# Correctness Witnesses: Exchanging Verification Results between Verifiers

**Dirk Beyer, Matthias Dangl, Daniel Dietsch, and Matthias Heizmann**

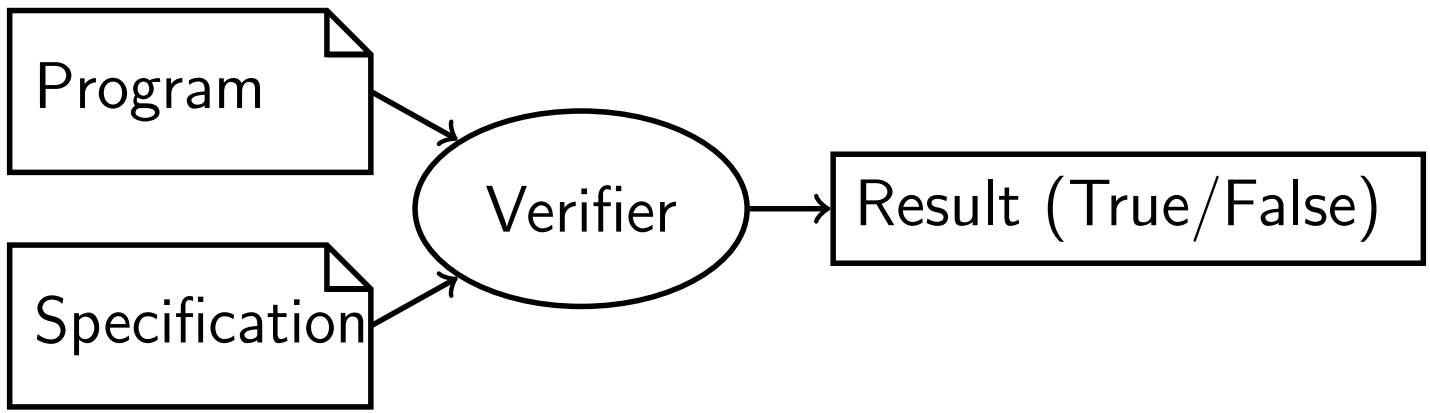



UNIVERSITÄT PASSAU
*Fakultät für Informatik und Mathematik*

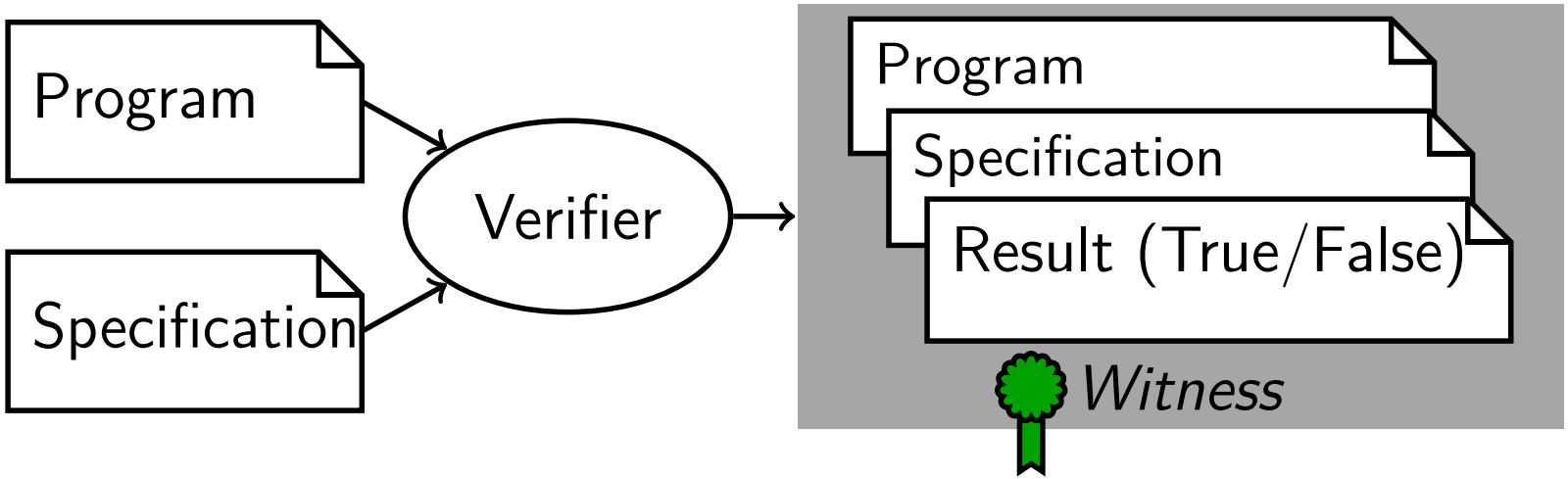
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

