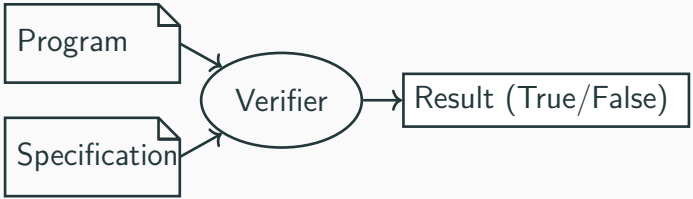
Software Verification Witnesses 2.0

Marian Lingsch-Rosenfeld

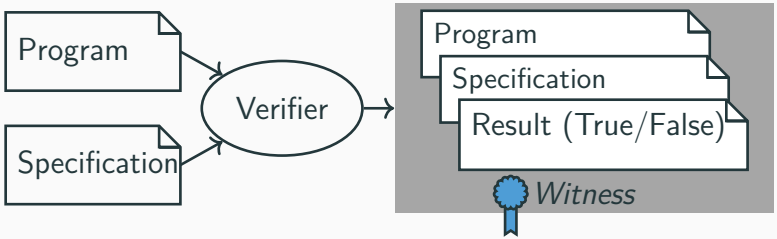
Paulína Ayaziová, Dirk Beyer, Martin Spiessl and
Jan Strejček

April 25, 2024
LMU Munich, SoSy-Lab

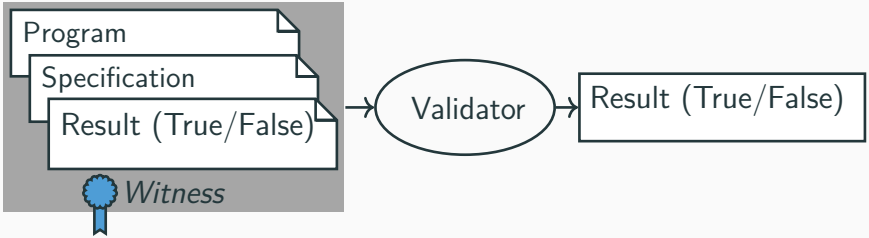
Software Verification



Software Verification with Witnesses



Witness Validation



- Validate untrusted results
- Usually easier than full verification

Witnesses for Cooperation

- CEGAR-PT [2]
- Component Based CEGAR [1]
- Cooperation Between Automatic and Interactive Software Verifiers [3]
- ...

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

Version 1.0 vs. Version 2.0

Version 1.0:

- ✗ Tied to the control-flow automaton
  - ⇒ Unclear semantics
  - ⇒ No underlying standard
  - ⇒ Harder to exchange information
- ✗ Difficult to read as file
- ✓ More complex
- ✓ Supports more properties
- ✗ Difficult to extend

Version 2.0:

- ✓ Tied to the program syntax
  - ⇒ Clear semantics
  - ⇒ Based on the C-Standard
  - ⇒ Easier to exchange information
- ✓ Human-readable
- ✗ Less complex
- ✗ Supports fewer properties
- ✓ Easy to extend

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

```
1  - entry_type: <...>
2    metadata:
3      format_version: "2.0"
4      uuid: ""
5      creation_time: ""
6      producer:
7        name: "CPAchecker"
8        version: "2.3.1-svn"
9        configuration: "svcomp24"
10     task:
11       input_files: <...>
12       input_file_hashes: <...>
13       specification: <...>
14       data_model: "ILP32"
15       language: "C"
16   content: <...>
```

Correctness Witnesses 2.0

```c
int main(void) {
  short x = nondet();
  int y = x;

  while (x < 1024) {
    x++;
    y++;
  }

  assert(x == y);
}
```

```yaml
<...>
  content:
  - invariant:
      type: "loop_invariant"
      location:
        file_name: "example.c"
        line: 5
        column: 3
        function: "main"
      value: "( y == x )"
      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"
```

Violation Witnesses 2.0

```c
int main(void) {
  short x = nondet();
  int y = x + 1;

  while (x < 1024) {
    x++;
    y++;
  }

  assert(x == y);
}
```

```yaml
<...>
  content:
  - segment:
    - waypoint:
        type: assumption
        location: {line: 3, file_name: 'a.c'}
        constraint:
          value: "x == 1024"
  - segment:
    - waypoint:
        type: branching
        location: {line: 5, file_name: 'a.c'}
        constraint:
          value: false
  - segment:
    - waypoint:
        type: target
        location: {line: 10, file_name: 'a.c'}
```

