. Näher, P. Schweitzer [14]



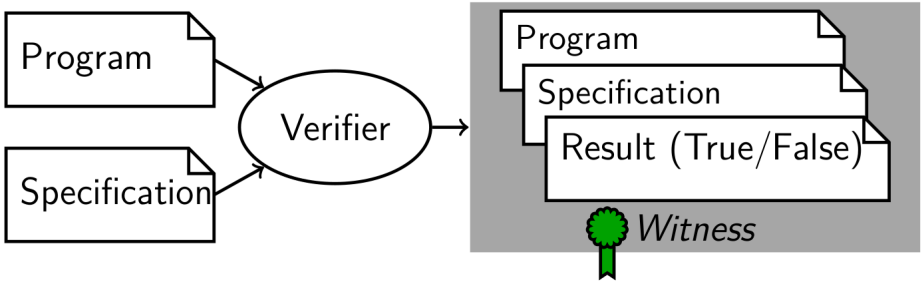
"Certifying algorithms are a preferred kind of algorithm. They prove their work and they are easier to implement reliably. Their widespread use would greatly enhance the reliability of algorithmic software."

Latest Literature

Software Verification Witnesses 2.0, Proc. SPIN 2024,
doi:10.1007/978-3-031-66149-5_11

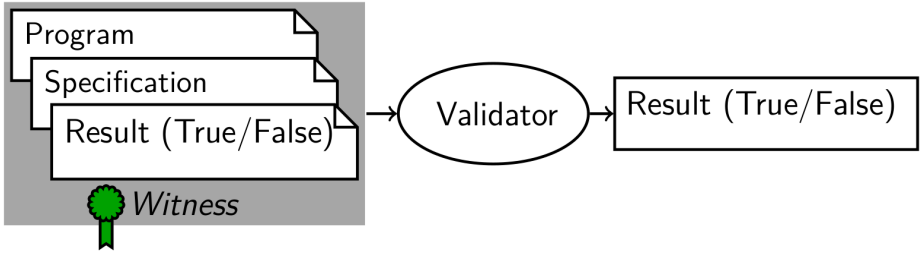
Software-Verification Witnesses

Witnesses are an important interface between tools.



[6, Proc. FSE 2015] [4, Proc. FSE 2016] [5, TOSEM 2022]

Witness-Based Result Validation



- Validate untrusted results
- Reestablish proof of correctness or violation
- Easier than full verification

Verification and Validation

Given program P and specification φ

- Verification: **prove** that $P \models \varphi$
  (mainly invariant construction)
- Validation with witness w: **re-prove** that $P \models \varphi$

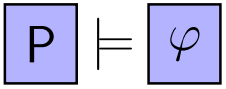
Untrusted methods (for example, AI) can be used to

- **write** programs
- **suggest** invariants for programs

and wrong guesses are no problem thanks to validation.

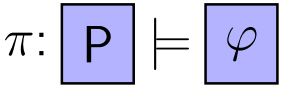
Correctness Witnesses

Program P, specification φ, proof π



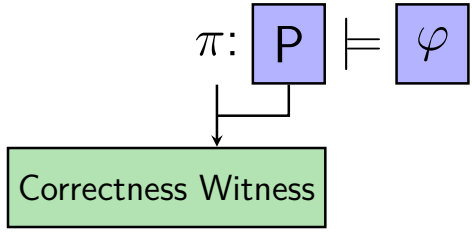
Correctness Witnesses

Program P, specification φ, proof π



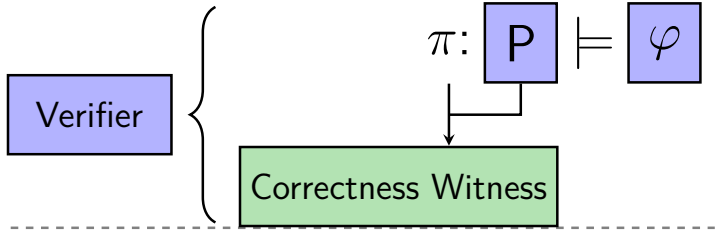
Correctness Witnesses

Program P, specification φ, proof π



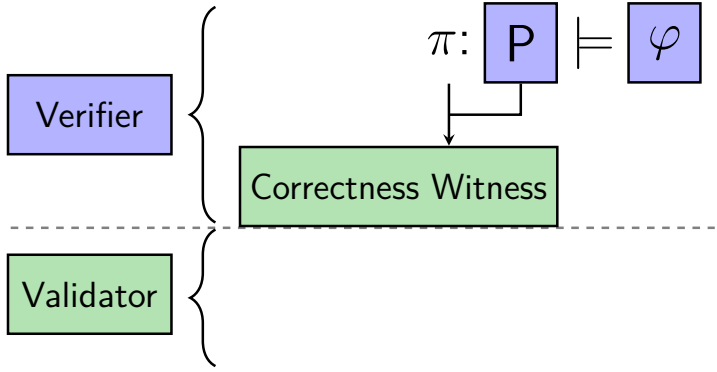
Correctness Witnesses

Program P, specification φ, proof π



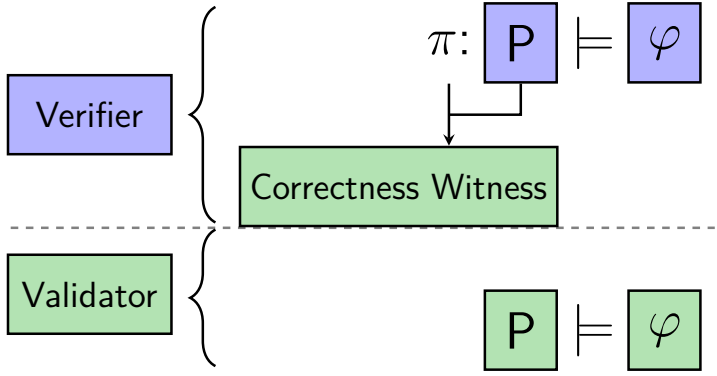
Correctness Witnesses

Program P, specification φ, proof π



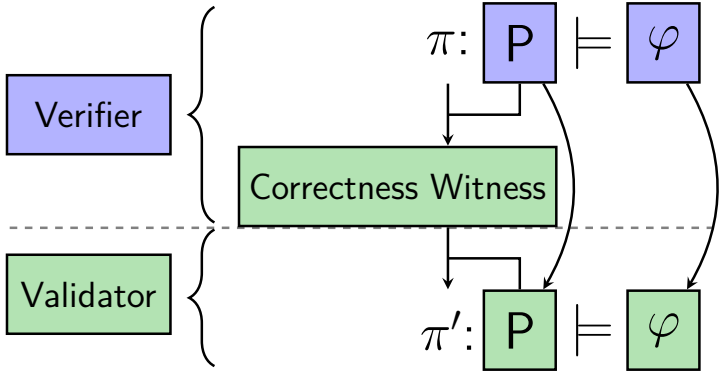
Correctness Witnesses

Program P, specification φ, proof π



Correctness Witnesses

Program P, specification φ, proof π



Example Program and Correctness Witness

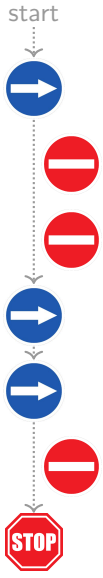
Program:

```
int main() {
  unsigned char n = __nondet_uchar();
  if (n == 0) {
    return 0;
  }
  unsigned char v = 0;
  unsigned int  s = 0;
  unsigned int  i = 0;
  while (i < n) {
    v = __nondet_uchar();
    s += v;
    ++i;
  }
  if (s < v) {
    reach_error();
    return 1;
  }
  if (s > 65025) {
    reach_error();
    return 1;
  }
  return 0;
}
```

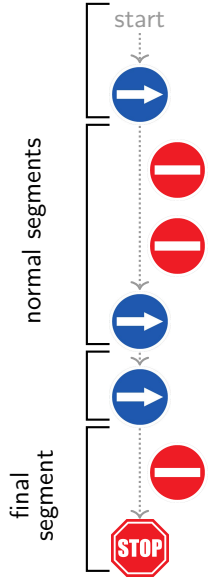
Correctness Witness (format v2.0 [1]):

```
content:
  - invariant:
      type: loop_invariant
      location:
        file_name: "inv-a.c"
        line: 12
        column: 1
        function: main
      value: "s <= i*255 && 0 <= i && i <= 255 && n <= 255"
      format: c_expression
```

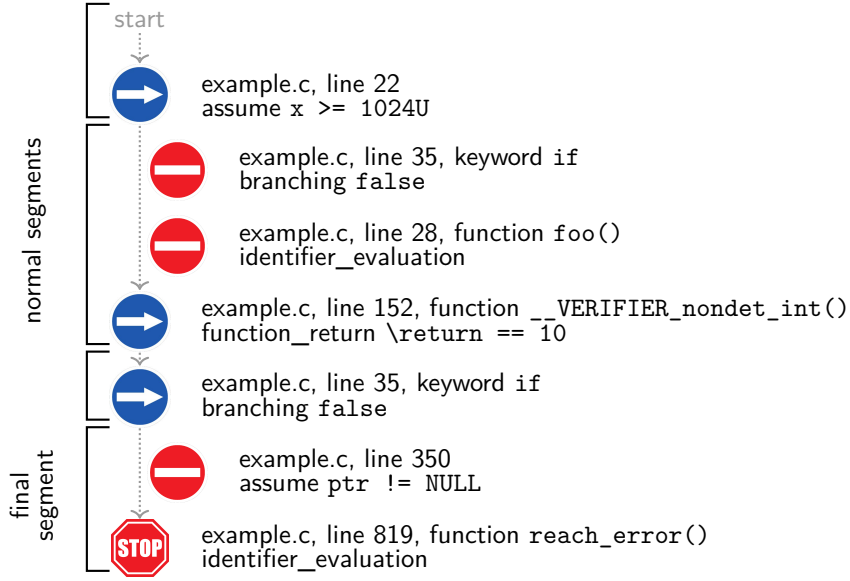
Example of a Violation Witness — Illustration