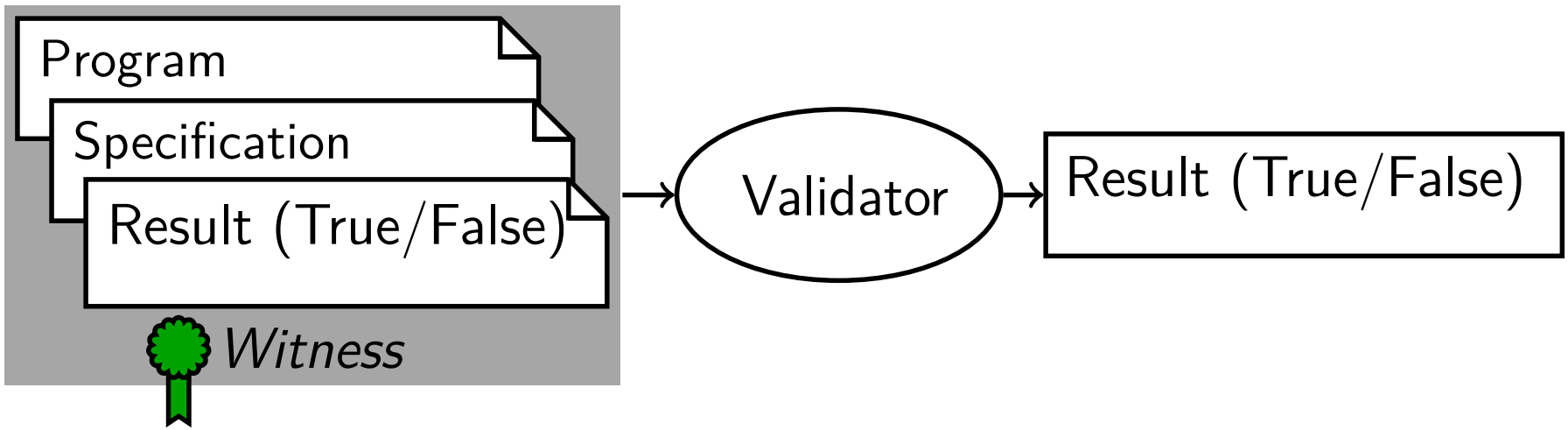
# Software Verification



# Software Verification with Witnesses

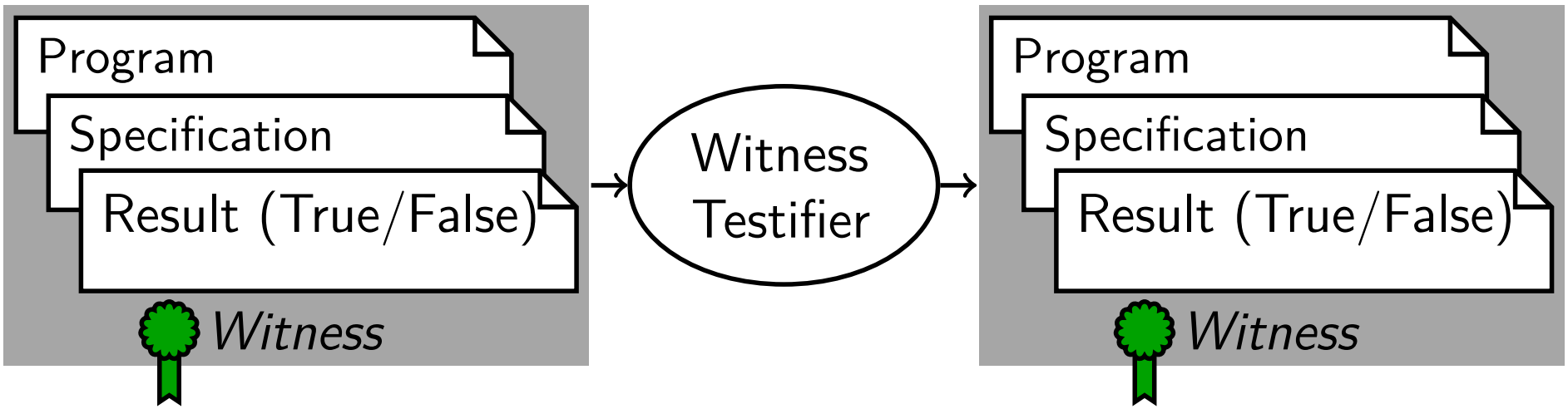


# Witness Validation



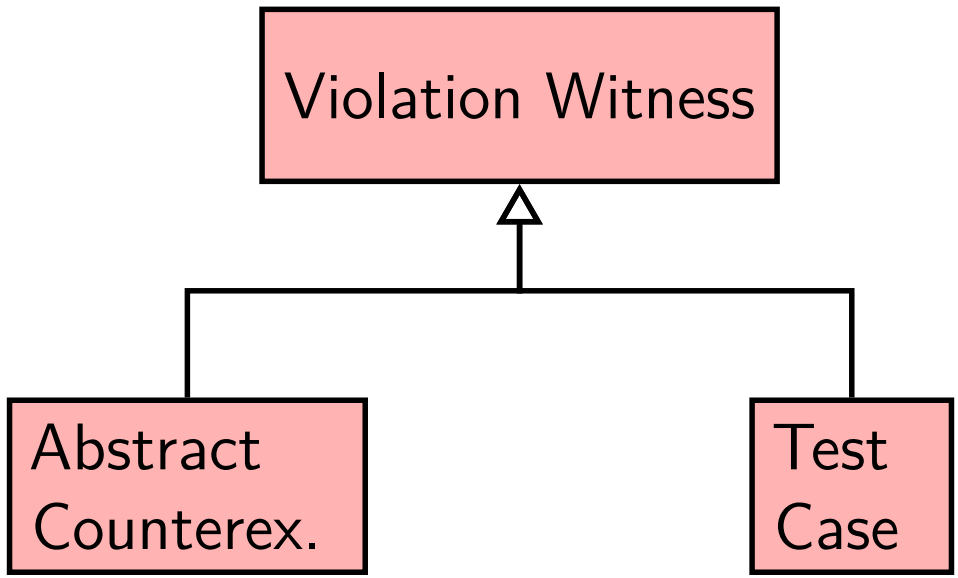

- Validate untrusted results
- Easier than full verification

# Stepwise Testification



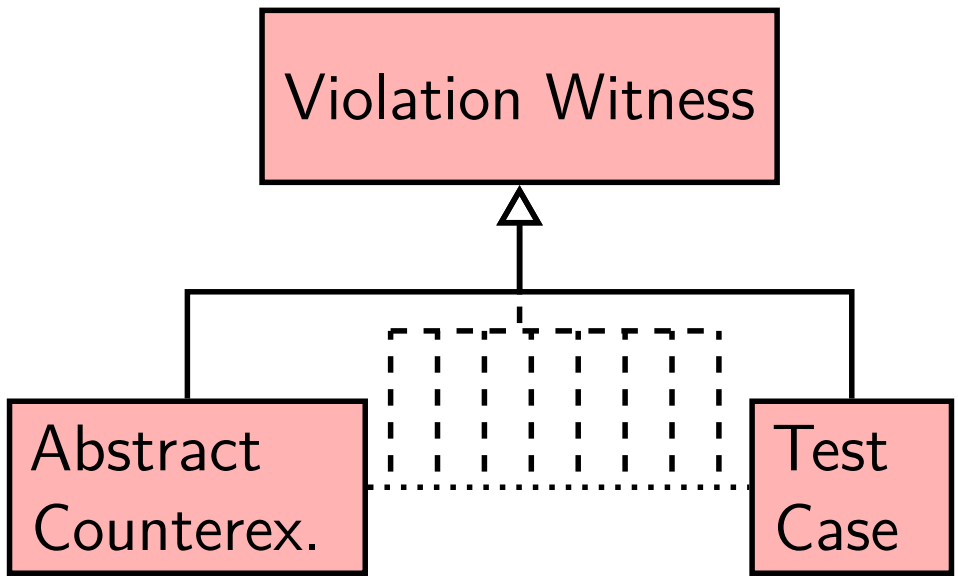
# Violation Witnesses

# Violation Witnesses

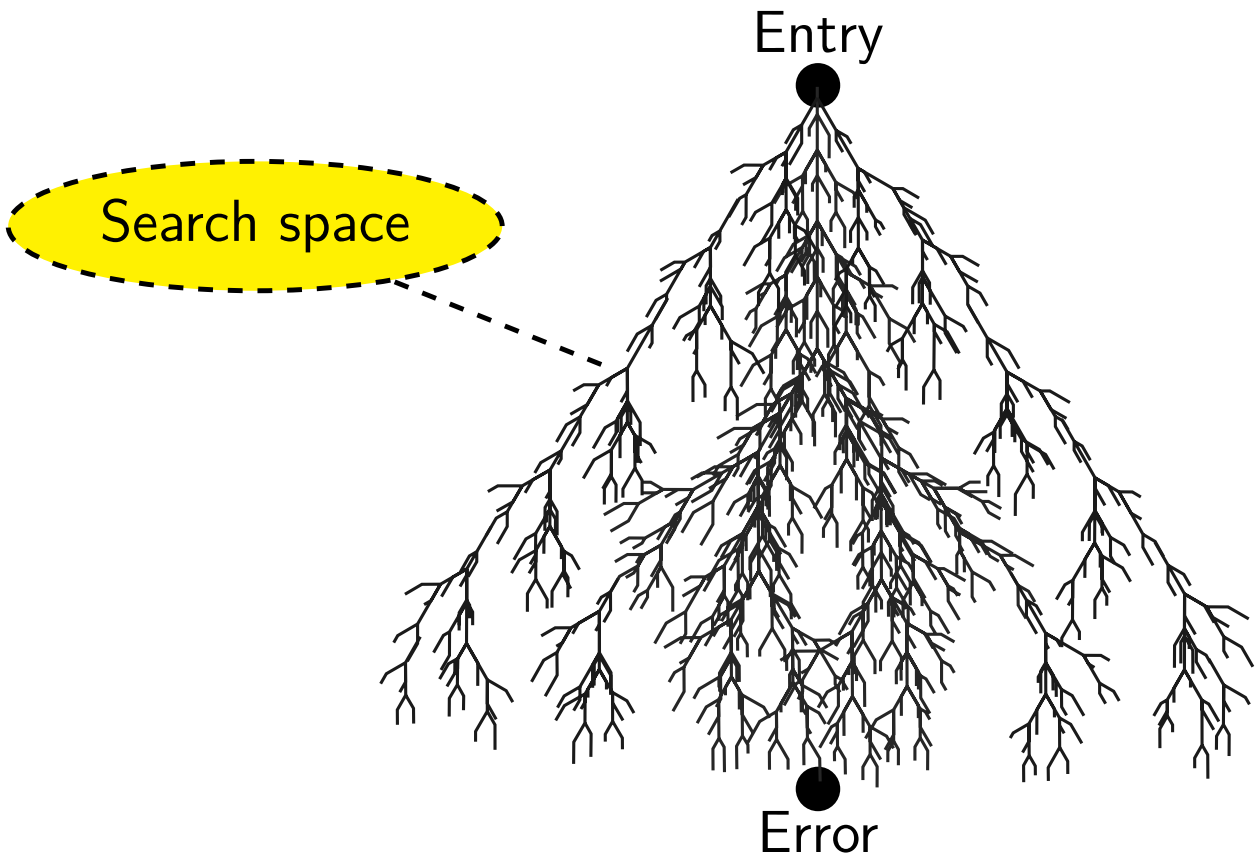


# Violation Witnesses

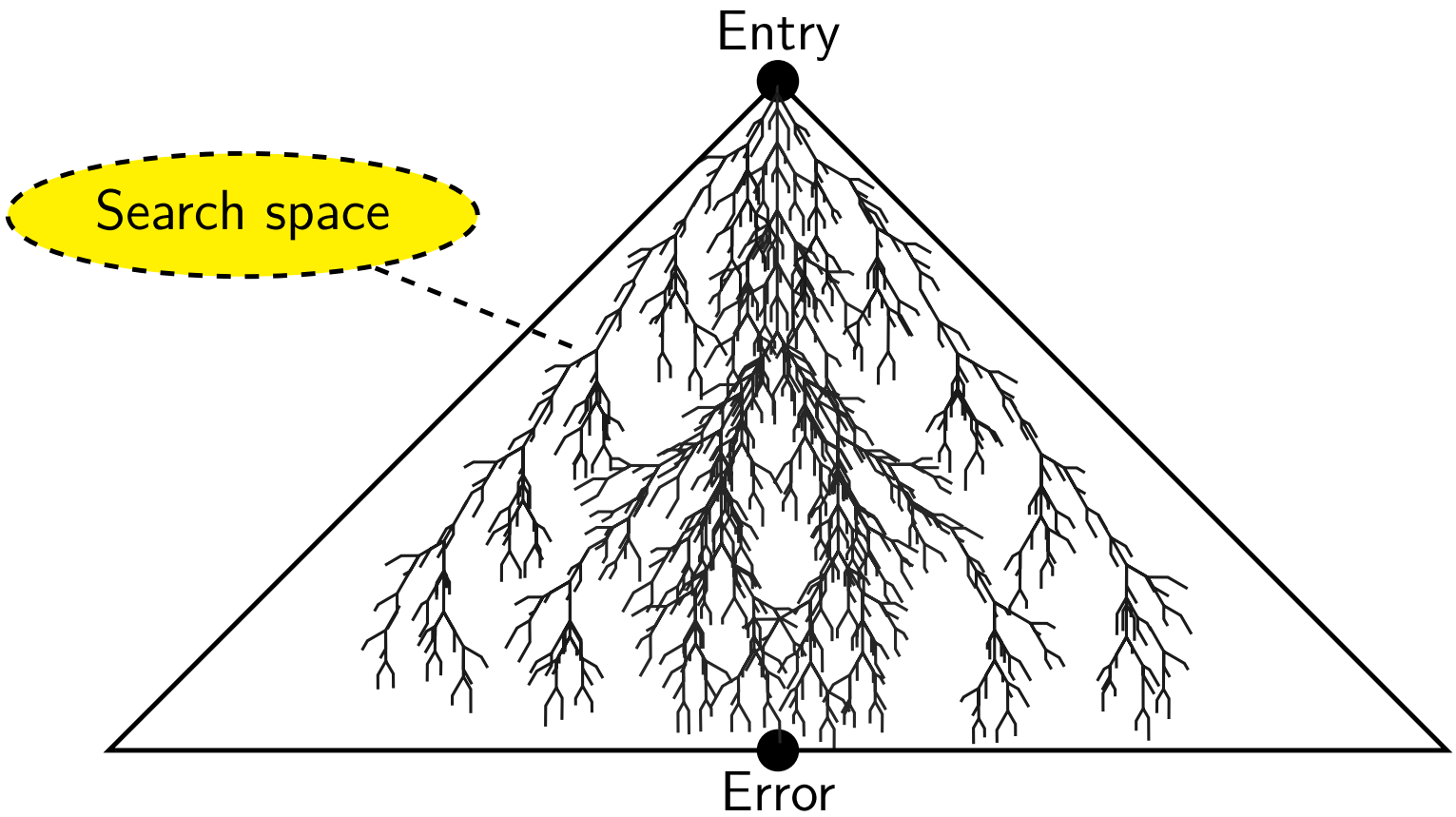
FSE'15



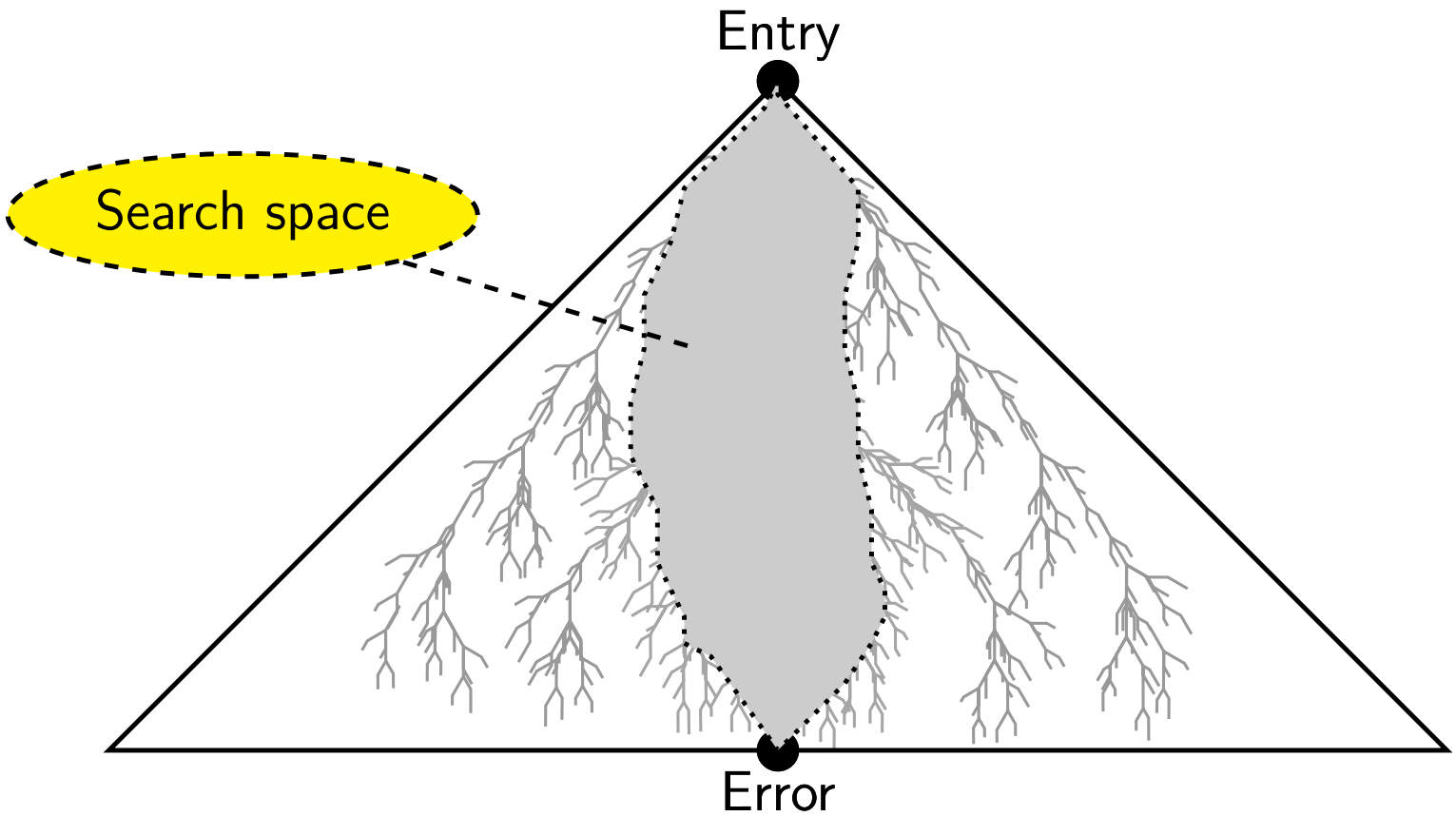
# Search-Space Reduction for Stepwise Testification



