1 Trace Properties

Question 1.1. Recall the approach for defining trace properties introduced in the lecture. For example, the property ZeroX, describing all traces terminating with a state mapping the program variable x to 0, is formally defined below:

\[
\text{ZeroX} \triangleq \{ [\langle \rho_0, s_0 \rangle, \ldots, \langle \rho_n, s_n \rangle] \mid s_n = \text{skip} \land \rho_n(x) = 0 \\
\land \forall 0 \leq i < n (\rho_i, s_i) \rightarrow (\rho_{i+1}, s_{i+1}) \}
\]

Give the formal definition of the following trace properties:

1. AllZero, describing the traces that terminate with a state mapping all of its variables to zero;
2. MonX, describing all traces such that the value of the program variable x never gets decremented;
3. AllMon, describing all traces such that the values of all program variables never get decremented;
4. XGreaterThanY, describing all traces such that, whenever both variables x and y are defined, x is greater than y;
5. XGreaterThanAll, describing all traces such that, whenever x is defined, it is greater than all other program variables;
6. YBookKeepX, describing all traces such that the variable x is only allowed to swap sign once, and, if it does, the variable y will eventually be set to the old value of x and will keep that value until the execution finishes.

Which of the properties above are liveness properties and which are safety properties? Justify your answer.

Question 1.2. Recall the syntax of WHILE programs introduced in the lecture:
\[
e_1, e_2 \in E \triangleq n | x | e_1 \oplus e_2 \\
s_1, s_2 \in S \triangleq \text{skip} | \text{x := e} | s_1; s_2 | \text{if (e) \{ s_1 \} else \{ s_2 \} | while (e) \{ s_1 \}}
\]
For each property in Question 1.1, write a `While` statement that satisfies the property and a `While` statement that does not. For the statement that does not, give a program trace that exhibits the bug.

2 Self-Composition and Symbolic Execution

**Question 2.1.** Consider the following `WHILE` statements:

1. if (h) { l := l + z } else { skip }
2. if (h) { x := x + z } else { skip }
   if (!h) { y := y + z } else { skip }
   l := x + y; x := 0; y := 0;
3. y := 1;
   while (x > 0) {
     if (y > h) { skip } else { y := y * x };
     x := x - 1
   }

Use self-composition and symbolic execution to check which of the above programs satisfy non-interference. Assume the standard lattice \( \mathcal{L} = \langle \{L, H\}, \sqsubseteq \rangle \) and a security labelling, \( \Gamma \), such that only the program variable \( h \) is mapped to the security level \( H \).

**Question 2.2.** Which of the programs above would be considered secure by a standard type system for information flow control? Which would be considered insecure? What can be concluded about the precision of self-composition + symbolic execution when compared to type systems for information flow control?