Exercise 11

2022/01/07

Reading Material

Please read the following material:

(1) Dirk Beyer, Matthias Dangl, Daniel Dietsch, Matthias Heizmann, Andreas Stahlbauer. Witness Validation and Stepwise Testification across Software Verifiers. Proc. FSE, 2015.
(please skip the appendix)

(2) Dirk Beyer, Matthias Dangl, Daniel Dietsch, Matthias Heizmann. Correctness Witnesses: Exchanging Verification Results Between Verifiers. Proc. FSE, 2016.
(please **don't** try to understand Ultimate Automizer's verifier and validator in detail)

Questions

Some short questions to get you thinking about the reading material:

Reading Material (1)

1. What is the difference between a *protocol automaton* and an *observer automaton*?
2. Which automaton should be used to express *violation witnesses*?
3. Do violation witnesses have to describe a single error path?

Reading Material (2)

1. Which automaton should be used to express *correctness witnesses*?
2. Which invariant does the correctness witness in Fig. 1(b) of the reading material describe (at which program location)?
3. Which technique does CPAchecker use to validate correctness witnesses? Does it produce its own invariants?

Witness exchange format

A detailed definition of the GraphML Witness-Exchange Format is available here:
`https://github.com/sosy-lab/sv-witnesses`

Composition $\mathbb{F} \times \mathbb{O}$

For witness validation in this exercise sheet, we will use the composition of the formula-based CPA $\mathbb{F}$ (from exercise sheet 3, exercise 3) and the protocol analysis $\mathbb{O}$ from the reading material. The composite analysis is similar to the combination of predicate analysis and observer automata in Sect. 16.5.4 of the Handbook of Model Checking:

1. The composite domain $D_x = D_{\mathbb{F}} \times D_{\mathbb{O}}$ is the product of the component domains $D_{\mathbb{F}}$ and $D_{\mathbb{O}}$.
2. The transfer relation $\rightsquigarrow_x$ has the transfer $(\varphi, (q, \psi)) \overset{g}{\rightsquigarrow}_x (\varphi'', (q', \psi'))$ if $\varphi \overset{g}{\rightsquigarrow}_{\mathbb{F}} \varphi'$ and $(q, \psi) \overset{g}{\rightsquigarrow}_{\mathbb{O}} (q', \psi')$, and if $\downarrow_{\mathbb{F}\times\mathbb{O}}(\varphi', (q', \psi'))$ is defined.
   The strengthening operator $\downarrow_{\mathbb{F}\times\mathbb{O}}(\varphi', (q', \psi'))$ is defined if and only if $\varphi' \wedge \psi'$ is satisfiable. Then, $\varphi'' = \downarrow_{\mathbb{F}\times\mathbb{O}}(\varphi', (q', \psi')) = \varphi' \wedge \psi'$.
3. The merge operator is $merge^{sep}$.
4. The stop operator is $stop^{sep}$.

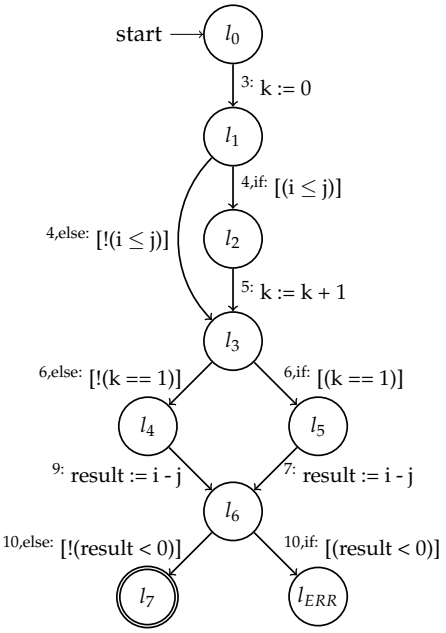
Composition of multiple CPAs

It is possible to compose more than two CPAs in a composite CPA by consecutive composition: We write $\mathbb{D}_0 \times \mathbb{D}_1 \times \mathbb{D}_2$ instead of $(\mathbb{D}_0 \times \mathbb{D}_1) \times \mathbb{D}_2$.