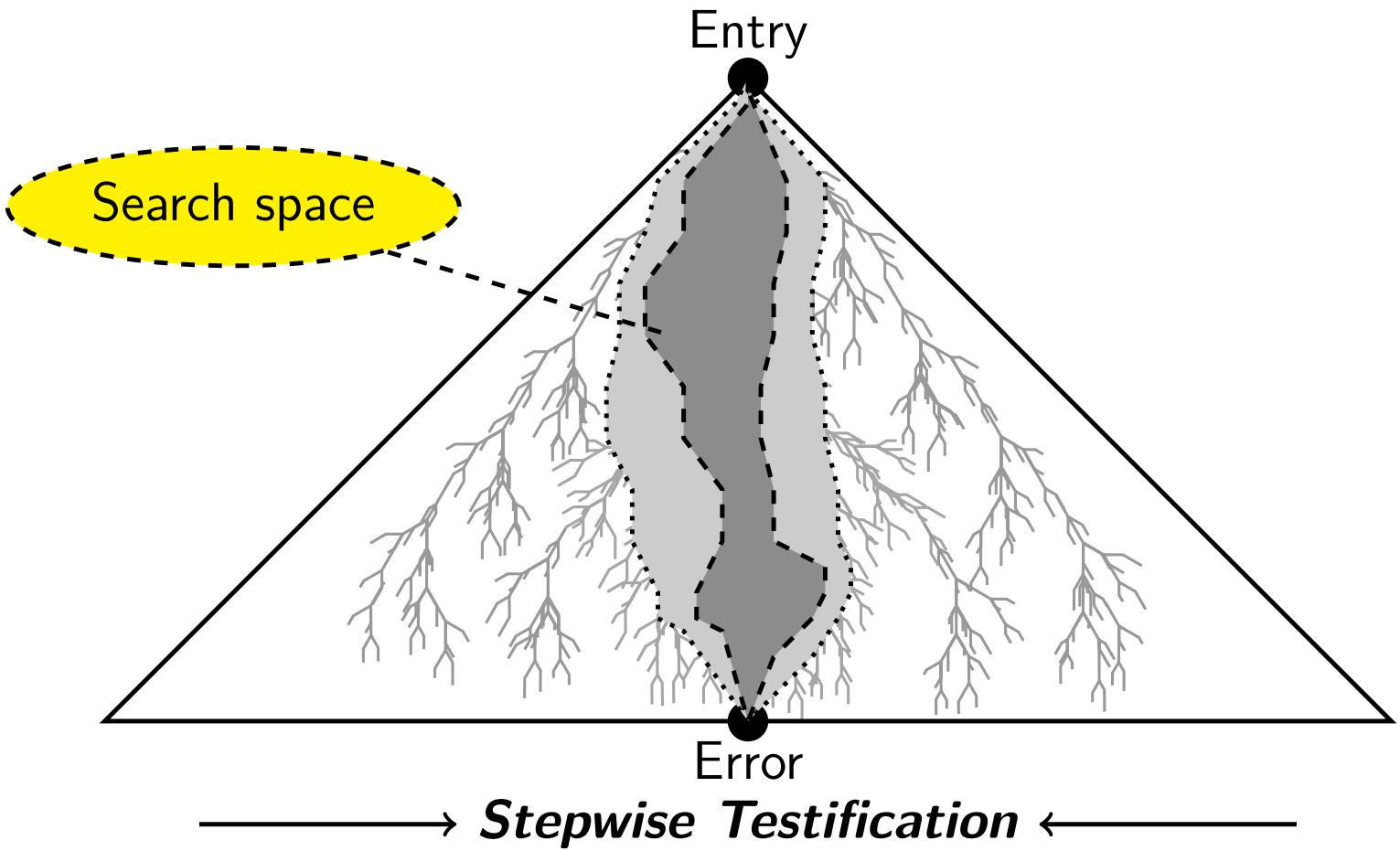
# Search-Space Reduction for Stepwise Testification



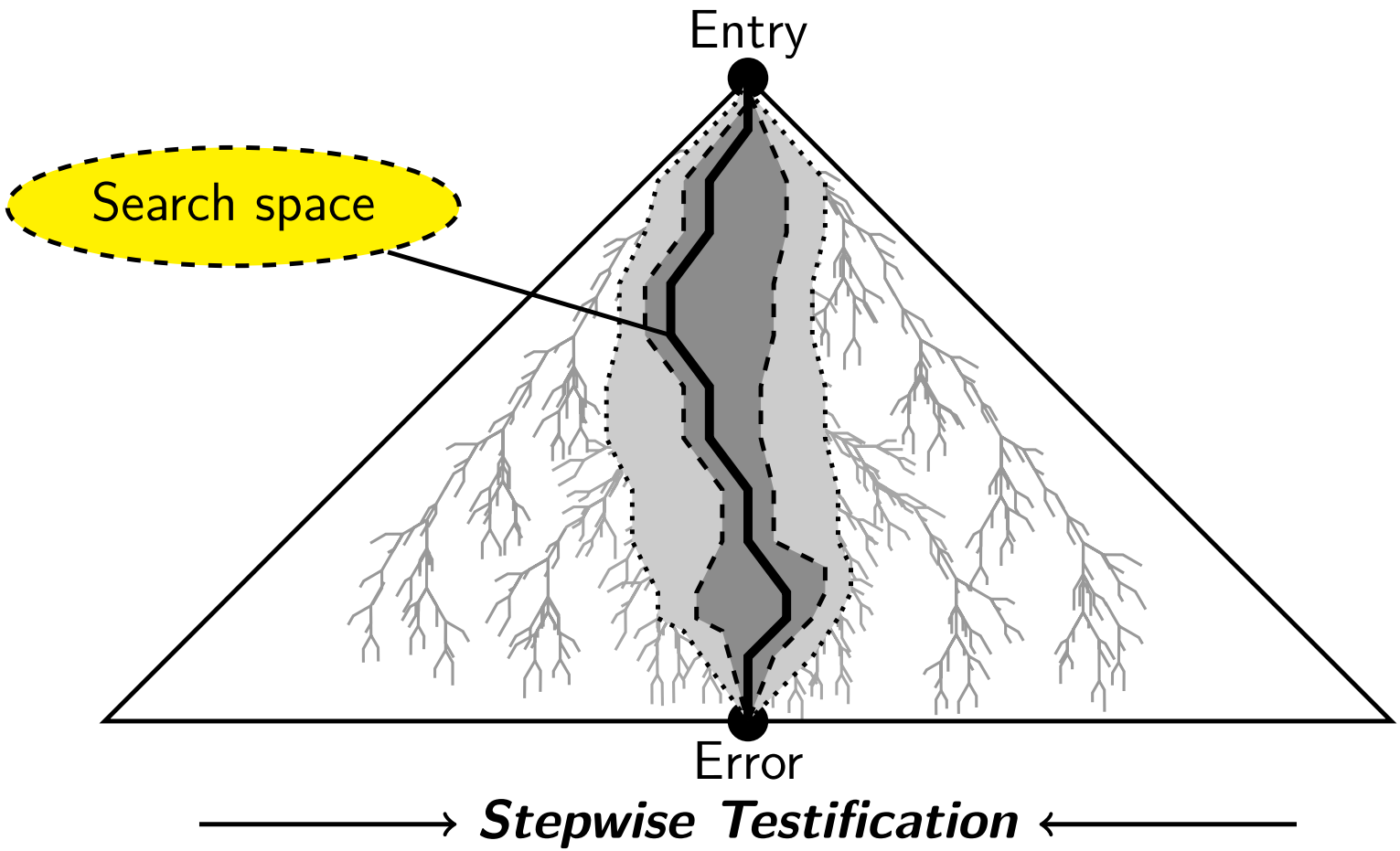
# Search-Space Reduction for Stepwise Testification



# Search-Space Reduction for Stepwise Testification



# Search-Space Reduction for Stepwise Testification



# 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

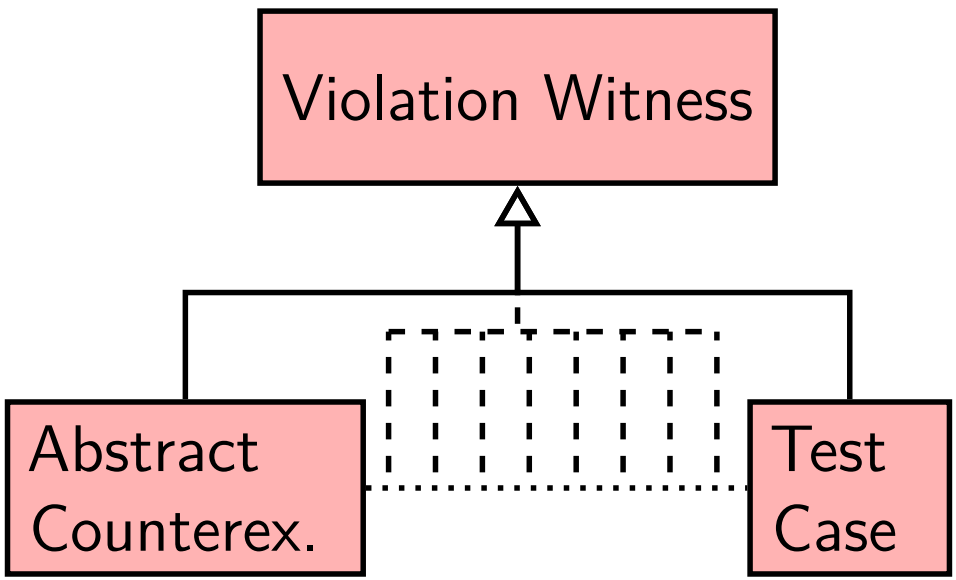
1. **Standardized way** to document verification results to enhance engineering processes **required**
2. **Difficult to establish trust** in results from an untrusted verifier

# Open Problems

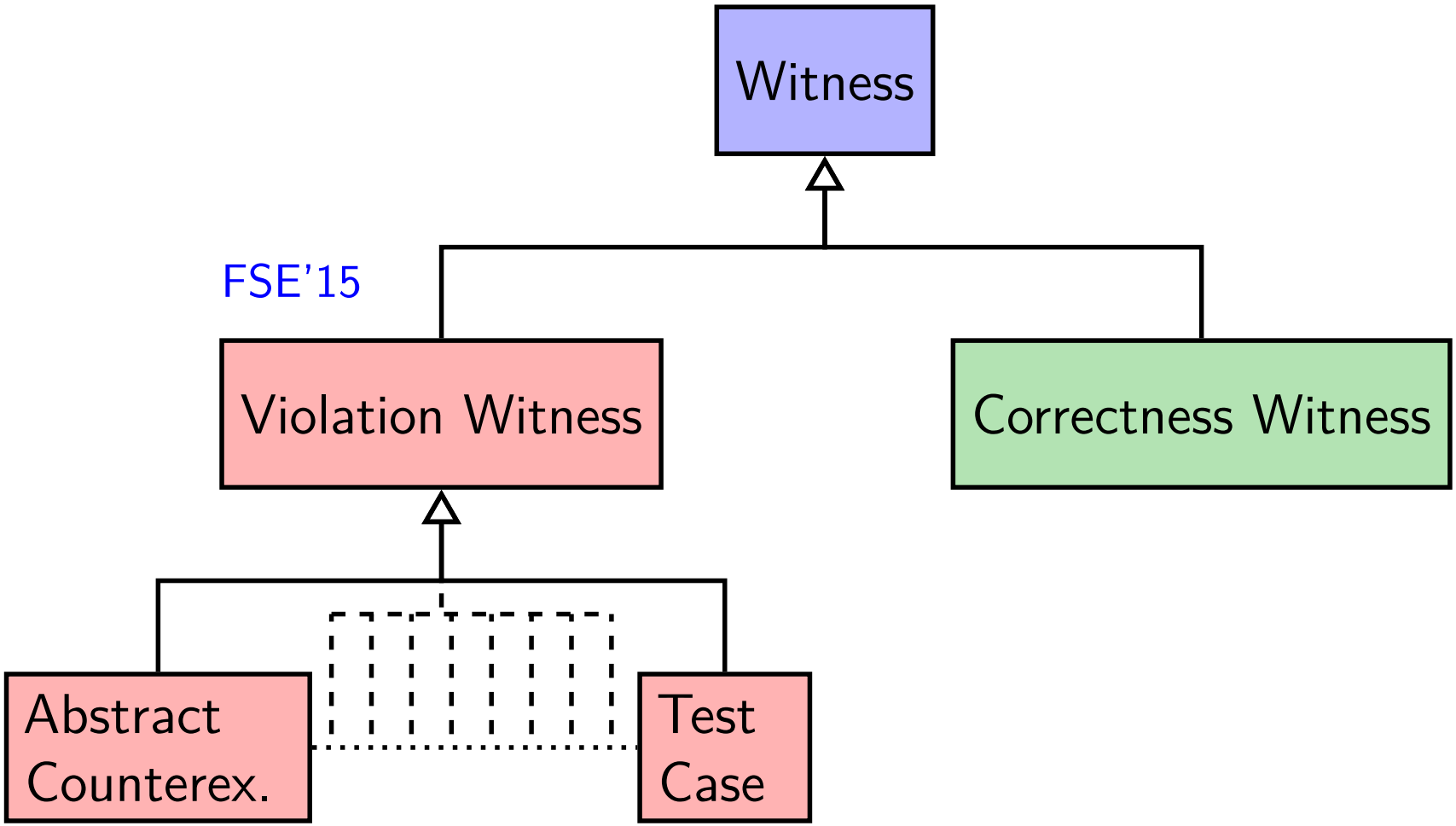
1. **Standardized way** to document verification results to enhance engineering processes **required**
2. **Difficult to establish trust** in results from an untrusted verifier
3. Potential for synergies between tools and techniques is **left unused**