Violation Witnesses 2.0

- Contain a set of segments
- Each segment is composed of waypoints
- Segments contain multiple waypoints, but only need to pass a specific one

```
1  <...>
2    content:
3      - segment:
4        - waypoint:
5            type: assumption
6            location: {line: 3, file_name: 'a.c'}
7            constraint:
8              value: "x == 1024"
9      - segment:
10       - waypoint:
11           type: branching
12           location: {line: 5, file_name: 'a.c'}
13           constraint:
14             value: false
15     - segment:
16       - waypoint:
17           type: target
18           location: {line: 10, file_name: 'a.c'}
```

Violation Witnesses 2.0: Waypoints

- **follow**: the waypoint has to be passed as soon as the location is entered, at most one per segment
- **avoid**: the run represented by the witness must not pass the waypoint (“sink node”)
- **target**: the property violation, at most one per segment

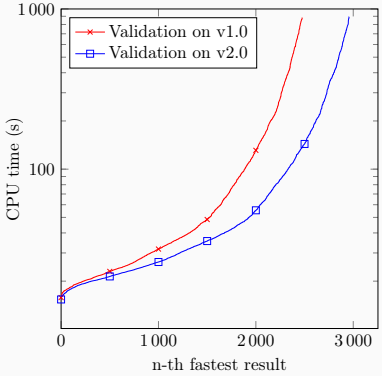
```
1  <...>
2    content:
3      - segment:
4        - waypoint:
5            type: assumption
6            location: {line: 3, file_name: 'a.c'}
7            constraint:
8              value: "x == 1024"
9      - segment:
10       - waypoint:
11           type: branching
12           location: {line: 5, file_name: 'a.c'}
13           constraint:
14             value: false
15     - segment:
16       - waypoint:
17           type: target
18           location: {line: 10, file_name: 'a.c'}
```

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

```
1  <...>
2    content:
3      - segment:
4        - waypoint:
5            type: assumption
6            location: {line: 3, file_name: 'a.c'}
7            constraint:
8              value: "x == 1024"
9      - segment:
10       - waypoint:
11           type: branching
12           location: {line: 5, file_name: 'a.c'}
13           constraint:
14             value: false
15     - segment:
16       - waypoint:
17           type: target
18           location: {line: 10, file_name: 'a.c'}
```

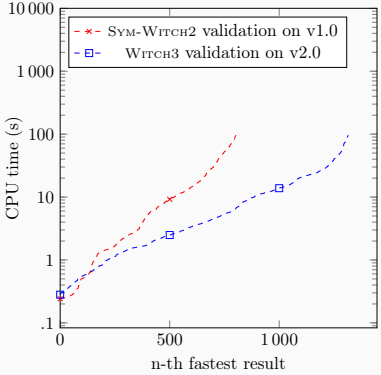
Version 1.0 vs. Version 2.0: Validation Correctness



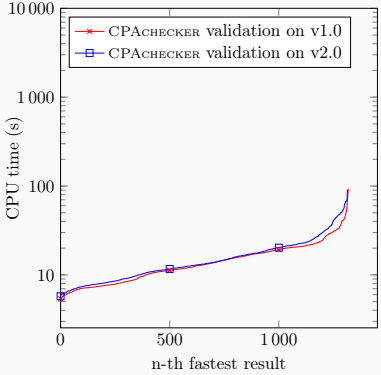
(a) Validation using UAutomizer

Correctness witnesses produced by CPAchecker: Quantile plots for the time taken for validation of the old and new witnesses for UAutomizer

Version 1.0 vs. Version 2.0: Validation Violation



(b) Validation of violation witnesses generated by CPAchecker



(c) Validation of violation witnesses generated by Symbiotic

Violation witnesses: Quantile plots for the time taken for validation of the old and the new witnesses generated by two different verifiers for two different validators

File Sizes Correctness Witnesses

	Witnesses v1.0			Witnesses v2.0		
Attribute	Min	Median	Max	Min	Median	Max
Length in Lines of Code	53	1536	1014533	18	28	1058
Size in kB	3	52	35573	1	1	965
Number of Invariants	0	1	162	0	1	104

Different attributes of correctness witnesses in version 1.0 and 2.0 generated by CPAchecker

File Sizes Violation Witnesses

	Witnesses v1.0			Witnesses v2.0		
Attribute	Min	Median	Max	Min	Median	Max
Length in Lines of Code	12	372	258730	27	171	114460
Size in kB	2	14	9098	1	6	3071
Number of Nodes	1	38	28304	-	-	-
Number of Waypoints	-	-	-	1	13	9537

Different attributes of violation witnesses in version 1.0 and 2.0 generated by CPAchecker and Symbiotic

Adoption of Witness Version 2.0 in SV-COMP 2024

	Correctness	Violation
Witness Generation	8	2
Witness Validation	4	2

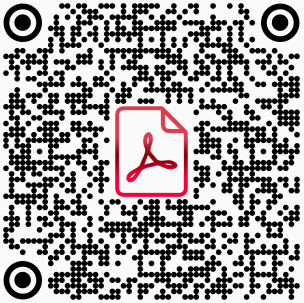
- Great interest considering the short development time!

