Homework 4
CSC 7101, Spring 2015
Due: 24 April 2015

1. Translate our operational semantics rules into Prolog clauses. Represent syntactic constructs using functors. E.g., the if-statement could be represented as ifThenElse(B,C0,C1). Represent the three different evaluation functions, <.,.,->., as predicates. Assume that appropriate predicates for arithmetic, comparisons, boolean operations are built in.

Assume that a state is represented as a list of pairs of a variable and a value, and that bindings at the beginning of the list shadow later bindings. E.g., the axioms for numeric constants and variables could be written as follows:

\[
\text{evalArithExp(const (N), Sigma, N).} \\
\text{evalArithExp(var(X), Sigma, Val) :- get(Sigma, X, Val).}
\]

where \(\text{get(Sigma, X, Val)}\) computes the result of \(\sigma(X)\) in \(Val\) and is defined as follows:

\[
\text{get([], X, undefined).} \\
\text{get([[X,V]|T], X, V).} \\
\text{get([H|T], X, V) :- get(T, X, V).}
\]

You don’t need to test and run your Prolog program. Writing it on paper is good enough.

2. Develop semantic rules for our operational semantics system for the following (simplified) loop construct adopted from Ada:

\[
\text{loop } S_1; \text{exit when } B; S_2 \text{ end}
\]

In the above statement, \(S_1\) and \(S_2\) are statements and \(B\) is a boolean expression. The semantics of ‘\text{exit when } B\’ is the same as ‘\text{if } (B) \text{ break;}\’ in C.

3. Consider the following statement:

\[
\langle \text{letstmt} \rangle := \text{let int } \langle id \rangle := \langle \text{intexp} \rangle \text{ in } \langle \text{stmtseq} \rangle \text{ end}
\]

When executing this statement, we first evaluate the integer expression \(\langle \text{intexp} \rangle\). We then bind the variable \(\langle id \rangle\) to the value of this expression, and then evaluate the statement sequence \(\langle \text{stmtseq} \rangle\).

Semantic constraints: The let-statement shadows any outer declaration of the variable \(\langle id \rangle\). I.e., any occurrence of \(\langle id \rangle\) in \(\langle \text{stmtseq} \rangle\) refers to the variable from the let-statement. The scope of this variable is only \(\langle \text{stmtseq} \rangle\). I.e., any occurrence of \(\langle id \rangle\) in \(\langle \text{intexp} \rangle\), in a post-condition of the let-statement, or in a statement following the let-statement refers to some outer declaration of \(\langle id \rangle\). There are no restrictions on the use of variable \(\langle id \rangle\) in \(\langle \text{stmtseq} \rangle\). In particular, it is possible to assign a new value to \(\langle id \rangle\). The let-statement is evaluated strictly. I.e., if \(\langle \text{intexp} \rangle\) does not terminate (assuming that there can be function calls in \(\langle \text{intexp} \rangle\)), then the let-statement does not terminate.

Define the operational semantics of the let-statement. Justify your proposal (informally). If you cannot formally define the semantics according to all the semantic constraints above, define it as good as you can and explain in English what’s missing.
4. Consider Dijkstra’s guarded loop:

\[
\langle c \rangle ::= \textbf{do} \langle be \rangle_1 \rightarrow \langle c \rangle_1 \{ \} \langle be \rangle_2 \rightarrow \langle c \rangle_2 \{ \} \langle be \rangle_3 \rightarrow \langle c \rangle_3 \textbf{od}
\]

where \textbf{do} and \textbf{od} are keywords, \langle be \rangle_i are boolean expressions and \langle c \rangle_i are commands. (Dijkstra’s guarded loop has arbitrarily many clauses; we’ll restrict ourselves to exactly three for simplicity.)

The guarded loop is a generalization of the while loop with multiple conditions (\langle be \rangle_1 \ldots \langle be \rangle_3) and multiple loop bodies (\langle c \rangle_1 \ldots \langle c \rangle_3). The boolean expressions are guards for the commands that follow them. A command \langle c \rangle_i can only be executed if its guard \langle be \rangle_i is true.

Informally, each loop iteration of the guarded loop is executed as follows. All the boolean expressions (guards) are evaluated. If none of the guards \langle be \rangle_i is true, none of the commands \langle c \rangle_i is executed and the loop terminates. If one guard is true, the corresponding command is executed. If more than one guard is true, one of the commands whose guard is true is chosen nondeterministically (at random) to be executed. After execution of the selected command, execution proceeds to the next loop iteration.

The special case in which two of the guards are always false is equivalent to the while loop.

Define the operational semantics of this command. Do not assume a certain deterministic evaluation order. Explain your semantic rules.

5. Suppose we change the intuitive operational model of the language whose semantics we have been defining as follows: Whenever the value of a variable is ‘used’ its value reduces by 1. In other words, expression evaluation has side effects. Thus executing an assignment command like ’\texttt{x := y + z}’ not only assigns a value to \texttt{x}, it also reduces the values of \texttt{y} and \texttt{z} by 1 each.

What changes would you have to make to the operational semantics of the language? Specify all the required changes. If you need to make any assumptions, state these assumptions clearly.