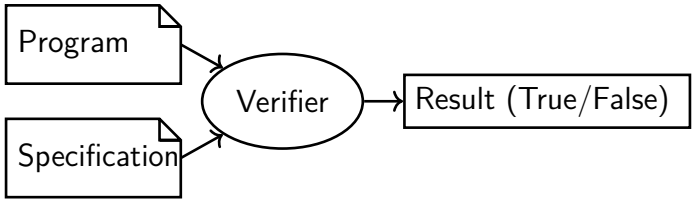
Software Verification Witnesses

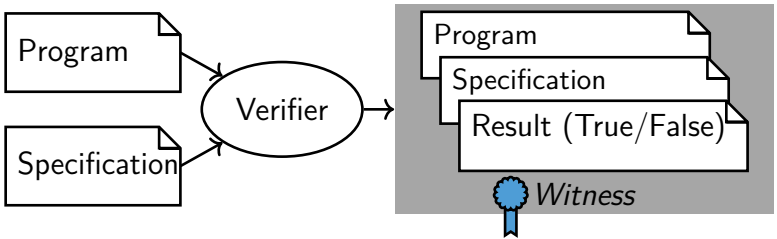
Marian Lingsch-Rosenfeld

LMU Munich, Germany

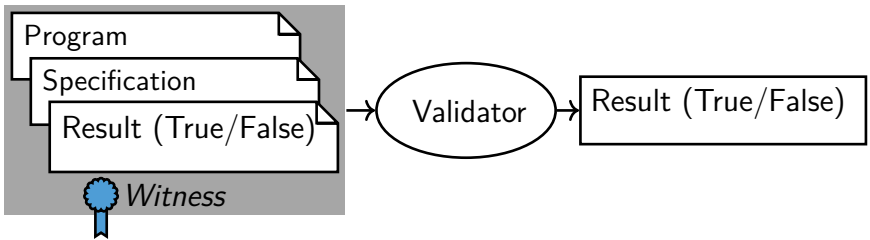
Software Verification



Software Verification with Witnesses



Witness Validation



- Validate untrusted results
- Easier than full verification

Correctness Witness 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"
```

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'
```

Purpose of Witnesses

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

Provide insights into the verification!

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

- 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

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

Validation!

Validation

- **LIV** [5]
- CPAchecker with K-Induction [1]
- MetaVal [4]
- UAutomizer [7]
- ...

Hoare Style Proof

```
1  int x = nondet();
2  int y = x;
3  //@ loop invariant x == y;
4  while (x<1024) {
5    x++;
6    y++;
7  }
8  assert (x == y);
```

$$\dfrac{\{P\}s_0\{R\} \qquad \dfrac{R \Rightarrow I \qquad \{I \wedge C\}B\{I\} \qquad \neg C \wedge I \Rightarrow Q}{\{R\}\ \texttt{while}\ C\ \texttt{do}\ B\ \{Q\}}\ \text{while}}{\{P\}s_0;\ \texttt{while}\ C\ \texttt{do}\ B\{Q\}}\ \text{comp}$$

Proof Obligations:

- $\{P\}s_0\{I\}$
- $\{I \wedge C\}B\{I\}$
- $\{\neg C \wedge I\}\{Q\}$

From Proof Obligations to Straight-Line Programs

Proof Obligations:

- $\{P\}s_0\{I\}$ (Reachability)
- $\{I \wedge C\}B\{I\}$ (Inductiveness)
- $\{\neg C \wedge I\}\{Q\}$ (Safety)

