IB Semantics - Supervision 3

Nandor Licker <nl364@cl.cam.ac.uk>

Due noon two days before supervision

We are going to consider static and dynamic taint tracking in the context of L2. The goal of taint tracking is to approximate the set of values which depend on a specific variable - usually some value obtained from an untrustworthy sources, such as the network. Taint information can be used to identify security vulnerabilities: if the target of an indirect jump is a value which was tainted, it means that an attacker can influence that value and can potentially hijack the execution of the current program. In order to emulate this, we are going to extend the set of values of the language to store a flag indicating whether the value is tainted or not:

$$n_t \in \mathbb{Z} \times \mathbb{B}, n_t = (n, t)$$

$$b_t \in \mathbb{B} \times \mathbb{B}, b_t = (b, t)$$

And we extend the language with an input statement introducing tainted integers:

$$\langle \texttt{input}(n), s \rangle \to \langle n_\text{t}, s \rangle$$

1. Taint tracking can be performed statically with the help of the type system. Extend the set of types T to include tainted types and adjust the typing rules of L2 to propagate taint information:

   $$T ::= ... | T'$$

   For example, the type of $\texttt{input}(n)$ should be $\texttt{int}'$. What are the limitations of this approach, thinking of conditionals and the store?

2. Provide operational semantics to perform taint tracking dynamically. Think carefully about conditionals, ensuring that taint information is preserved across control dependencies: if the condition of an `if` statement is tainted, then all computations inside either branches of the if statement should be tainted as well. What are the benefits of the dynamic approach?

3. Consider the following language:

   $$E = n | x | \lambda x.E | E_1 E_2$$

   Provide typing judgements and reduction rules for this language. Show if a term e reduces to a value n, then $n \in \text{ints}(e)$. Define $\text{ints}(e)$ to be the set of integers in a term e.

4. Extend the language with a rule for addition. If an expression in the previous language does not contain any odd numbers and evaluates to a number, prove the number is even.

5. Extend the language with a rule for addition. Declare and prove a property relating the value an expression can evaluate to with the constants of the expression.

6. Define a predicate $\text{tainted}(e, s)$ which holds if neither the expression, nor the store is tainted. Show that if an expression with a non-tainted initial store and no `input` terms evaluates to a value, the value is not tainted.

7. Provide an example of an expression which contains an `input` term, but does not evaluate to a tainted value. Show all the reduction steps.

8. Consider the following language:

$$E = n|x|\lambda x.E|\texttt{input}(n)$$

   Provide typing judgements and reduction rules to propagate tainted values originating from the input construct and untainted values introduced by constants. Show if a term e reduces to a value n, then n is one of the integers defined in e, excluding the ones from `input` rules.

9. Extend the language from the previous exercise with a + operator. Assuming an expression e evaluates to an untainted number, prove it is the sum of constants defined in e, minus the ones from `input` expressions.

10. Describe a strategy to prove that non-tainted values produced by the extended version of L2 do not depend on values introduced by `input` rules, making use of the properties of the simplified language.