Part 4: Cooperative Verification by Correctness Witnesses: Exchanging Verification Results between Verifiers

Dirk Beyer
Joint work with Matthias Dangl, Daniel Dietsch, and Matthias Heizmann
Software Verification

Program

Verifier

Specification

Result (True/False)
Software Verification with Witnesses

Diagram:
- Program
- Specification
- Verifier
- Specification
- Result (True/False)
- Witness
Witness Validation

- Validate untrusted results
- Easier than full verification
Stepwise Refinement

Witness
Program
Specification
Result (True/False)

Witness
Program
Specification
Result (True/False)

Witness

Witness

Witness
Violation Witnesses

Violation Witness
Violation Witnesses

Violation Witness

Abstract Counterex.

Test Case
Search-Space Reduction for Stepwise Witness Refinement
Search-Space Reduction for Stepwise Witness Refinement

Search space
Search-Space Reduction for Stepwise Witness Refinement
Search-Space Reduction for Stepwise Witness Refinement
Search-Space Reduction for Stepwise Witness Refinement
Correctness: State of the Art

1. Rarely any additional information
Correctness: State of the Art

1. Rarely any additional information
2. Not human readable
Correctness: State of the Art

1. Rarely any additional information
2. Not human readable
3. Not easily exchangeable across tools
Open Problems

1. **Standardized way** to document verification results to enhance engineering processes **required**
Open Problems

1. **Standardized way** to document verification results to enhance engineering processes **required**

2. **Difficult to establish trust** in results from an untrusted verifier
Open Problems

1. **Standardized way** to document verification results to enhance engineering processes **required**

2. **Difficult to establish trust** in results from an untrusted verifier

3. Potential for synergies between tools and techniques is **left unused**
Verification Witnesses: Classification

FSE’15

Violation Witness

Abstract Counterex.
Test Case

Taleghani & Atlee, ASE’10
Necula, POPL’97
Verification Witnesses: Classification

Witness

FSE’15

Violation Witness

Correctness Witness

Abstract Counterex.

Test Case
Verification Witnesses: Classification

Witness

FSE'15

Violation Witness

Abstract Counterex.

Test Case

Correctness Witness

SCC

PCC

Taleghani & Atlee, ASE'10  Necula, POPL'97
Verification Witnesses: Classification

Witness

FSE'15

Violation Witness

Abstract Counterex.

Test Case

Correctness Witness

SCC

PCC

Taleghani & Atlee, ASE’10  Necula, POPL’97
Correctness Witnesses and Proof Certificates

- **Full proofs** seem nice, but in practice become **too large**
- Witnesses **support**, but do **not enforce** full proofs
- **Instead**, correctness witnesses may also represent **proof sketches**
Correctness Witnesses

\[ P \models \varphi \]
Correctness Witnesses

\[ \pi : \begin{array}{c} P \\ \hline \end{array} \models \varphi \]
Correctness Witnesses

\[ \pi : \mathcal{P} \models \varphi \]

Correctness Witness

Dirk Beyer
Correctness Witnesses

Verifier

π: \[ P \vdash \phi \]

Correctness Witness

Validator

Testifier

π′: \[ P \vdash \phi \]
Correctness Witnesses

\[ \pi : P \Rightarrow \varphi \]

Verifier \rightarrow Correctness Witness \rightarrow Validator
Correctness Witnesses

Verifier

Correctness Witness

Validator

\( \pi : \mathcal{P} \models \varphi \)

\( \mathcal{P} \models \varphi \)
Correctness Witnesses

Verifier

Correctness Witness

Validator

\[ \pi : P \models \varphi \]

\[ \pi' : P' \models \varphi \]
Correctness Witnesses

Verifier

Testifier

Correctness Witness

\[ \pi : P \Rightarrow \varphi \]

Correctness Witness 2

\[ \pi' : P \Rightarrow \varphi \]
Witness Automata

- Express witness as automaton
Witness Automata

- Express witness as **automaton**
- Witness Validation **matches** the **witness** to the **program**
Witness Automata

- Express witness as **automaton**
- Witness Validation **matches** the **witness** to the **program**
- **Decoupled from** specific verification **techniques** and implementations
Witness Automata

