Dynamic Analysis – Exercises

Ana Matos
Software Security 2020/2021, IST

November 29, 2020

1 Instrumenting an interpreter to monitor executions

1. Download an off-the-shelf interpreter of a simple imperative language. For example, this one:

   `http://jayconrod.com/posts/37/a-simple-interpreter-from-scratch-in-python-part-1`.

   Understand the code that performs evaluation (in the above case, `imp_ast.py`).

2. Modify the interpreter so that it controls information flows as it executes programs.

3. Analyse the precision of the resulting monitorization.

2 Small Step Semantics

1. Consider the Structural Operational Semantics (a.k.a Small Step Semantics) for the WHILE language, as was presented in class. The following exercises are closely based on examples and exercises that appear in Nielson & Nielson 1992 (†).

   (a) Consider the statement

   $$y := 1; \text{while } x \neq 1 \text{ do } (y := y \times x; x := x - 1)$$

   and let ρ be a state with $\rho(x) = 3$. Construct a derivation sequence that allows to determine the final state resulting from the evaluation of this statement.

   Answer: See Example 2.15 of † .

   (b) Construct a derivation sequence for the statement

   $$z := 0; \text{while } y \leq x \text{ do } (z := z + 1; x := x - y)$$

   when executed in a state where x has the value 17 and y has the value 5. Determine a state ρ such that the derivation sequence obtained for the above statement and ρ is infinite.

2. Now consider the Structural Operational Semantics (a.k.a Small Step Semantics) for the Dynamic WHILE language, as was presented in class.

   (a) Construct a derivation sequence for the statement

   $$y := 1; \text{eval } z$$

   when executed in a state where x has value 3 and z has the value "if $x \neq 1$ then $((y := y \times x; x := x - 1); \text{eval } z)$ else skip".

3 Lock-step monitor

Consider the lock-step monitor studied in class.

1. For the following program, show that the monitored execution satisfies Noninterference.

   $$y_H := "x_L"; \text{if } w_H \text{ then } z_H := "True" \text{ else } z_H := "False"; \text{eval } (y_H + + ":=" + + z_H)$$

2. For each of the restrictions imposed by the monitor rules, give an example of an insecure program execution that would be rejected by it.

3. Formulate rigorously the result which states that all terminating executions that the monitored program can make, also the unmonitored program can.

4 Monitor inlining compiler

Consider the lock-step monitor studied in class.

1. For the following program, obtain the compiled program, and show that it satisfies Noninterference.

   $$y_H := "x_L"; \text{if } w_H \text{ then } z_H := "True" \text{ else } z_H := "False"; \text{eval } (y_H + + ":=" + + z_H)$$

2. Formulate rigorously the result which states that all terminating executions that the compiled program can make, also the original program can. Would this result be true if the rule for compiling the assignment had been:

   $$[x := a]^{pc}_{\Gamma} \;=\; \text{if } (pc \sqcup \text{lev}(a) \sqsubseteq \Gamma(x)) \text{ then } x := a \text{ else skip}$$

3. What language features would be necessary for the inlined monitor to be fully executable in the language?