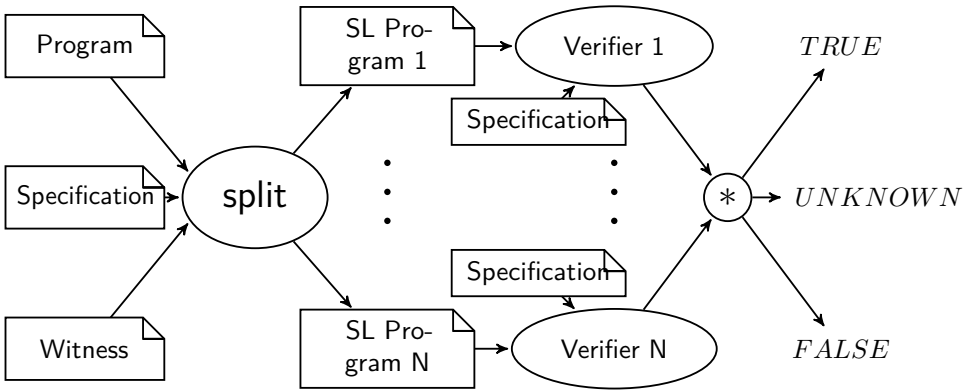
Straight-Line Programs:

```
int x = nondet();
int y = x;
assert (x == y);
```

```
int x = nondet();
int y = nondet();
assume(x == y && x < 1024);
x++;
y++;
assert (x == y);
```

```
int x = nondet();
int y = nondet();
assume(x == y);
assert (x == y);
```

LIV


Program
Specification
Witness
split
SL Pro-
gram 1
SL Pro-
gram N
Specification
Specification
Verifier 1
Verifier N
*
TRUE
UNKNOWN
FALSE

Information Exchange!

Information Exchange

- **CEGAR-PT** [3]
- Component Based CEGAR [2]
- Cooperation Between Automatic and Interactive Software Verifiers [6]
- ...

Program Transformation

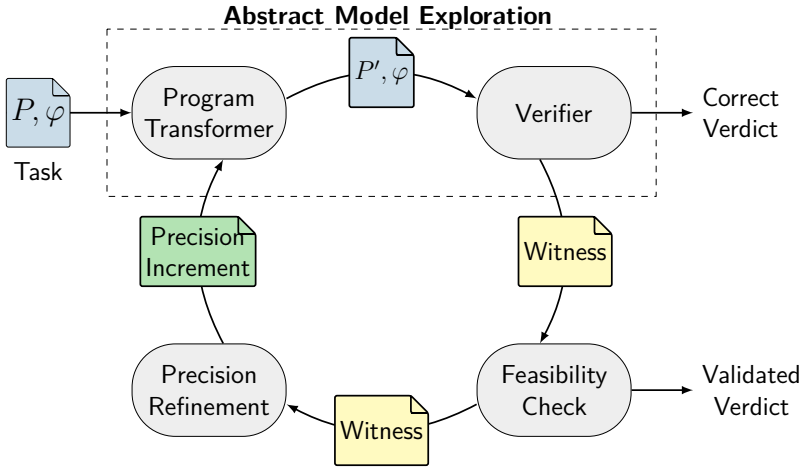
```
1  int main() {
2     int y = nondet();
3     if (y < 100) {
4        while (y < 10000) {
5           y += 1;
6        }
7        assert(y == 10000);
8     }
9  }
```

Figure: An example program.

```
1  int main() {
2     int y = nondet();
3     if (y < 100) {
4
5        y = 10000;
6
7        assert(y == 10000);
8     }
9  }
```

Figure: An easier program to verify producing the same result.

CEGAR-PT


Abstract Model Exploration
P, φ
Task
Program Transformer
P', φ
Verifier
Correct Verdict
Precision Increment
Witness
Precision Refinement
Witness
Feasibility Check
Validated Verdict

CEGAR-PT: Example

```
1  int y = nondet();
2  if (y < 100) {
3      while (y < 10000)
4          y += 1;
5      assert(y == 10000);
6  } else {
7      while (y > 0)
8          y -= 1;
9  }
```