Example of a Violation Witness — Illustration



Example of a Violation Witness — Illustration



Example of a Violation Witness — Text Notation

example.c, line 35, keyword `if`
branching `false`

example.c, line 28, function `foo()`
`identifier_evaluation`

example.c, line 152,
function `__VERIFIER_nondet_int()`
`function_return \return == 10`

```
- segment:
  - waypoint:
    action: avoid
    type: branching
    location:
      file_name: example.c
      line: 35
    constraint:
      string: false
  - waypoint:
    action: avoid
    type: identifier_evaluation
    location:
      file_name: example.c
      line: 28
  - waypoint:
    action: follow
    type: function_return
    location:
      file_name: example.c
      line: 152
    constraint:
      string: \return == 10
```

State of the Art Regarding Tool Support

- 62 verifiers participating in SV-COMP support verification witnesses (versions 1.0 and 2.0, C and Java)
- 18 validators exist for C and Java
- 4 formats for witnesses exist (1.0 and 2.0, correctness and validation)
- Competition on Software Verification (SV-COMP) has a validation track

History

- 2011: Certifying Algorithms [14, J. CSR '11]
- 2013: Witnesses for Software Verification proposed [11, Proc. SPIN '13]
- 2014: Competition on Software Verification adopts Witnesses
- 2015: Violation Witnesses 1.0 [6, Proc. FSE '15]
- 2016: Correctness Witnesses 1.0 [4, Proc. FSE '16]
- 2018: Execution-Based Witness Validation [7, Proc. TAP '18]
- 2016: Verification-Aided Debugging [3, Proc. CAV '16]
- 2020: Multi-threaded Programs [8, 12, ISoLA '20, VMCAI '25]
- 2024: Verification Witnesses 2.0 [1, Proc. SPIN '24]
- 2025: Function Contracts [13, TR '25]

Notions

- Certifying vs. Certified
- Certificates vs. Proofs
- Certificate checkers must check original problem
  (in contrast to proof checkers, which check only consistency of proof)
- Interesting: How much information to store?
  (also mentioned by Katalin for SAT and Hanial for SMT,
  which clauses are worth storing)

A Wish to the SMT Community

- Please provide proofs in a standard exchange format
- We maintain JavaSMT [2]
  `https://gitlab.com/sosy-lab/software/java-smt`
- We could implement solver-independent interpolation

Conclusion

- Standard exchange format exists (since 2014)
- Supported by many verification and validation tools (since SV-COMP 2015)
- Witnesses are used for communicating between tools [9]
- Witness Validation with Deductive Verification are similar [10]

References I

[1] Ayaziová, P., Beyer, D., Lingsch-Rosenfeld, M., Spiessl, M., Strejček, J.: Software verification witnesses 2.0. In: Proc. SPIN. pp. 184–203. LNCS 14624, Springer (2024). https://doi.org/10.1007/978-3-031-66149-5_11

[2] Baier, D., Beyer, D., Friedberger, K.: JavaSMT 3: Interacting with SMT solvers in Java. In: Proc. CAV. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_9

[3] Beyer, D., Dangl, M.: Verification-aided debugging: An interactive web-service for exploring error witnesses. In: Proc. CAV (2). pp. 502–509. LNCS 9780, Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_28

[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., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022). https://doi.org/10.1145/3477579

[6] 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

[7] 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

References II

[8] Beyer, D., Friedberger, K.: Violation witnesses and result validation for multi-threaded programs. In: Proc. ISoLA (1). pp. 449–470. LNCS 12476, Springer (2020). https://doi.org/10.1007/978-3-030-61362-4_26

[9] 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

[10] Beyer, D., Spiessl, M., Umbricht, S.: Cooperation between automatic and interactive software verifiers. In: Proc. SEFM. p. 111–128. LNCS 13550, Springer (2022). https://doi.org/10.1007/978-3-031-17108-6_7

[11] Beyer, D., Wendler, P.: Reuse of verification results: Conditional model checking, precision reuse, and verification witnesses. In: Proc. SPIN. pp. 1–17. LNCS 7976, Springer (2013). https://doi.org/10.1007/978-3-642-39176-7_1

[12] Erhard, J., Bentele, M., Heizmann, M., Klumpp, D., Saan, S., Schüssele, F., Schwarz, M., Seidl, H., Tilscher, S., Vojdani, V.: Correctness witnesses for concurrent programs: Bridging the semantic divide with ghosts. In: Proc. VMCAI, Part I. pp. 74–100. LNCS 15529, Springer (2025). https://doi.org/10.1007/978-3-031-82700-6_4

[13] Heizmann, M., Klumpp, D., Lingsch-Rosenfeld, M., Schüssele, F.: Correctness witnesses with function contracts. arXiv/CoRR **2501**(12313) (July 2025). https://doi.org/10.48550/ARXIV.2501.12313

[14] McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. Computer Science Review **5**(2), 119–161 (2011). https://doi.org/10.1016/j.cosrev.2010.09.009