1 Violation Witnesses

```
1  int i, j; // defined, but arbitrary value
2  int result;
3  int k = 0;
4  if (i <= j)
5      k = k + 1;
6  if (k == 1)
7      result = i - j;
8  else
9      result = i - j;
10 if (result < 0)
11     ERR:;
```

Program P_e

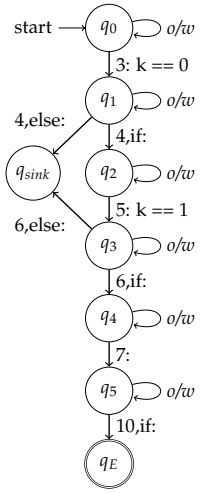


CFA A_e

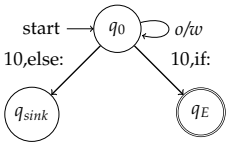
Above you see a faulty program P_e and its CFA A_e. To make the exercise clearer, each CFA edge lists the file location it was created from (e.g., $^{4,\text{if:}}$).

1. For each violation witness below, list all (syntactic) program paths described by that witness. (Note that the notation "*source-code guard : state-space guard*" below for witness automata is from the reading material. It differs only syntactically from the notation (D, ψ) for observer automata used in exercise sheet 7. A blank state-space guard means *true*.)

Witness a:



Witness b:



Witness c:

```
<graphml>
  <!-- .. snip metadata .. -->
  <graph>
  <node id="A0">
    <data key="entry">true</data>
  </node>
  <node id="A2" />
  <edge source="A0" target="A2">
    <data key="startline">1</data>
    <data key="assumption">i == 1; j == 2;</data>
    <data key="assumption.scope">main</data>
  </edge>
  <node id="A75">
    <data key="violation">true</data>
  </node>
  <edge source="A2" target="A75">
    <data key="startline">10</data>
    <data key="control">condition-true</data>
  </edge>
  <node id="sink">
    <data key="sink">true</data>
  </node>
  <edge source="A2" target="sink">
    <data key="startline">10</data>
    <data key="control">condition-false</data>
  </edge>
  </graph>
</graphml>
```

2. **(asynchronous)**
   Write the GraphML representation of *Witness b* (without metadata).

3. **(asynchronous)**
   Perform violation-witness validation for *Witness a*:

   a) Draw the observer automaton A_{ERR} for the property "label ERR is never reached".

   b) Apply the CPA algorithm with the composite CPA $\mathbb{L} \times \mathbb{F} \times \mathbb{O}_{ERR} \times \mathbb{O}_w$, where $\mathbb{O}_{ERR}$ is the protocol analysis for the observer automaton A_{ERR} and $\mathbb{O}_w$ is the protocol analysis for *Witness a*.

      Is *Witness a* confirmed or rejected?

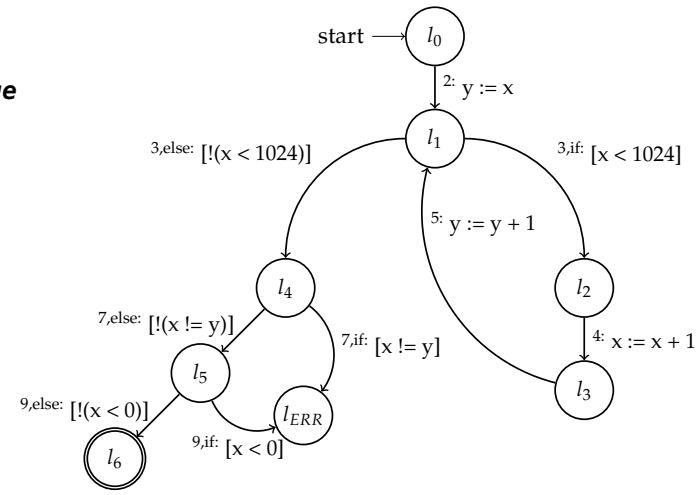
4. For some witnesses A and B, we say $A <_{testified} B$ if and only if witness A describes a subset of the state-space that is described by witness B.
   Check all correct statements:

   ☐ *Witness a* $<_{testified}$ *Witness b*
   ☐ *Witness b* $<_{testified}$ *Witness a*
   ☐ *Witness b* $<_{testified}$ *Witness c*
   ☐ *Witness c* $<_{testified}$ *Witness b*
   ☐ *Witness a* $<_{testified}$ *Witness c*
   ☐ *Witness c* $<_{testified}$ *Witness a*

2 Correctness Witnesses

```
1  int x; // defined, but arbitrary value
2  int y = x;
3  while (x < 1024) {
4      x = x + 1;
5      y = y + 1;
6  }
7  if (x != y)
8      goto ERR;
9  if (x < 0)
10     ERR:;
```

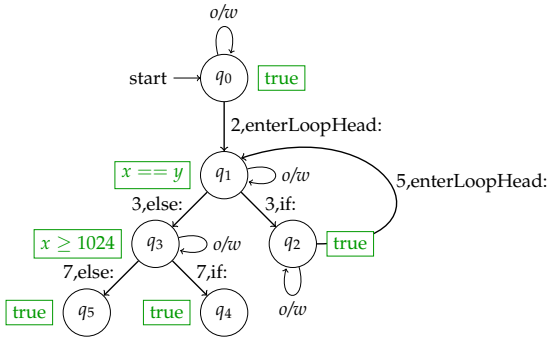
Program P_c



CFA A_c

1. For each correctness witness below, list all candidate invariants (l, i) described by that witness. Each candidate invariant (l, i) consists of a program location $l \in A_c$ where the invariant is supposed to hold, and the invariant i.

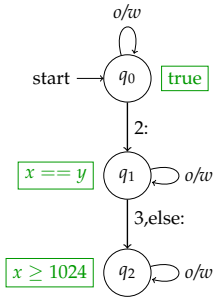
Witness d:



Witness e:

```
1  <graphml>
2    <!— .. snip metadata .. —>
3    <graph>
4    <node id="q0">
5      <data key="entry">true</data>
6    </node>
7    <node id="q1">
8      <data key="invariant">x == y</data>
9      <data key="invariant.scope">main</data>
10   </node>
11   <edge source="q0" target="q1">
12     <data key="enterLoopHead">true</data>
13     <data key="startline">2</data>
14   </edge>
15   </graph>
16 </graphml>
```

Witness f:



2. Perform the base step of correctness-witness validation for *Witness d*:

   a) Convert *Witness d* into an equivalent observer automaton A_{Inv} that ensures that the specified invariants always hold.

      If a specified invariant is violated at all possible program locations it matches to, the observer automaton should report a violation.

   b) **(asynchronous)**
      Use the CPA algorithm to check for counterexamples to the specified invariants for loop-bound $k = 1$. Apply the CPA algorithm with the composite CPA $\mathbb{L} \times \mathbb{F} \times \mathbb{LB} \times \mathbb{O}_{\mathbb{ERR}} \times \mathbb{O}_{\mathbb{I}}$, where $\mathbb{O}_{\mathbb{ERR}}$ is the protocol analysis for automaton A_{ERR} from before, and $\mathbb{O}_{\mathbb{I}}$ is the protocol analysis for automaton A_{Inv}.

      Does this report any counterexamples to the invariants?

3. After performing the base step for $k = 1$, how would correctness-witness validation in CPAchecker continue to prove the invariants valid? Provide a short sketch of the final steps.