CEGAR-PT: Example

```
1  int y = nondet();
2  if (y < 100) {
3
4      while (y < 10000)
5          y += 1;
6
7      assert(y == 10000);
8  } else {
9
10     while (y > 0)
11         y -= 1;
12
13 }
```

```
1  int y = nondet();
2  if (y < 100) {
3      // START HAVOCSTRATEGY
4      if (y < 10000) y = nondet();
5      if (y < 10000) abort();
6      // END HAVOCSTRATEGY
7      assert(y == 10000);
8  } else {
9      // START HAVOCSTRATEGY
10     if (y > 0) y = nondet();
11     if (y > 0) abort();
12     // END HAVOCSTRATEGY
13 }
```

CEGAR-PT: Example

```c
int y = nondet();
if (y < 100) {
    // START HAVOCSTRATEGY
    if (y < 10000) y = nondet();
    if (y < 10000) abort();
    // END HAVOCSTRATEGY
    assert(y == 10000);
} else {
    // START HAVOCSTRATEGY
    if (y > 0) y = nondet();
    if (y > 0) abort();
    // END HAVOCSTRATEGY
}
```

```yaml
<...>
  content:
    - segment:
        - waypoint:
            type: branching
            location:
              line: 2
              file_name: 'example.c'
            constraint:
              value: true
    - segment:
        - waypoint:
            type: branching
            location:
              line: 4
              file_name: 'example.c'
            constraint:
              value: true
    - segment:
        - waypoint:
            type: target
            location:
              line: 7
              file_name: 'example.c'
```

CEGAR-PT: Example

```
1  int y = nondet();
2  if (y < 100) {
3      // START HAVOCSTRATEGY
4      if (y < 10000) y = nondet();
5      if (y < 10000) abort();
6      // END HAVOCSTRATEGY
7      assert(y == 10000);
8  } else {
9      // START HAVOCSTRATEGY
10     if (y > 0) y = nondet();
11     if (y > 0) abort();
12     // END HAVOCSTRATEGY
13 }
```

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

CEGAR-PT: Example

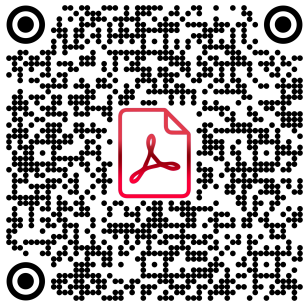
```
1  int y = nondet();
2  if (y < 100) {
3      // START HAVOCSTRATEGY
4
5
6      if (y < 10000) y = nondet();
7      if (y < 10000) abort();
8
9
10     // END HAVOCSTRATEGY
11     assert(y == 10000);
12 } else {
13     // START HAVOCSTRATEGY
14     if (y > 0) y = nondet();
15     if (y > 0) abort();
16     // END HAVOCSTRATEGY
17 }
```

```
1  int y = nondet();
2  if (y < 100) {
3      // START NAIVELOOPACCELERATION
4      if (y < 10000) {
5          y = nondet();
6          if (!(y < 10000)) abort();
7          if (y < 10000) y += 1;
8          if (y < 10000) abort();
9      }
10     // END NAIVELOOPACCELERATION
11     assert(y == 10000);
12 } else {
13     // START HAVOCSTRATEGY
14     if (y > 0) y = nondet();
15     if (y > 0) abort();
16     // END HAVOCSTRATEGY
17 }
```

Conclusion

Witnesses contain machine-readable data to:

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



Paper: witnesses version 2.0

[1] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_42

[2] 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

[3] 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

[4] Beyer, D., Spiessl, M.: MetaVal: Witness validation via verification. In: Proc. CAV. pp. 165–177. LNCS 12225, Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_10

[5] Beyer, D., Spiessl, M.: LIV: A loop-invariant validation using straight-line programs. In: Proc. ASE. pp. 2074–2077. IEEE (2023). https://doi.org/10.1109/ASE56229.2023.00214