# Verification Witnesses: Classification

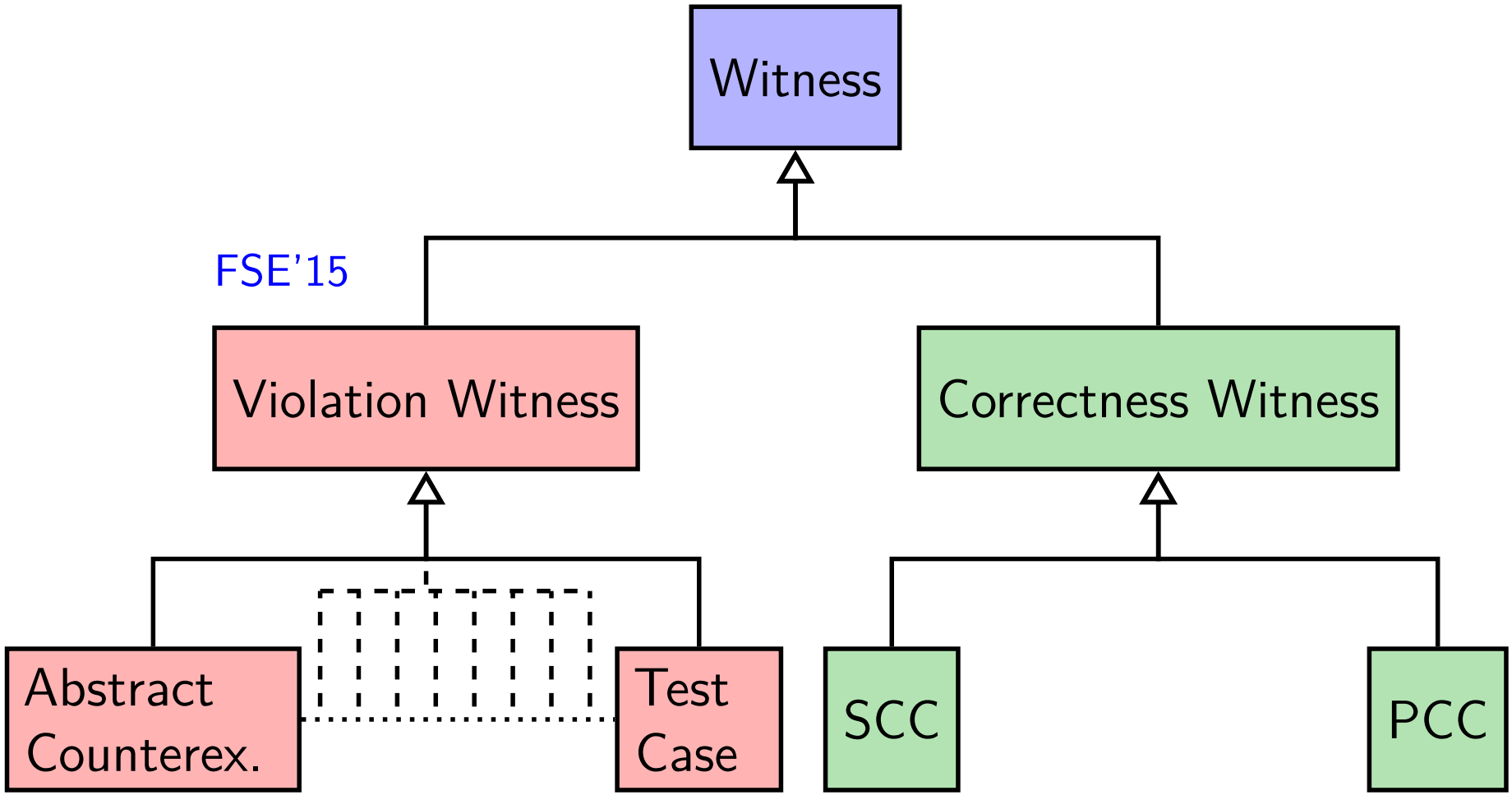
FSE'15



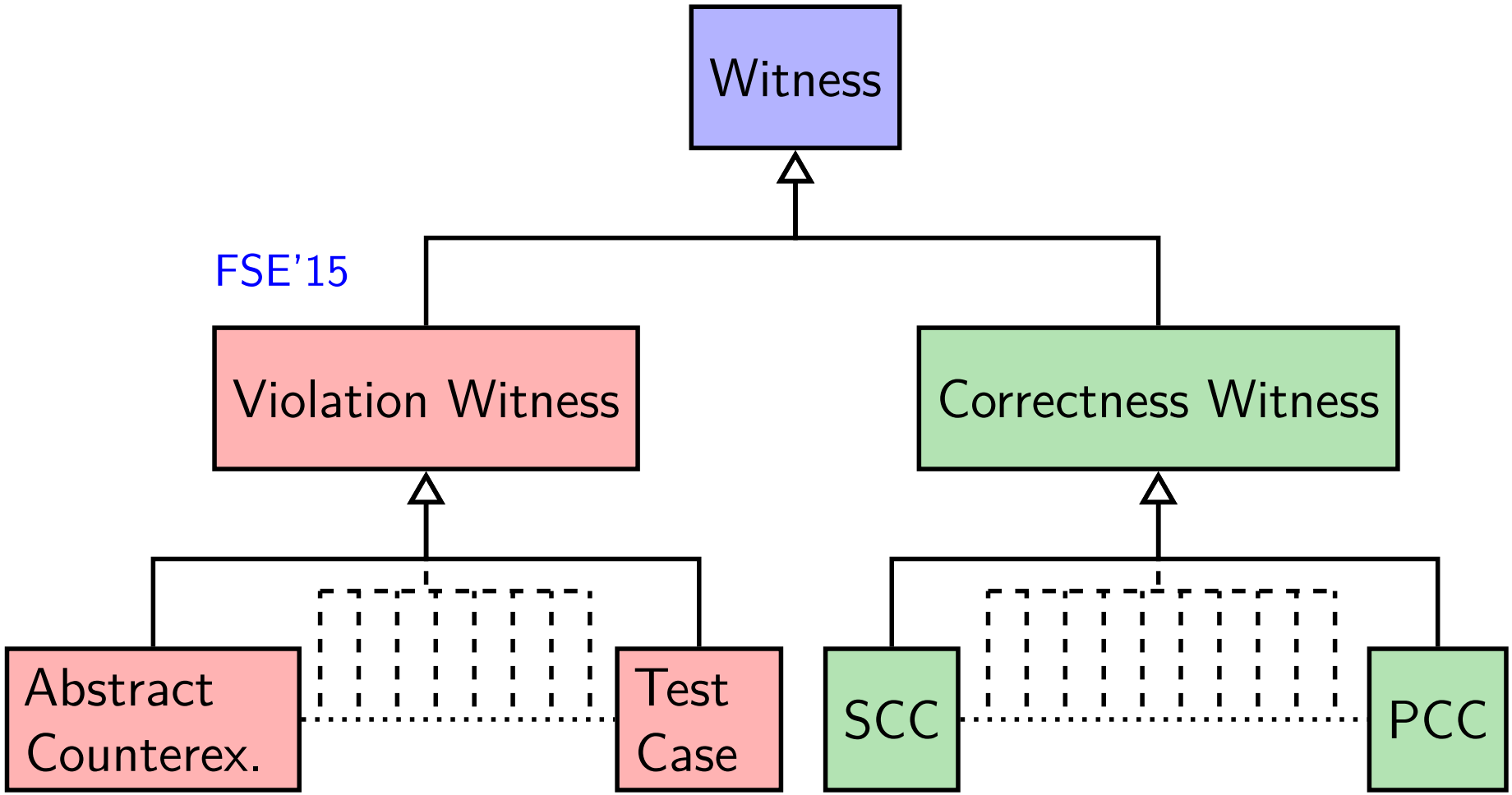
# Verification Witnesses: Classification



# Verification Witnesses: Classification



# Verification Witnesses: Classification

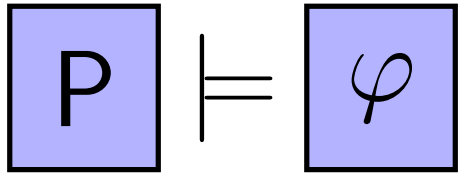



Taleghani & Atlee, ASE'10 Necula, POPL'97

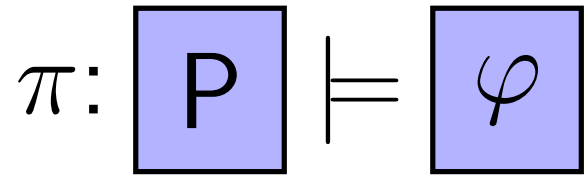
# Correctness Witnesses and Proof Certificates

