Lehrstuhl für Software and Computational Systems
Dirk Beyer, Thomas Lemberger

Exercise

2021/02/19

Reference: https://www.sosy-lab.org/research/pub/2018-HBMC.Combining_Model_Checking_and_Data-Flow_Analysis.pdf

1 Hands-on

1.1 (Bounded) Model Checking

Jupyter Notebook: 03_BMC.ipynb

1.2 Configurable Program Analysis

Jupyter Notebook: 07_Verifier-Design-part-1.ipynb

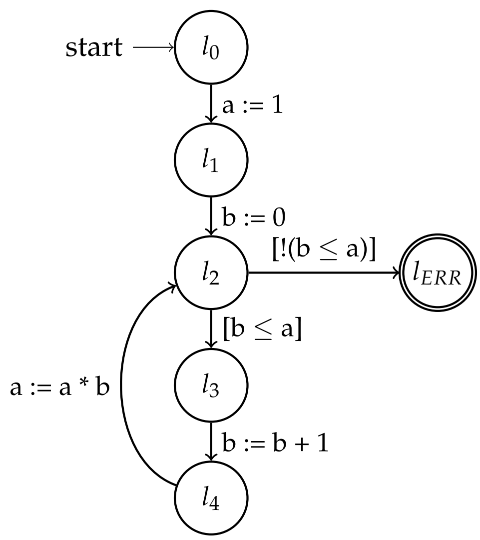
2 Theory

2.1 Observer Automata

```
int a = 1;
int b = 0;

while (b <= a) {
    b = b + 1;
    a = a * b;
}
ERR:;
```

Program



CFA $P = (L, E, l_0)$

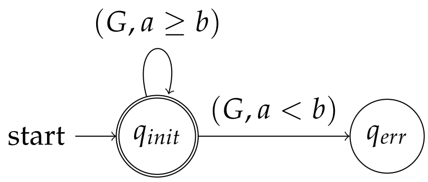
1. Define an observer automaton for the given program $P = (L, E, l_0)$ and program variables X, for each of the following specifications:

    a) $\Box(l' \neq l_{ERR})$ (for $g = (l, op, l')$)

    b) $\forall x \in X.\ \forall z \in X.\ \Box(op = [x \leq z] \implies \circ(\forall y \in X \cup \mathbb{Z}.\ op \neq [x \leq y]\ \mathcal{W}\ op = x := x + 1))$

    c) $\forall x \in X.\ \forall y \in X. \Box(op = x := x * y \implies y > 0)$

2. Consider the following observer automaton A:



   a) State the LTL-formula equivalent of A.

   b) Consider observer analysis $\mathbb{O}$ for observer automaton A and precision π:

$$\pi = \{x = n \mid x \in X, n \in \mathbb{Z}\} \cup \{x \geq n \mid x \in X, n \in \mathbb{N}\} \cup \{b \leq a, b \leq a+1\} \cup \{false\}$$

   Apply the CPA algorithm with composite analysis $\mathbb{L} \times \mathbb{P} \times \mathbb{O}$ and initial state $e_0 = (l_0, \emptyset, (q_{init}, true))$ to program P. State the final reached set and whether the specification is violated, according to the algorithm.

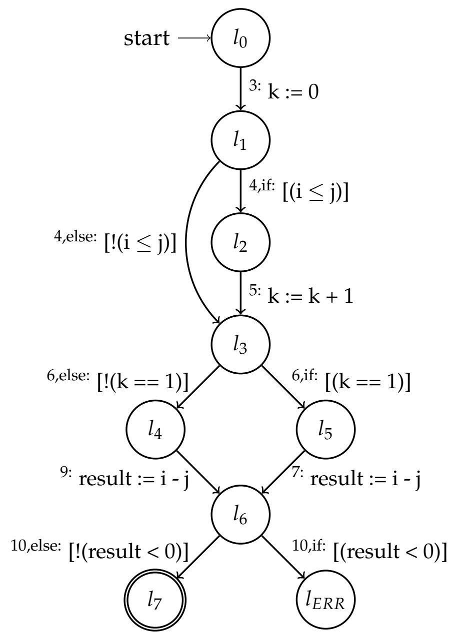
2.2 Verification-Result Witnesses (45 minutes)

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

2.2.1 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

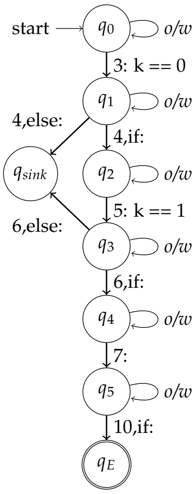


CFA A_e

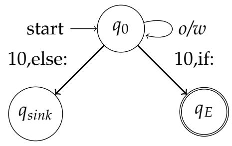
Above you see faulty program P_e and its CFA A_e. Each CFA edge lists the file location it was created from (e.g., $^{4,\text{if:}}$).

1. For each violation witness below, list all (syntactic) program paths described by that witness:

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>
```

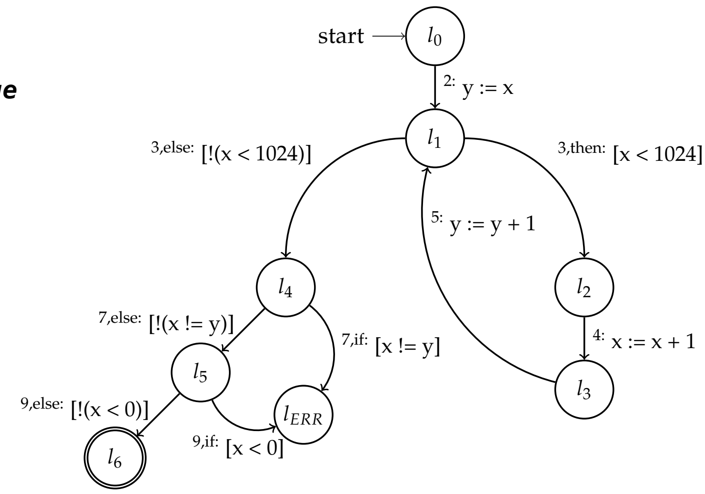
2. For some witnesses A and B, we say $A <_{testified} B$ iff witness A describes a subset of the state-space that is described by witness B.
   Check all correct statements:

   - ☐ *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*

2.2.2 Correctness Witnesses

```
int x; // defined, but arbitrary value
int y = x;
while (x < 1024) {
    x = x + 1;
    y = y + 1;
}
if (x != y)
    goto ERR;
if (x < 0)
    ERR:;
```

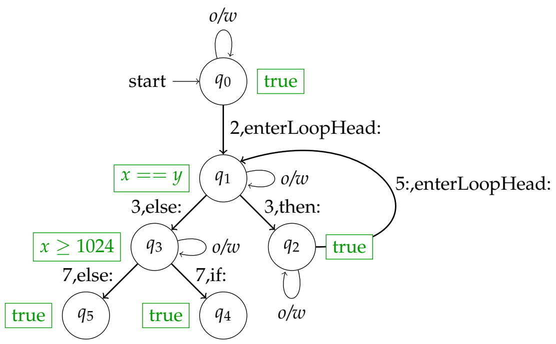
Program P_c



CFA A_c

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:

```
<graphml>
  <!-- .. snip metadata .. -->
  <graph>
  <node id="q0">
    <data key="entry">true</data>
  </node>
  <node id="q1">
    <data key="invariant">x == y</data>
    <data key="invariant.scope">main</data>
  </node>
  <edge source="q0" target="q1">
    <data key="enterLoopHead">true</data>
    <data key="startline">2</data>
  </edge>
  </graph>
</graphml>
```

Witness f: