

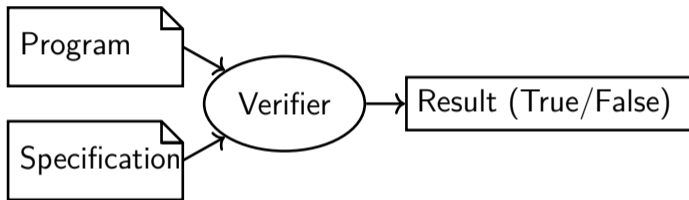
Software Verification Witnesses

Marian Lingsch-Rosenfeld

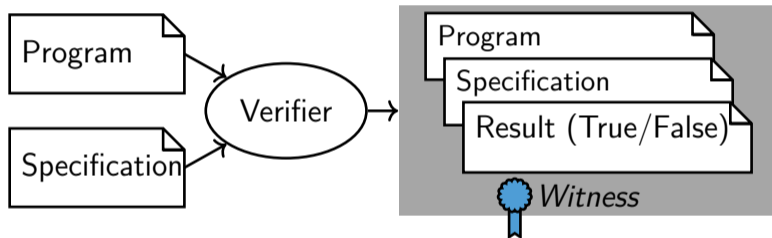
LMU Munich, Germany



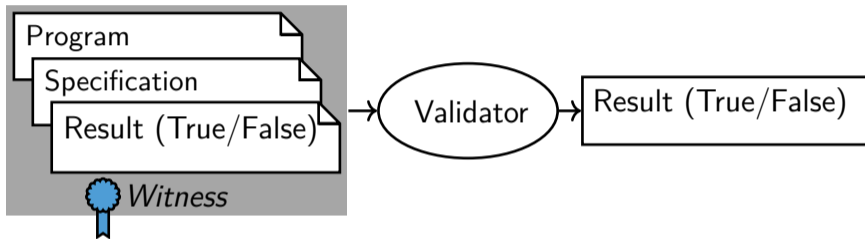
Software Verification



Software Verification with Witnesses



Witness Validation



- ▶ Validate untrusted results
- ▶ Easier than full verification

Purpose of Witnesses

- ▶ Provide insights into the verification process
- ▶ Validate verification results
- ▶ Exchange information between different tools

Software Verification Witnesses

Correctness Witnesses:

- ▶ Contains invariants
- ▶ Aids in reconstructing the proof

Violation Witnesses:

- ▶ Encode a set of paths
- ▶ At least one of the paths is a violation
- ▶ Aids in replaying a violation

Witnesses 2.0

- ▶ Contain a list of entries
 - ▶ **invariant_set**: part of a correctness witness
 - ▶ **violation_sequence**: part of a violation witness
- ▶ Metadata with information about the producer, program, and specification
- ▶ Content with the actual witness data

```
– entry_type: <...>
  metadata:
    format_version: "2.0"
    uuid: ""
    creation_time: ""
    producer:
      name: "CPAchecker"
      version: "2.3.1-svn"
      configuration: "svcomp24"
    task:
      input_files: <...>
      input_file_hashes: <...>
      specification: <...>
      data_model: "ILP32"
      language: "C"
  content: <...>
```

Correctness Witness 2.0

```
1  int main(void) {
2      short x = nondet();
3      int y = x;
4
5      while (x < 1024) {
6          x++;
7          y++;
8      }
9
10     assert(x == y);
11 }
```

```
1  <...>
2  content:
3  - invariant :
4      type: "loop_invariant "
5      location :
6          file_name: "example.c"
7          line : 5
8          column: 3
9          function : "main"
10     value: "( y == x )"
11     format: "c_expression"
```


Correctness Witnesses 2.0

- ▶ Contain a set of location and loop invariants
- ▶ **location invariant**: holds for every path through the given location
- ▶ **loop invariant**: holds for every path before the loop condition is evaluated

```
1 <...>
2 content:
3 - invariant :
4     type: "loop_invariant "
5     location :
6         file_name: "example.c"
7         line : 5
8         column: 3
9         function : "main"
10    value: "( y == x )"
11    format: "c_expression "
```

Correctness Witnesses & CPAchecker

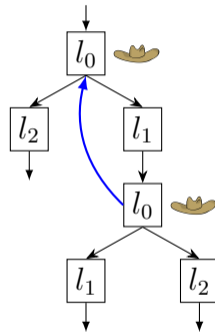
- ▶ Validation
 - ▶ Through K-Induction
 - ▶ Through Predicate Analysis for Overflows

Correctness Witnesses & CPAchecker

- ▶ Validation
 - ▶ Through K-Induction
 - ▶ Through Predicate Analysis for Overflows
- ▶ Exchange information
 - ▶ Add predicates to the predicate analysis
 - ▶ Add invariants to K-Induction

Correctness Witnesses & CPAchecker

- ▶ Validation
 - ▶ Through K-Induction
 - ▶ Through Predicate Analysis for Overflows
- ▶ Exchange information
 - ▶ Add predicates to the predicate analysis
 - ▶ Add invariants to K-Induction
- ▶ Export
 - ▶ Loop invariants are exported everywhere
 - ▶ Location invariants are exported at function calls



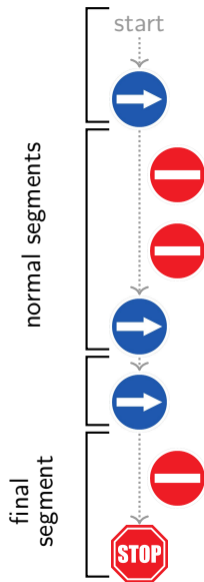
Violation Witness 2.0

```
1 int main(void) {
2     short x = nondet();
3     int y = x + 1;
4
5     while (x < 1024) {
6         x++;
7         y++;
8     }
9
10    assert(x == y);
11 }
```

```
1 <...>
2 content:
3 - segment:
4   - waypoint:
5     type: assumption
6     location :
7       line : 3
8       file_name: 'example.c'
9     constraint :
10      value: "x == 1024"
11 - segment:
12   - waypoint:
13     type: branching
14     location :
15       line : 5
16       file_name: 'example.c'
17     constraint :
18       value: false
19 - segment:
20   - waypoint:
21     type: target
22     location :
23       line : 10
24       file_name: 'example.c'
```

Violation Witnesses 2.0

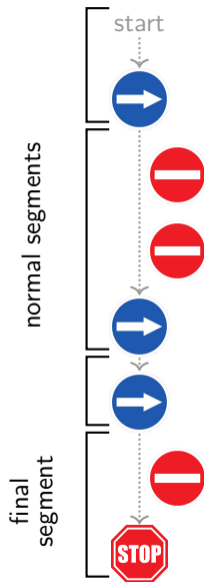
- ▶ Contain a set of waypoints
- ▶ **follow**: the waypoint has to be passed as soon as the location is entered
- ▶ **avoid**: the run represented by the witness must not pass the waypoint (“sink node”)
- ▶ **target**: the property violation



Violation Witnesses 2.0

Follow Waypoints

- ▶ **assumption**: the constraint must be valid before executing the statement
- ▶ **branching**: follow the given branch at the location
- ▶ **function_enter**: the function must be entered
- ▶ **function_return**: the value returned by the function



Violation Witnesses & CPAchecker

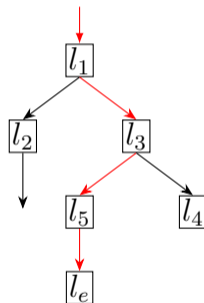
- ▶ Validation
 - ▶ Through Predicate Analysis
 - ▶ The witness restricts the state-space

Violation Witnesses & CPAchecker

- ▶ Validation
 - ▶ Through Predicate Analysis
 - ▶ The witness restricts the state-space
- ▶ Exchange information
 - ▶ Testification through `CPA-WITNESS2TEST`
 - ▶ Show path to the violation

Violation Witnesses & CPAchecker

- ▶ Validation
 - ▶ Through Predicate Analysis
 - ▶ The witness restricts the state-space
- ▶ Exchange information
 - ▶ Testification through CPA-WITNESS2TEST
 - ▶ Show path to the violation
- ▶ Export
 - ▶ Export the counterexample directly
 - ▶ Matching CFA nodes to statements in code



Conclusion

Witnesses contain machine-readable data to:

- ▶ Provide insights into the verification process
- ▶ Increase confidence in the verification results
- ▶ Allow for information exchange

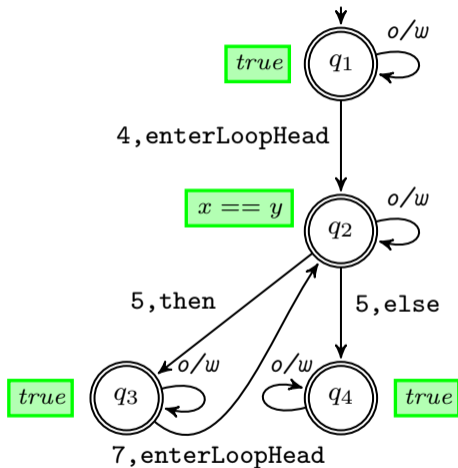


Witnesses version 2.0

Witnesses Version 1.0

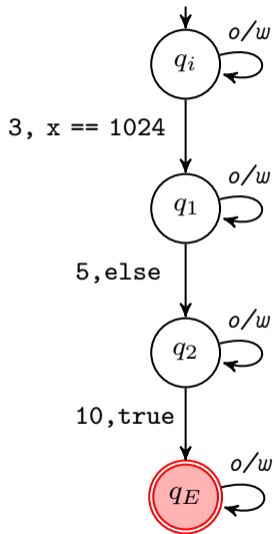
Correctness Witnesses 1.0

```
1 int main(void) {
2     short x = nondet();
3     int y = x;
4
5     while (x < 1024) {
6         x++;
7         y++;
8     }
9
10    assert(x == y);
11 }
```



Violation Witnesses 1.0

```
1  int main(void) {  
2      short x = nondet();  
3      int y = x + 1;  
4  
5      while (x < 1024) {  
6          x++;  
7          y++;  
8      }  
9  
10     assert(x == y);  
11 }
```



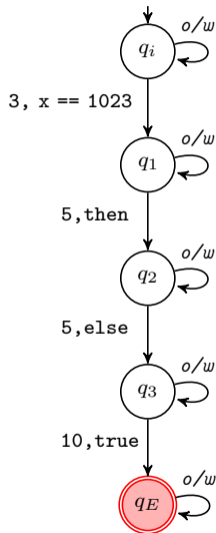
Why do we need Version 2.0?

Software Verification Witnesses 1.0: Unreadable Files

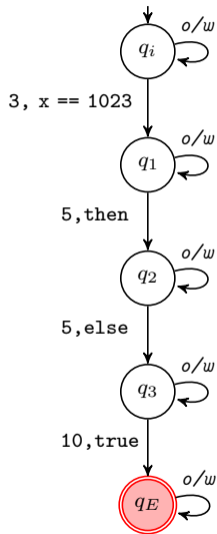
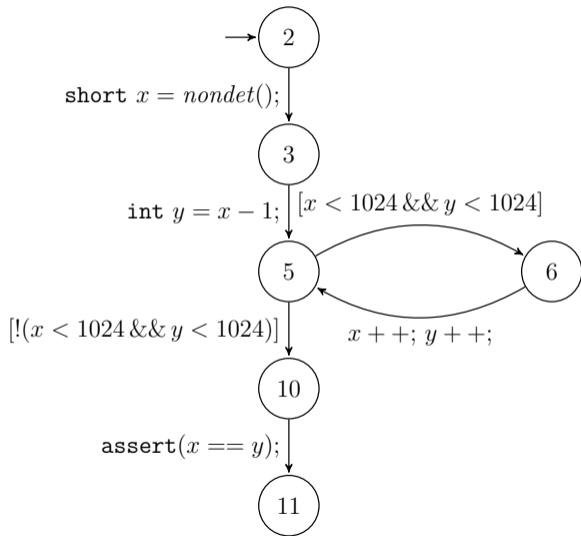
```
1 <node id="A0">
2 <data key="entry">>true</data>
3 </node>
4 <node id="A2_3_1"/>
5 <edge source="A0" target="A2_3_1">
6 <data key="startline">4</data>
7 <data key="endline">4</data>
8 <data key="enterFunction">main</data>
9 </edge>
10 <node id="A2"/>
11 <edge source="A2_3_1" target="A2">
12 <data key="enterLoopHead">>true</data>
13 <data key="startline">6</data>
14 <data key="endline">6</data>
15 <data key="endoffset">130</data>
16 </edge>
```


Software Verification Witnesses 1.0: Unclear semantics

```
1  int main(void) {  
2      short x = nondet();  
3      int y = x - 1;  
4  
5      while (x < 1024 && y < 1024) {  
6          x++;  
7          y++;  
8      }  
9  
10     assert(x == y);  
11 }
```



Software Verification Witnesses 1.0: Unclear semantics



Software Verification Witnesses 1.0: Unclear semantics