- **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**

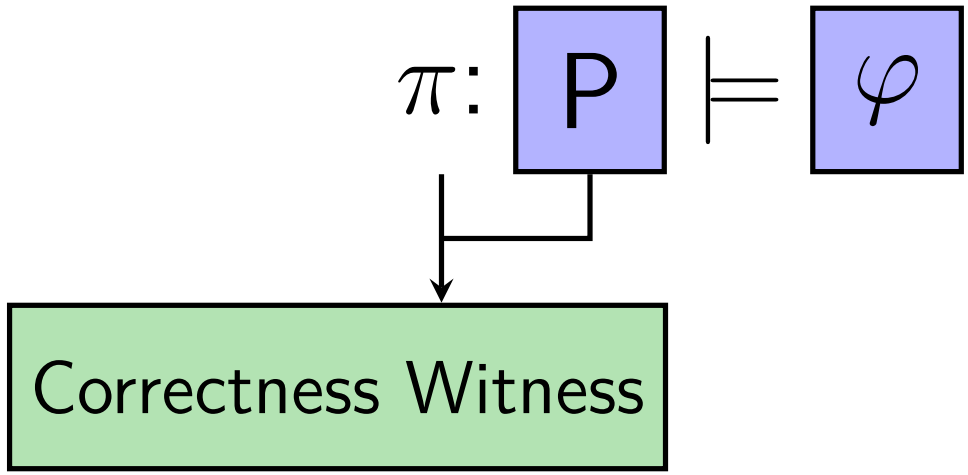
# Correctness Witnesses



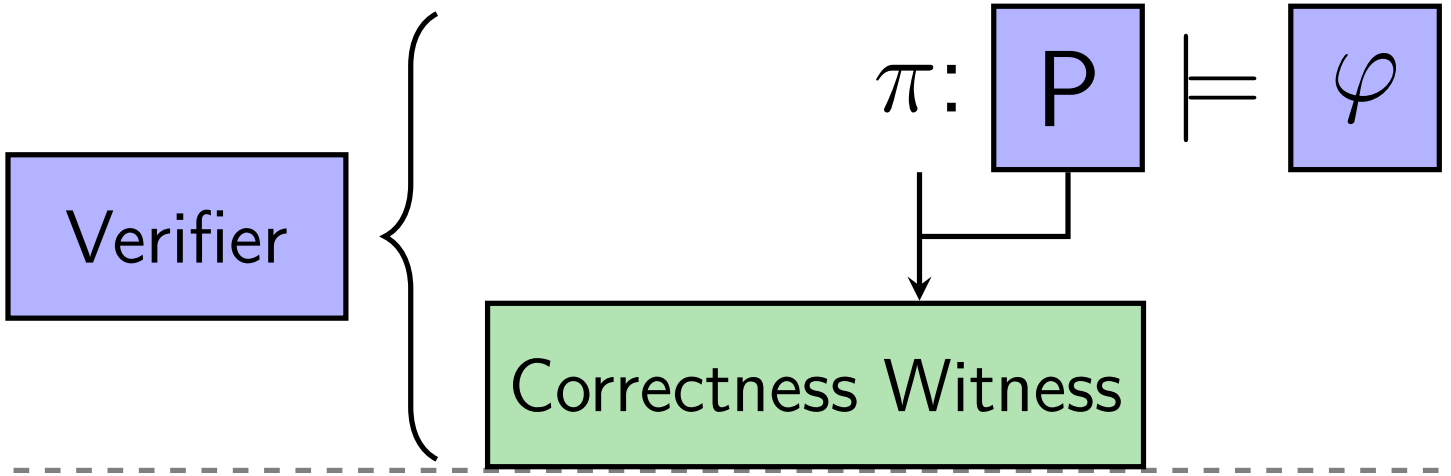
# Correctness Witnesses



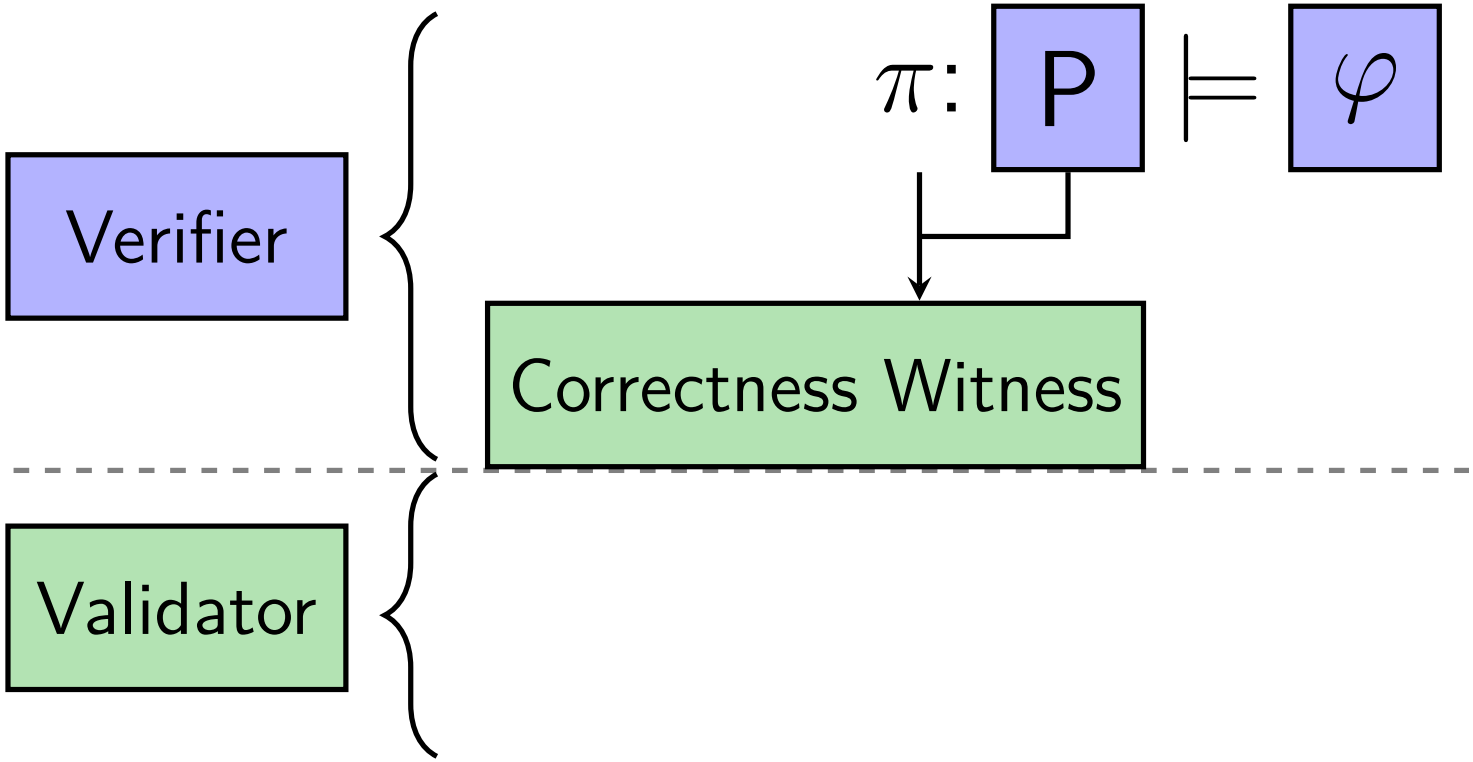
# Correctness Witnesses



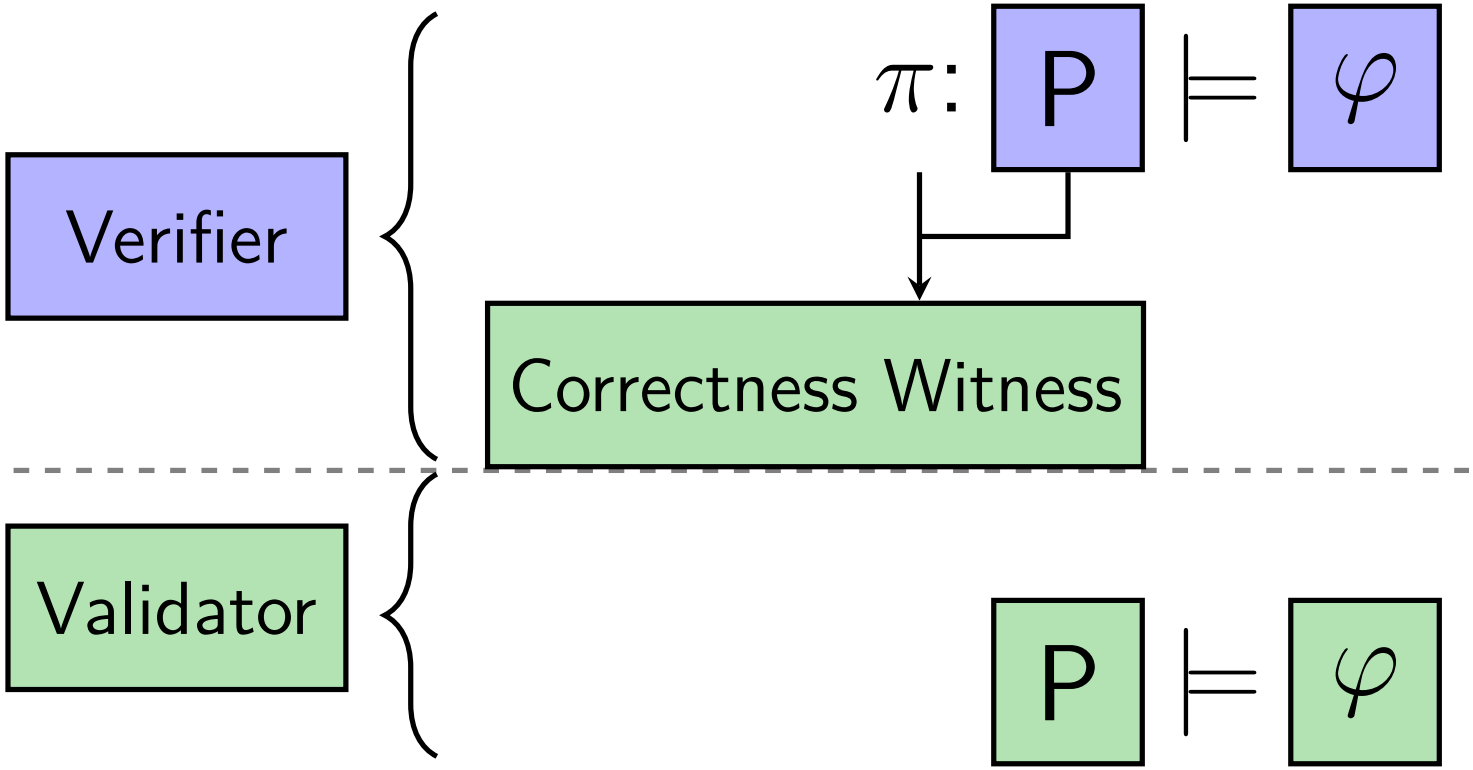
# Correctness Witnesses



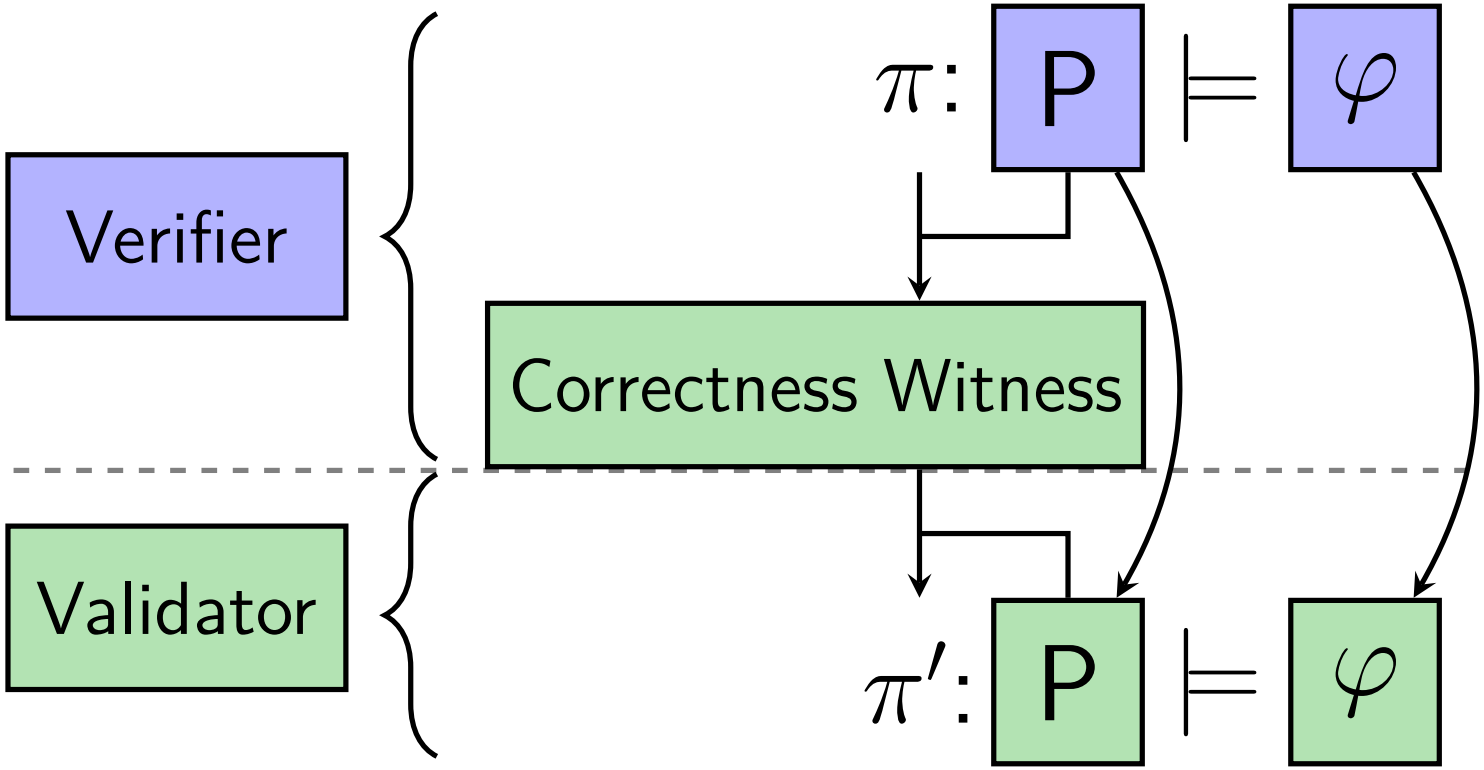
# Correctness Witnesses



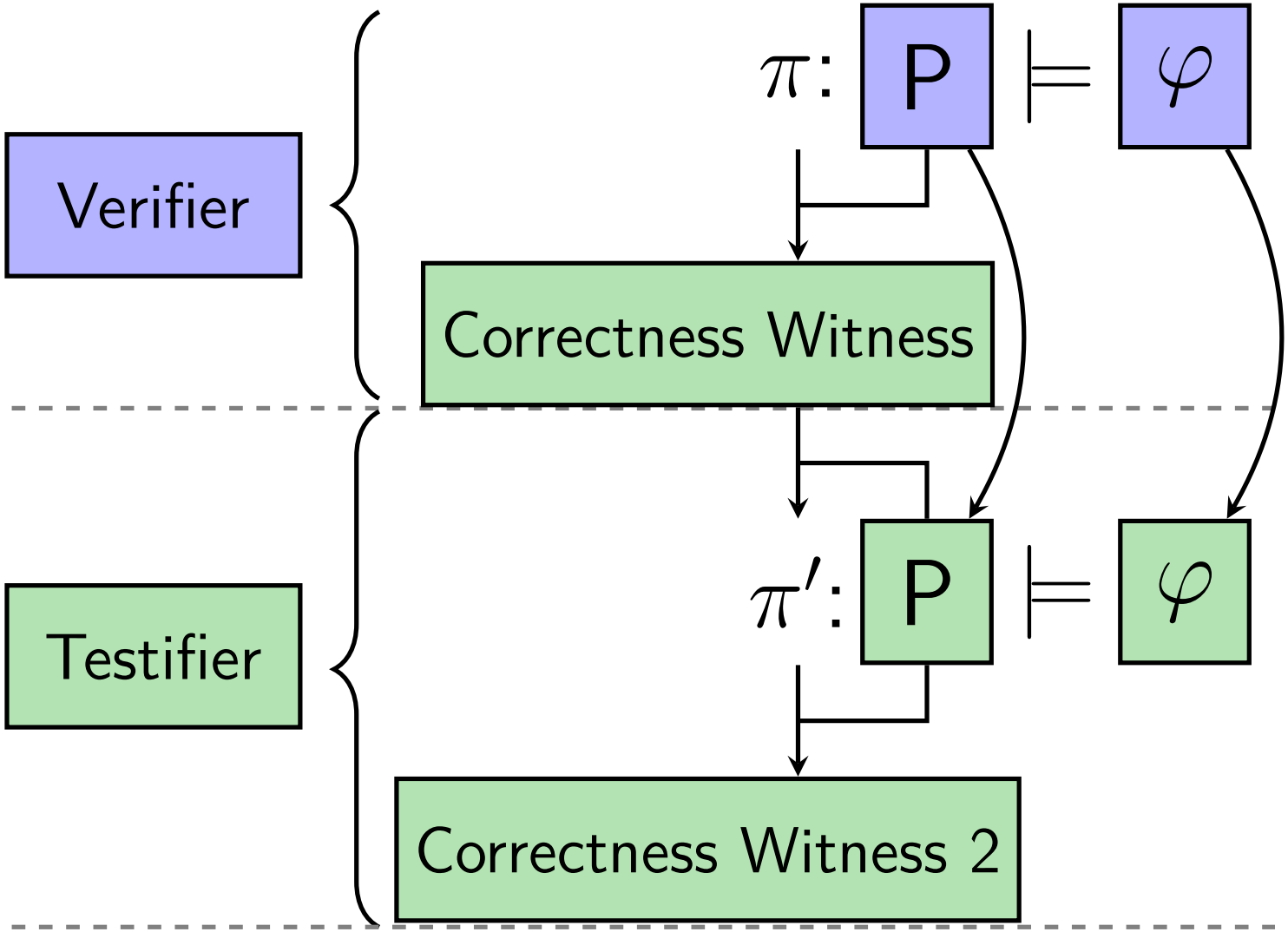
# Correctness Witnesses



# Correctness Witnesses



# Correctness Witnesses



# 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**

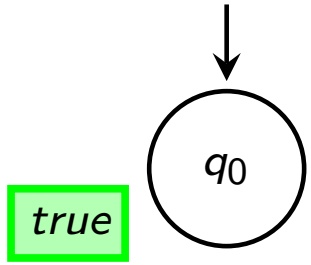
# Witness Automata

- 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

```
1  int main() {
2    unsigned int x = nondet();
3    unsigned int y = x;
4    while (x < 1024) {
5      x = x + 1;
6      y = y + 1;
7    }
8    // Safety property
9    assert(x == y);