Future Work

- Concurrent programs
- Data races
- Function contracts
- Termination
- Memory Safety

Conclusion

- Witnesses version 2.0 are an improvement over version 1.0
  - ✓ Clear semantics
  - ✓ Smaller
  - ✓ Human readable
  - ✓ No loss in performance
- Great interest from the SV-COMP community to use version 2.0!



References i

[1] 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

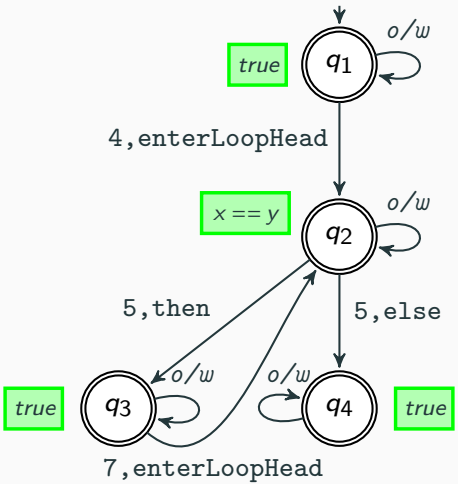
[2] Beyer, D., Lingsch-Rosenfeld, M., Spiessl, M.: CEGAR-PT: A tool for abstraction by program transformation. In: Proc. ASE. pp. 2078–2081. IEEE (2023). https://doi.org/10.1109/ASE56229.2023.00215

[3] 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

Witnesses Version 1.0

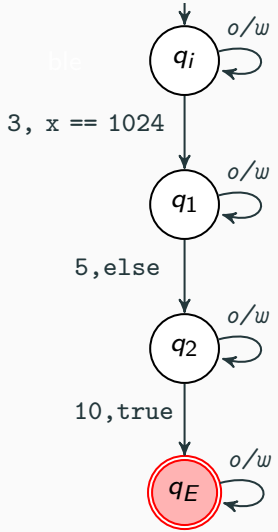
Correctness Witnesses 1.0

```
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 }
```



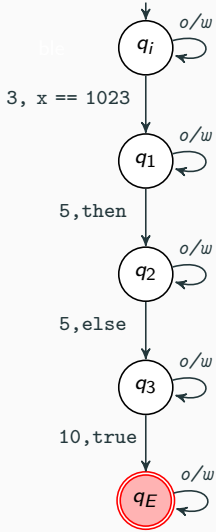
Why do we need Version 2.0?

Software Verification Witnesses 1.0: Unreadable Files

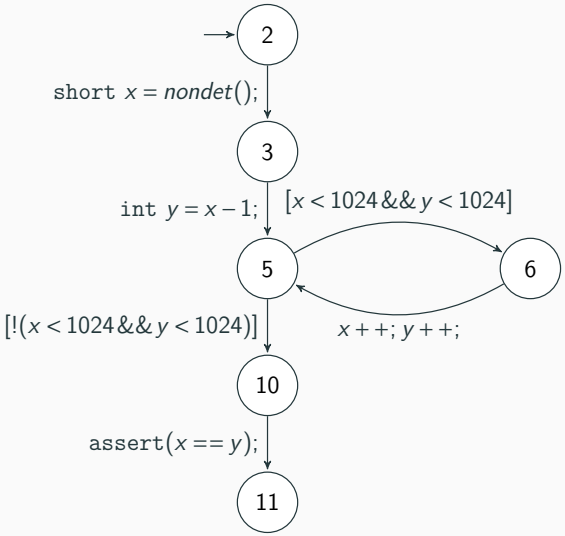
```
1  <node id="A0">
2  <data key="entry">true</data>
3  </node>
4  <node id="A2_3_1"/>
5  <edge source="A0" target="A2_3_1">
6  <data key="startline">4</data>
7  <data key="endline">4</data>
8  <data key="enterFunction">main</data>
9  </edge>
10 <node id="A2"/>
11 <edge source="A2_3_1" target="A2">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

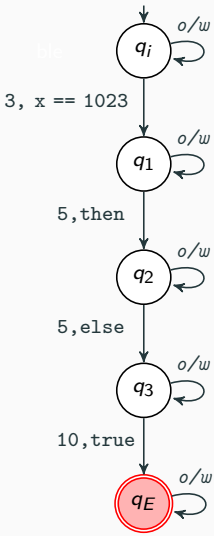
Software Verification Witnesses 1.0: Unclear semantics

```
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 }
```

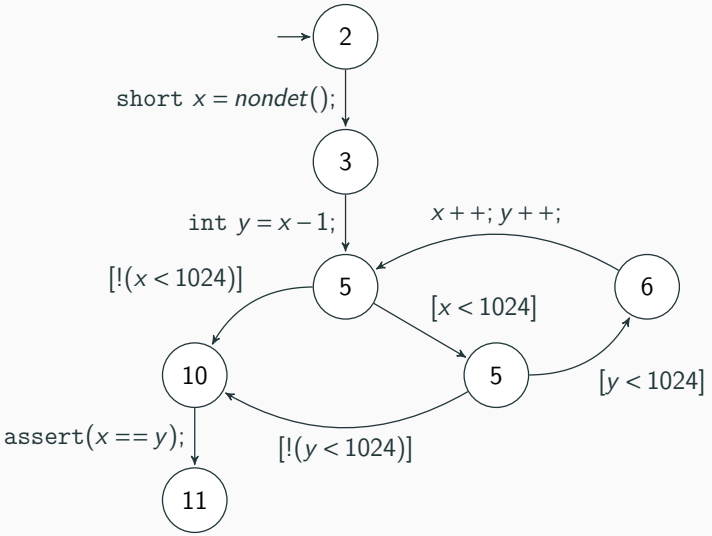


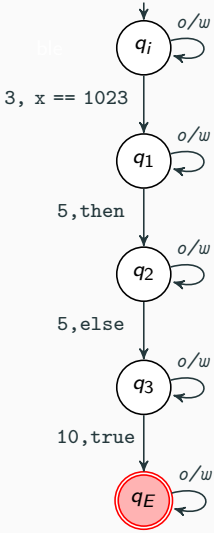
Software Verification Witnesses 1.0: Unclear semantics





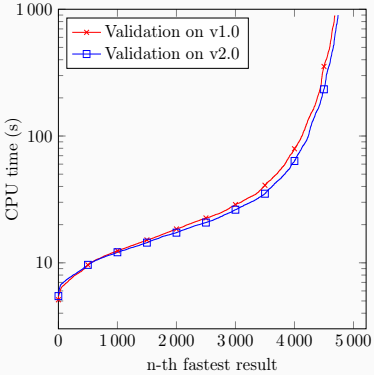
Software Verification Witnesses 1.0: Unclear semantics



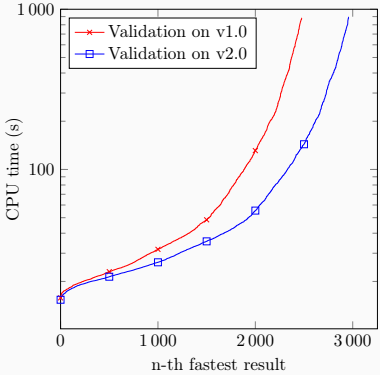


Validation results!

Version 1.0 vs. Version 2.0: Validation Correctness



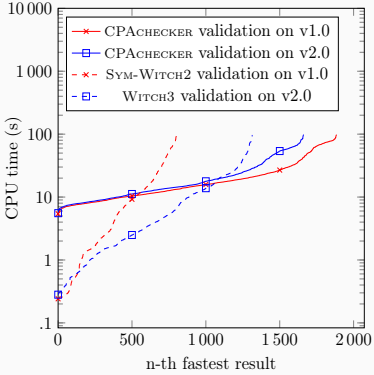
(d) Validation using CPAchecker



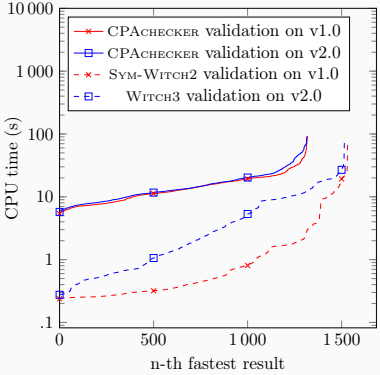
(e) Validation using UAutomizer

Correctness witnesses produced by CPAchecker: Quantile plots for the time taken for validation of the old and new witnesses for two different validators

Version 1.0 vs. Version 2.0: Validation Violation



(f) Validation of violation witnesses generated by CPAchecker



(g) Validation of violation witnesses generated by Symbiotic

Violation witnesses: Quantile plots for the time taken for validation of the old and the new witnesses generated by two different verifiers for two different validators

Adoption in SV-COMP 2024!

Adoption of Witness Version 2.0 in SV-COMP 2024

	Witness Generation				Witness Validation			
	Correctness		Violation		Correctness		Violation	
Tool	v1.0	v2.0	v1.0	v2.0	v1.0	v2.0	v1.0	v2.0
CPAchecker	•	•	•	•	•	•	•	•
Symbiotic	•		•	•				
Symbiotic-Witch2							•	
Witch3								•
UAutomizer	•	•	•		•	•	•	
UKojak	•	•	•					
UTaipan	•	•	•					
UGemCutter	•	•	•					
Mopsa	•	•				•		
CPV	•	•	•					
Goblint	•	•				•		