- Express witness as **automaton**
- Witness Validation **matches** the **witness** to the **program**
- **Decoupled from** specific verification **techniques** and implementations
- One **common exchange format** for violation witnesses and correctness witnesses
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```

```
true
```

3, enterLoopHead:

```
x = y
```

4, then:

```
true
```

5, else:

```
```

6, enterLoopHead:

```
```

7, o/w

8, o/w

9, o/w

10, o/w

11, o/w
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Produce and Consuming Witnesses: SV-COMP

Table 8: Confirmation rate of witnesses

<table>
<thead>
<tr>
<th>Result</th>
<th>Total</th>
<th>Confirmed</th>
<th>Unconfirmed</th>
<th>Total</th>
<th>Confirmed</th>
<th>Unconfirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automizer</td>
<td>3558</td>
<td>3481</td>
<td>77</td>
<td>1173</td>
<td>1121</td>
<td>52</td>
</tr>
<tr>
<td>SMACK</td>
<td>2947</td>
<td>2695</td>
<td>252</td>
<td>1929</td>
<td>1768</td>
<td>161</td>
</tr>
<tr>
<td>CPA-SEQ</td>
<td>3357</td>
<td>3078</td>
<td>279</td>
<td>2342</td>
<td>2315</td>
<td>27</td>
</tr>
</tbody>
</table>

Verifiable Witnesses. For SV-COMP, it is not sufficient to answer with just TRUE or FALSE: each answer must be accompanied by a verification witness. For correctness witnesses, an unconfirmed answer TRUE was still accepted, but was assigned only 1 point instead of 2 (cf. Table [2]). All verifiers in categories that required witness validation support the common exchange format for violation and correctness witnesses. We used the two independently developed witness validators that are integrated in CPAchecker and UAutomizer [7, 8].
Stepwise Refinement: Classification

Verification

Result?

Verification

Condition

Unknown

Refine?

Output

condition

No

Conditional

MC

Yes

Violation Witness

False

Refine?

Violation

Refinement

Yes

Output

witness

No

Result?

False

Output

"Unknown"

Rejected

Correctness Witness

True

Refine?

Output

witness

No

Correctness

Refinement

Yes

Result?

True

Output

"Unknown"

Rejected
Stepwise Refinement: Classification

- Verification
- Result?
  - Unknown
  - Verification Condition
  - Refine?
  - Output condition
    - No
    - Conditional MC
  - Output witness
    - No
  - Result?
    - False
    - Output "Unknown"
      - Rejected
    - Correctness Witness
      - True
      - Output witness
        - No
        - Correctness Refinement
          - Yes
    - Result?
      - True
      - Output "Unknown"
        - Rejected
Stepwise Refinement: Classification

Verification

Result?

Verification Condition

Refine?

Yes

Output condition

No

Conditional MC
Stepwise Refinement: Classification

Violation Witness

Verification

Result?

False

Verification Condition

Refine?

Yes

Output condition

No

Conditional MC

Unknown

Dirk Beyer
Stepwise Refinement: Classification

- **Violation Witness**
  - Refine? (Yes → Violation Refinement; No → Verification)
  - Result?
    - False → Output "Unknown"
    - Unknown → Output condition
    - True → Correctness Witness
      - Refine? (Yes → Conditional MC; No → Rejected)
      - Result?
        - False → Output "Unknown"
        - Yes → Correctness Refinement
Stepwise Refinement: Classification

- Violation Witness
  - Refine? (Yes) → Violation Refinement
  - Refine? (No) → Output witness
  - Result? (False) → Rejected
  - Result? (True) → Output "Unknown"

- Verification
  - Result? (False) → False
  - Result? (True) → Correctness Witness
  - Refine? (Yes) → Output condition
  - Refine? (No) → Conditional MC

- Correctness Witness
  - True
  - False
Conclusion

Correctness-Witnesses...

1. are easy to implement for verifiers that already support violation witnesses
Conclusion

Correctness-Witnesses...

1. are easy to implement for verifiers that already support violation witnesses

2. enable information exchange across different software verifiers
Conclusion

Correctness-Witnesses...

1. are easy to implement for verifiers that already support violation witnesses

2. enable information exchange across different software verifiers

3. efficiently increase confidence in results by validation