10   return 0;
11 }
```

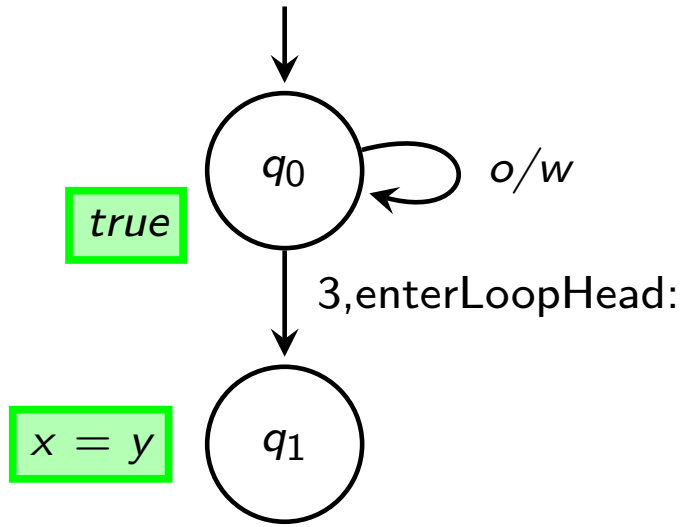
# Example: Inject Invariants




```
1  int main() {
2    unsigned int x = nondet();
3    unsigned int y = x;
4    while (x < 1024) {
5      x = x + 1;
6      y = y + 1;
7    }
8    // Safety property
9    assert(x == y);
10   return 0;
11 }
```

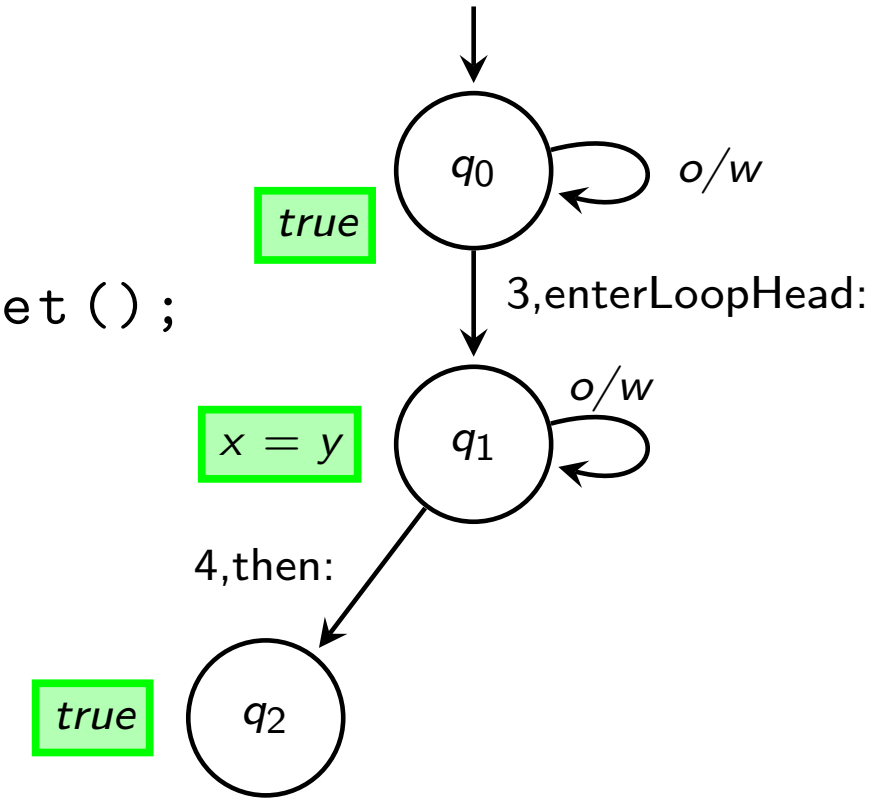
# Example: Inject Invariants

```
1  int main() {
2    unsigned int x = nondet();
3    unsigned int y = x;
4    while (x < 1024) {
5      x = x + 1;
6      y = y + 1;
7    }
8    // Safety property
9    assert(x == y);
10   return 0;
11 }
```



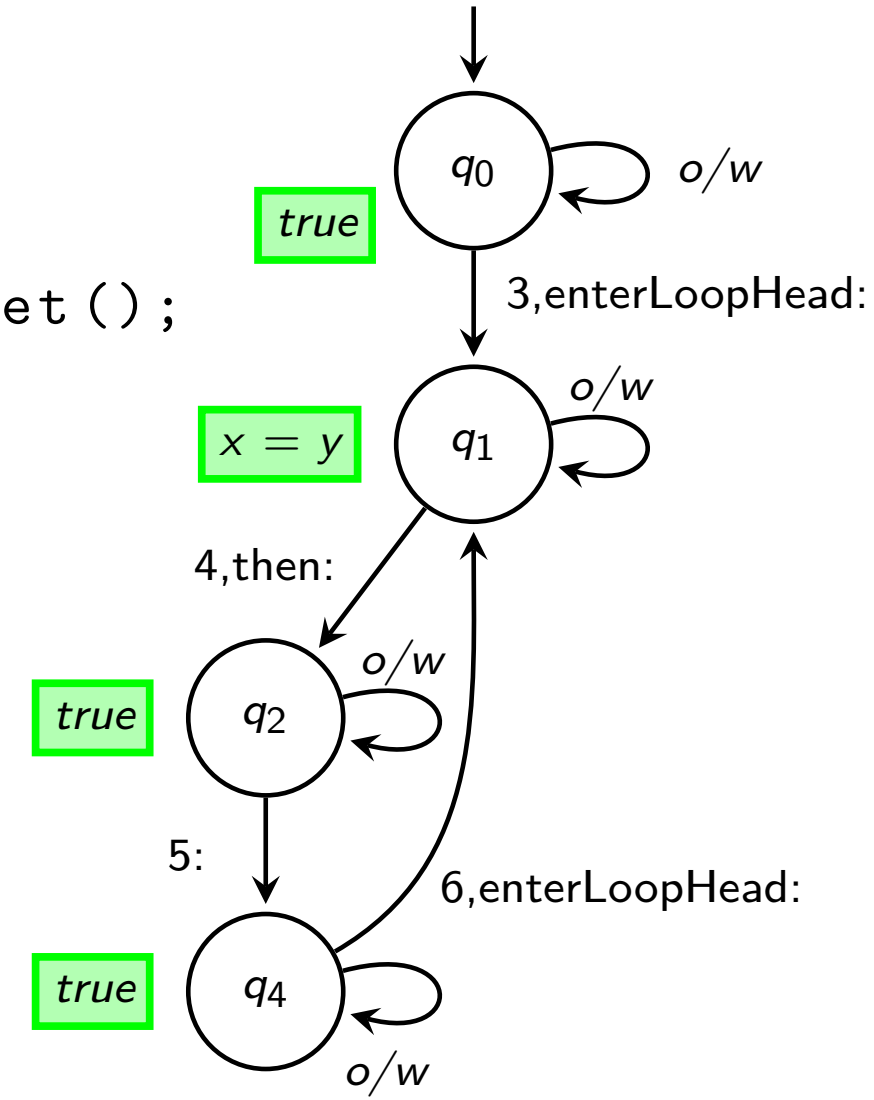
# Example: Inject Invariants

```
1  int main() {
2    unsigned int x = nondet();
3    unsigned int y = x;
4    while (x < 1024) {
5      x = x + 1;
6      y = y + 1;
7    }
8    // Safety property
9    assert(x == y);
10   return 0;
11 }
```



