Dr. Matthias Heizmann
Tanja Schindler
Dominik Klumpp

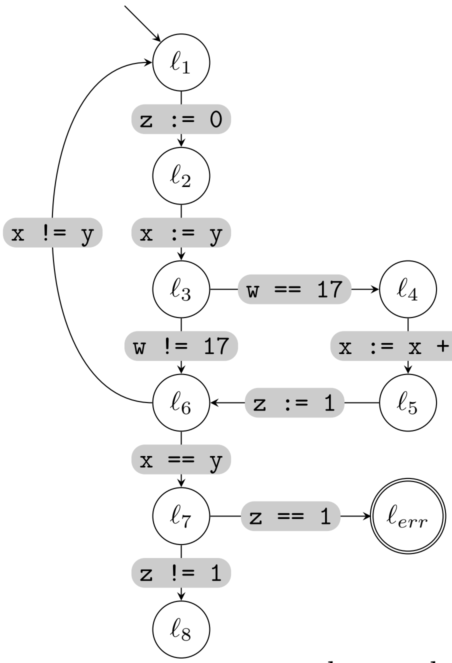
Hand in until 10:00 on July 22, 2019
Discussion: July 22, 2019

Tutorial for Program Verification
Exercise Sheet 22

Exercise 1: Trace Abstraction 3 Points

In this task, you should apply trace abstraction to prove that a program, here given by its control-flow graph, is safe.

Consider the following control-flow graph for a program P, and let $\mathcal{A}_P$ be the corresponding automaton.



Give two error traces π_1 and π_2 and construct corresponding Floyd-Hoare automata $\mathcal{A}_1$ and $\mathcal{A}_2$ such that the inclusion $L(\mathcal{A}_P) \subseteq L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ holds.

Exercise 2: Termination 2 Points

In the lecture, we discussed four different properties of programs. One property was *termination* the other properties where related to termination. We provide formal definitions here. In each case, we consider a program P with a CFG $(Loc, \Delta, \ell_{\mathsf{init}}, \ell_{\mathsf{ex}})$.

(a) We say that P *can reach the exit location* if there exists a finite execution, such that the first configuration (ℓ, s) is initial, and the last configuration is (ℓ_{ex}, s') for some state s'.

(b) We say that P *can stop* if there exists a reachable configuration (ℓ, s) such that there exists no configuration (ℓ', s') and statement st with $(\ell, st, \ell') \in \Delta$ and $(s, s') \in [\![st]\!]$.

(c) We say that P *always reaches the exit location* if there exist no infinite executions, and all finite executions end in a configuration (ℓ', s') where we either have a successor (i.e., there exists a configuration (ℓ'', s'') and statement st with $(\ell', st, \ell'') \in \Delta$ and $(s', s'') \in [\![st]\!]$) or we have that ℓ' is ℓ_{ex}.

(d) We say that P *always stops* (resp. P *terminates* if there exist no infinite executions.

In this exercise, you should give programs that differentiate between these definitions. In particular, for each of the following pairs, give a program such that one definition holds but the other does not. Explain which of the definitions holds and why.

(a) *P can reach the exit location* **vs.** *P can stop*

(b) *P can stop* **vs.** *P always stops*

Exercise 3: Ranking Functions 5 Points

For each of the following programs, state whether it (always) terminates or not. If it terminates, give a ranking function for each loop in the program. If it may not terminate, give an infinite execution of the program.

```
while (x > 0) {
  while (y > 0) {
    y := y-1;

  }
  x := x-1;
  havoc y;

}
```

Listing 1: Program P_1

```
while (x > 0) {
  if (y > 0) {
    y := y-1;

  } else {
    x := x-1;
    havoc y;
  }
}
```

Listing 2: Program P_2

```
while (x > 0) {
  if (y > 0) {
    y := y-1;
    havoc x;
  } else {
    x := x-1;
    havoc y;
  }
}
```

Listing 3: Program P_3

Hint: For simple loops is often convenient to use a function whose range is $\mathbb{N}$ and the strictly greater than relation $>$ on natural numbers. For more complex loops, this is sometimes not sufficient but we can use instead a function $f : S_{V,\mu} \to \mathbb{N}_1 \times \ldots \times \mathbb{N}_n$ whose range are n-tuples of natural numbers and the *lexicographic order* $>_{\mathsf{lex}}$ that we define as follows.

$(m_1, \ldots, m_n) >_{\mathsf{lex}} (m'_1, \ldots, m'_n)$ iff there exists $i \in \{1, \ldots n\}$ such that $m_i > m'_i$ and for all $k \in \{1, \ldots i-1\}$ the equality $m_k = m'_k$ holds

If a function with that signature together with the order $>_{\mathsf{lex}}$ is a ranking function, it is often called a *lexicographic ranking function*.