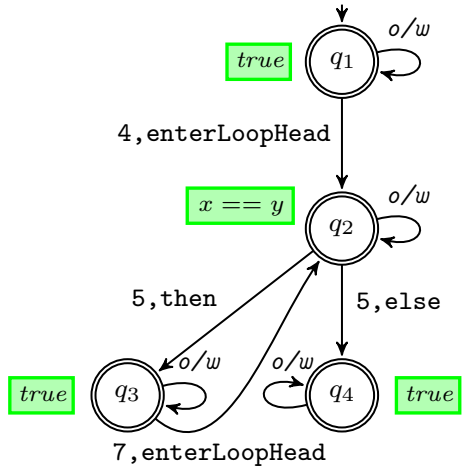
[6] Beyer, D., Spiessl, M., Umbricht, S.: Cooperation between automatic and interactive software verifiers. In: Proc. SEFM. p. 111–128. LNCS 13550, Springer (2022). https://doi.org/10.1007/978-3-031-17108-6_7

[7] Heizmann, M., Barth, M., Dietsch, D., Fichtner, L., Hoenicke, J., Klumpp, D., Naouar, M., Schindler, T., Schüssele, F., Podelski, A.: Ultimate Automizer 2023 (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)

Witnesses Version 1.0

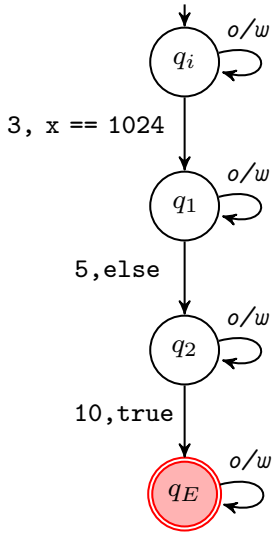
Correctness Witnesses 1.0

```
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 }
```



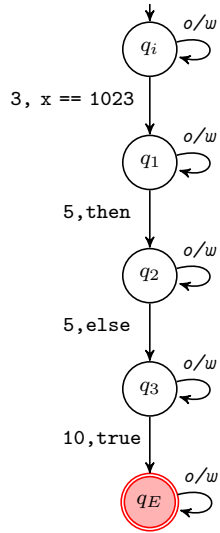
Why do we need Version 2.0?

Software Verification Witnesses 1.0: Unreadable Files

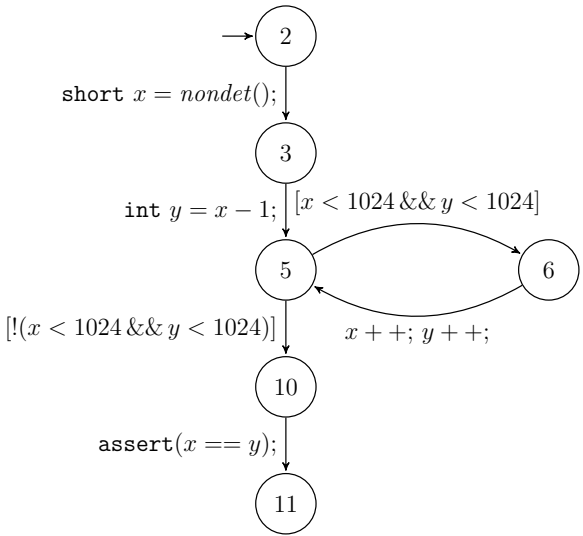
```
 1 <node id="A0">
 2 <data key="entry">true</data>
 3 </node>
 4 <node id="A2_3_1"/>
 5 <edge source="A0" target="A2_3_1">
 6 <data key="startline">4</data>
 7 <data key="endline">4</data>
 8 <data key="enterFunction">main</data>
 9 </edge>
10 <node id="A2"/>
11 <edge source="A2_3_1" target="A2">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

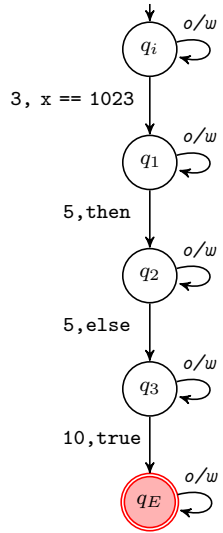
Software Verification Witnesses 1.0: Unclear semantics

```
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