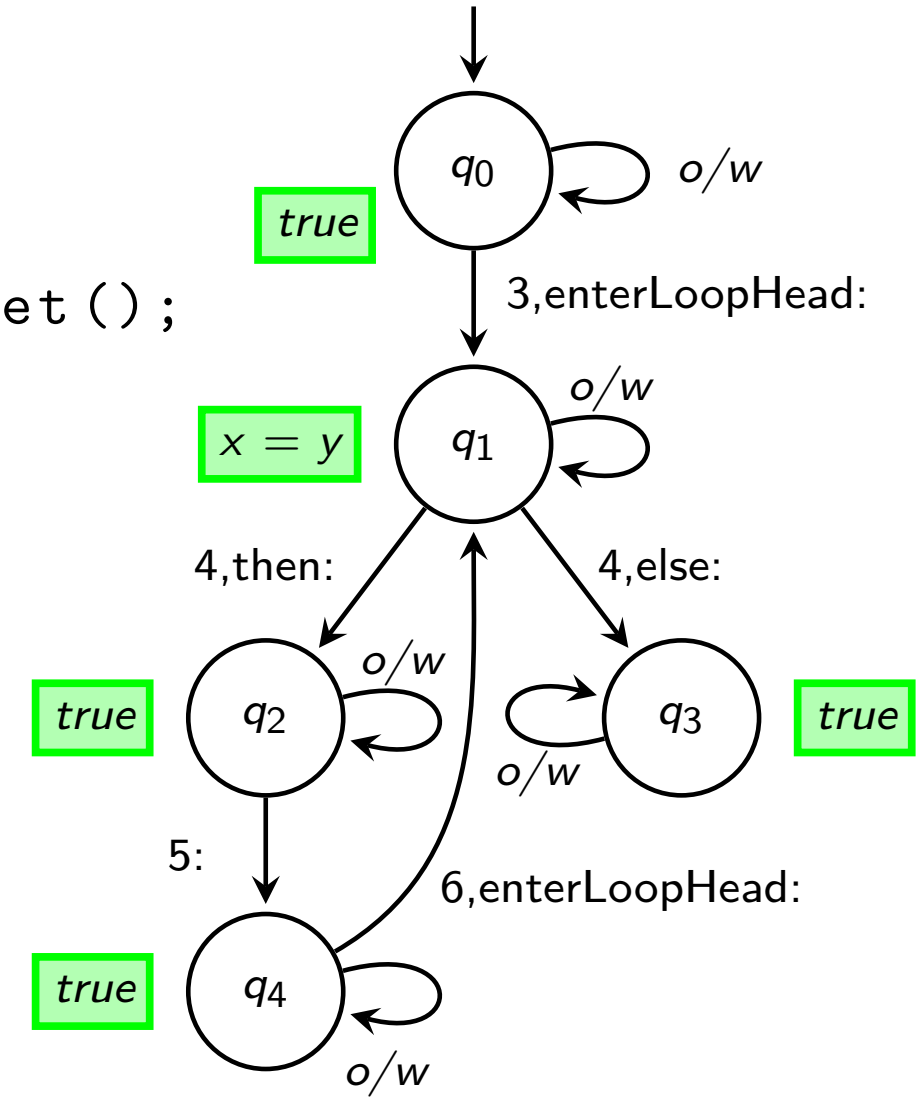
# Example: Inject Invariants

```
1  int main() {
2    unsigned int x = nondet();
3    unsigned int y = x;
4    while (x < 1024) {
5      x = x + 1;
6      y = y + 1;
7    }
8    // Safety property
9    assert(x == y);
10   return 0;
11 }
```



# Example: Inject Invariants

```
1  int main() {
2    unsigned int x = nondet();
3    unsigned int y = x;
4    while (x < 1024) {
5      x = x + 1;
6      y = y + 1;
7    }
8    // Safety property
9    assert(x == y);
10   return 0;
11 }
```



# Experiments

## Tasks and Limits

- Benchmark set: Competition on Software Verification 2016 (SV-COMP'16)
- CPU time: 15 min
- Memory: 15 GB

## Configurations

- CPAchecker with $k$-induction
- UltimateAutomizer with automata-based trace abstraction

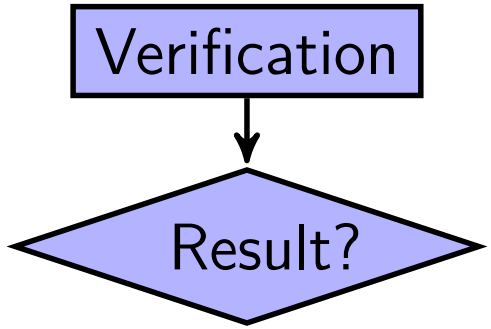
# Producing and Consuming Witnesses
# SV-COMP

Table 8: Confirmation rate of witnesses

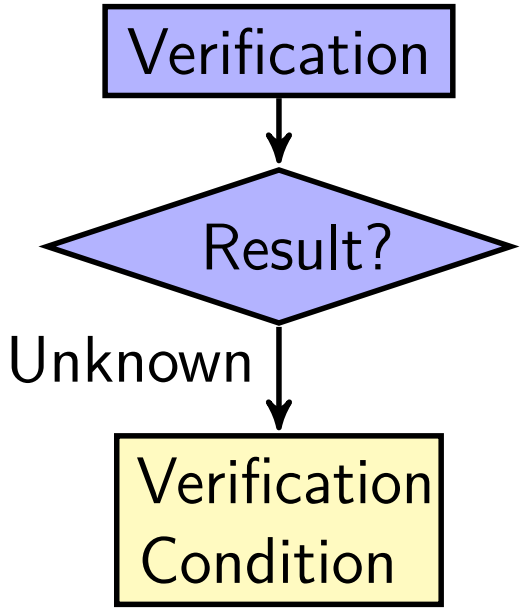
| Result | True | | | False | | |
|---|---|---|---|---|---|---|
| | Total | Confirmed | Unconfirmed | Total | Confirmed | Unconfirmed |
| UAutomizer | 3 558 | 3 481 | 77 | 1 173 | 1 121 | 52 |
| SMACK | 2 947 | 2 695 | 252 | 1 929 | 1 768 | 161 |
| CPA-Seq | 3 357 | 3 078 | 279 | 2 342 | 2 315 | 27 |

**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].

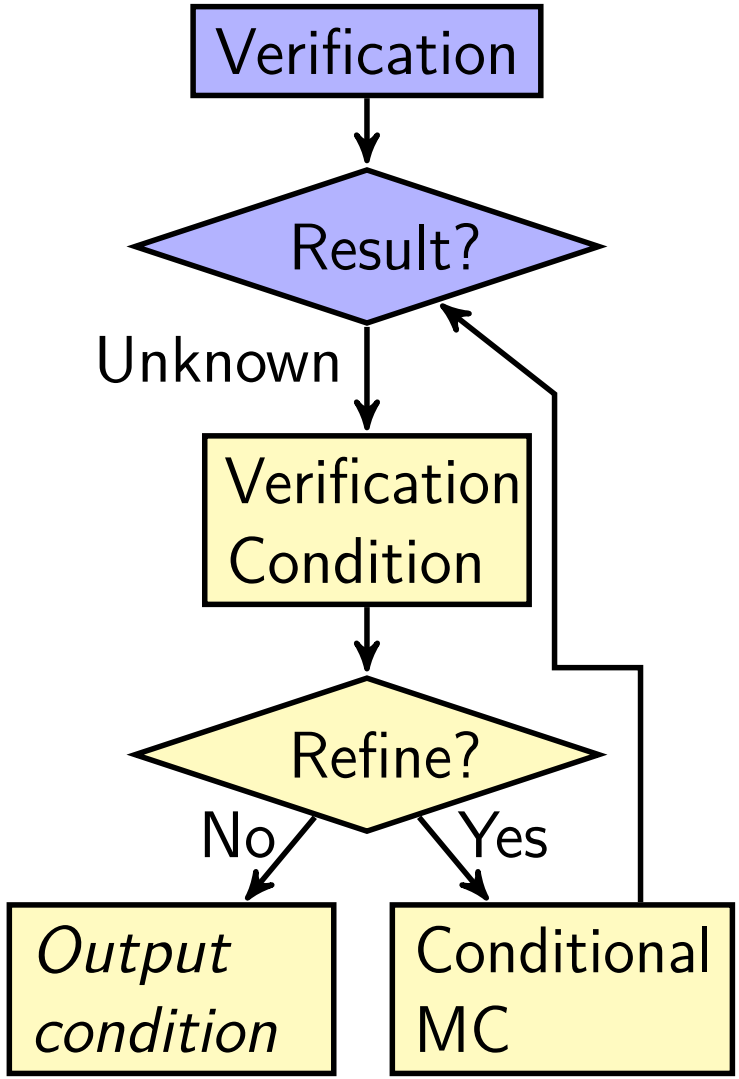
# Stepwise Testification: Classification



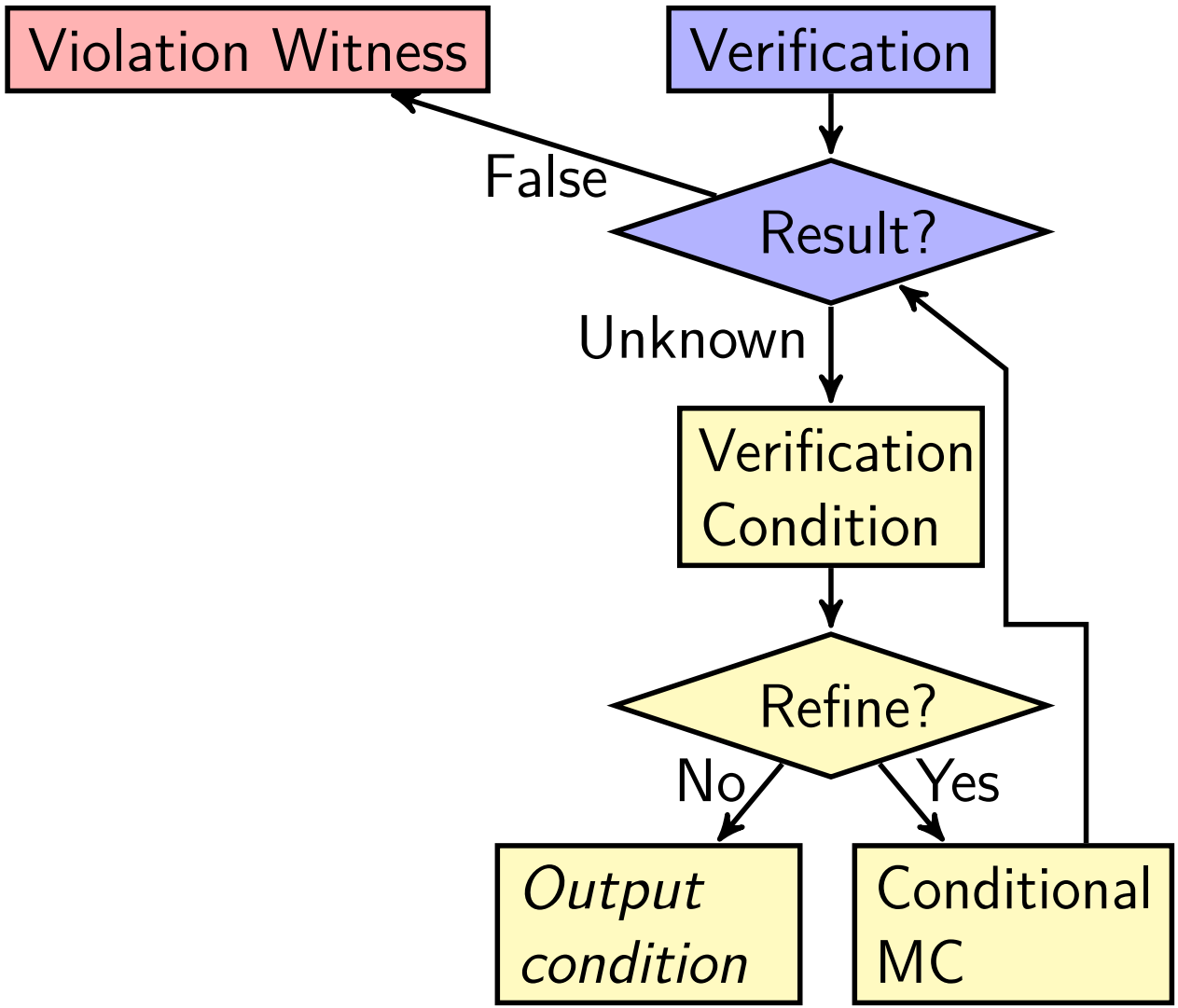
# Stepwise Testification: Classification



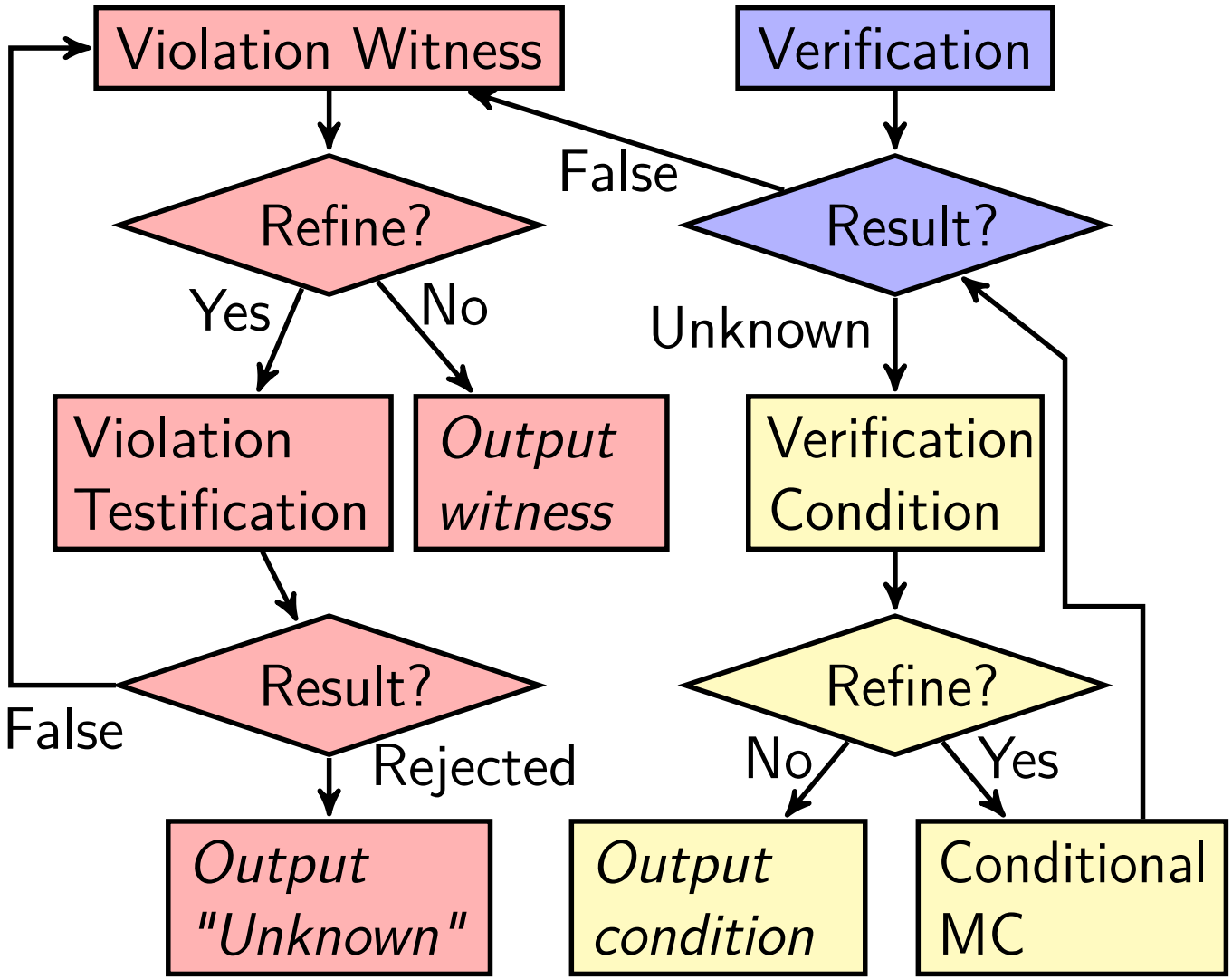
# Stepwise Testification: Classification



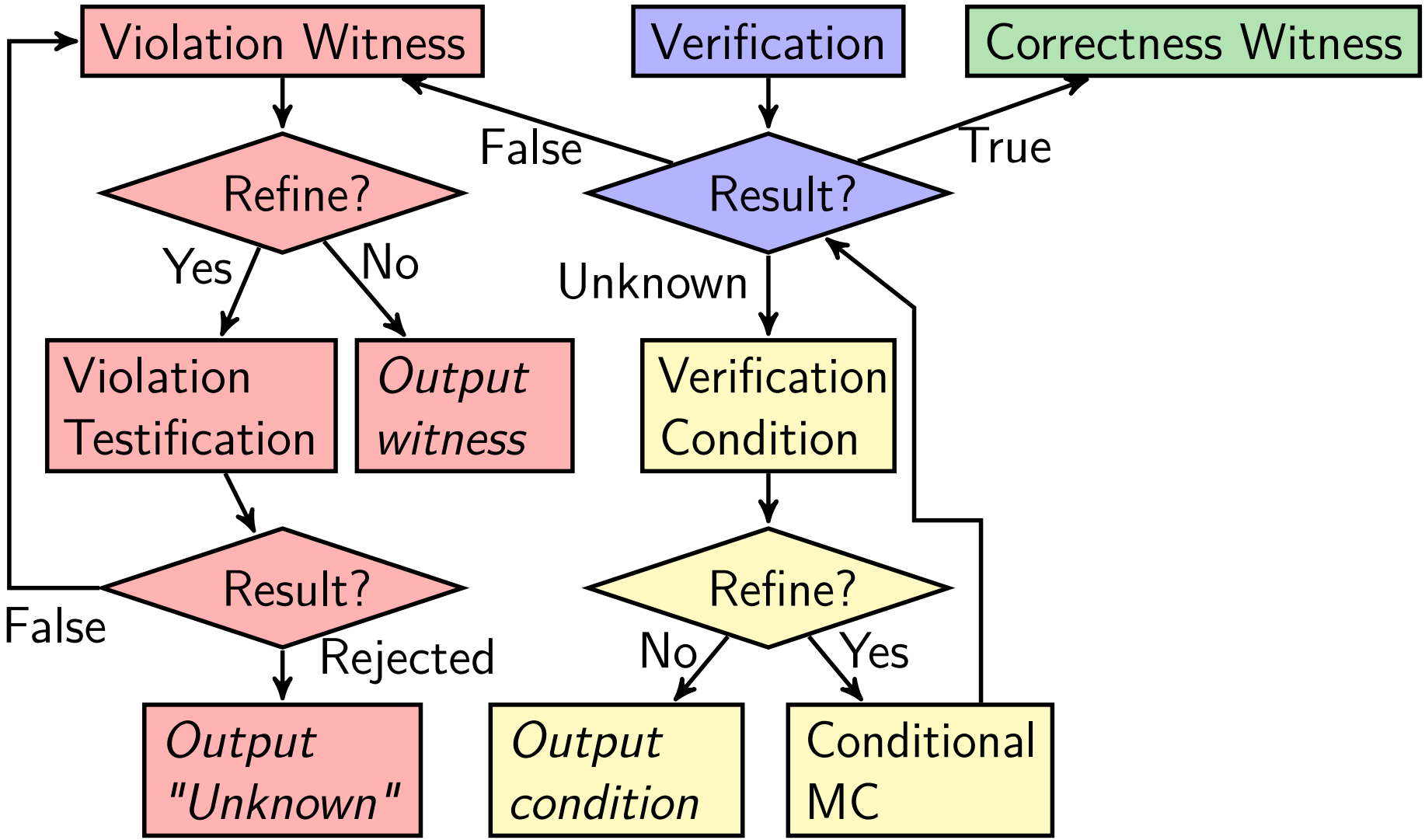
# Stepwise Testification: Classification



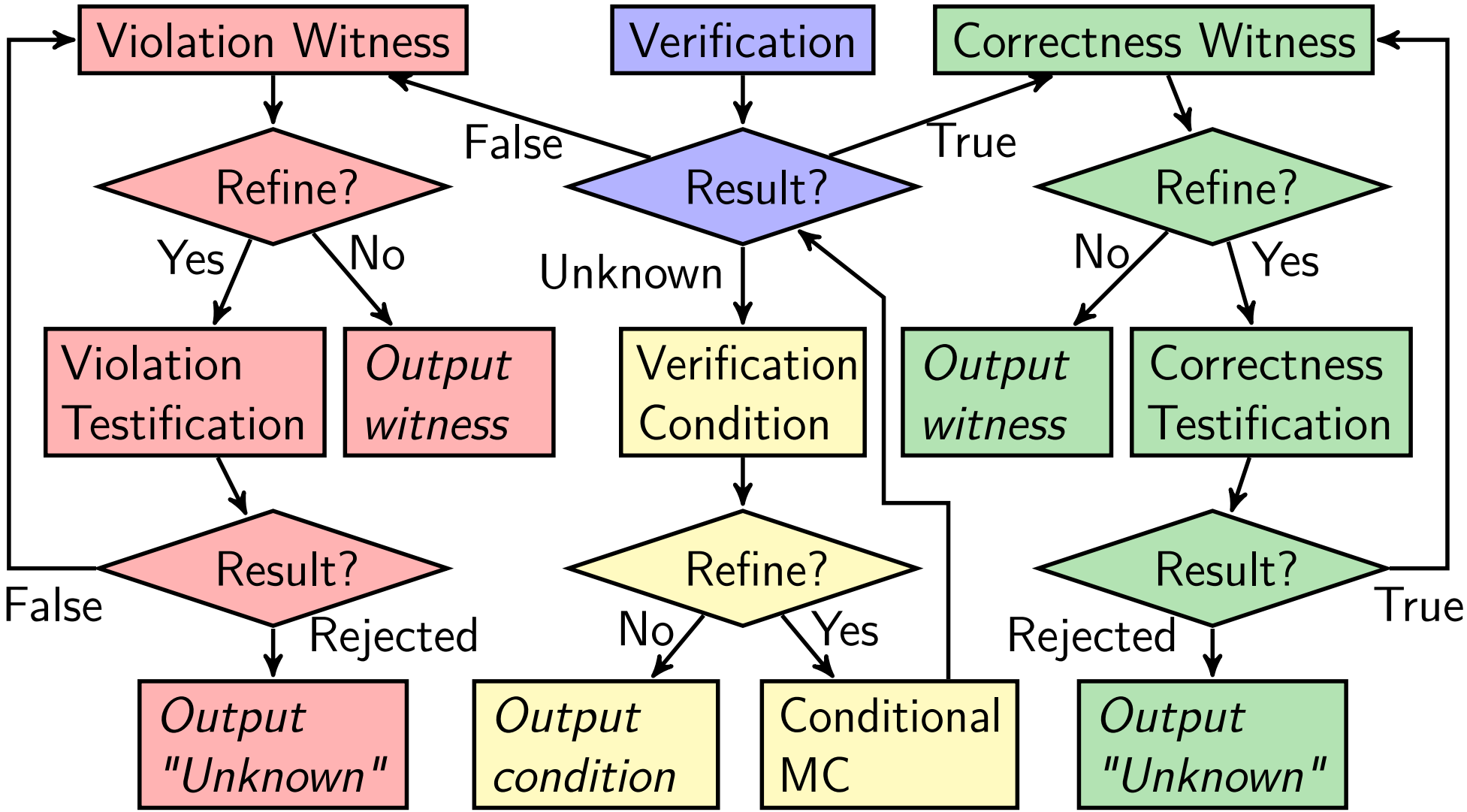
# Stepwise Testification: Classification



# Stepwise Testification: Classification



# Stepwise Testification: Classification



# 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**