

Homework 3

CSC 7101, Spring 2007

Due: 22 March 2007

1. Consider a standard programming language (say, like C or Pascal). Suppose we have defined the context sensitive syntax of this language using attribute grammars in the same manner as what we discussed in class. Suppose we now wish to add the requirement that all variables must be initialized or assigned a value before they are used (without requiring syntactically that every declaration contains an initializer). E.g., the Java specification has such a requirement.

Is it possible to check this condition in the context-sensitive syntax of the language (using attributes such as *symtab*, *tbl*, etc.)? If yes, explain briefly how you would modify the grammar to check this condition. If no, explain how a code-generation attribute grammar would need to be modified such that this condition is checked at run time. Such a check could be performed independently of the scope rules of the language, so you can restrict yourself to the simple languages discussed in class.

2. For each of the following assertions, state whether it is operationally valid or not. Justify your answers briefly. If for either of these assertions you cannot decide whether it is operationally valid or not, explain why you are not able to decide.

(a) $\{x = 0\}$ **while** $x < 10$ **do** $x := x - 1$; **end** $\{x = x + 1\}$

(b) $\{x = 0\}$ **while** $x < 10$ **do** $x := x + 1$; **end** $\{x = x + 1\}$

(c) $[x = 0]$ **while** $x < 10$ **do** $x := x - 1$; **end** $[x = x + 1]$

(d) $[x = 0]$ **while** $x < 10$ **do** $x := x + 1$; **end** $[x = x + 1]$

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 binding in 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.

Propose an axiom or a proof rule for 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:

$$\langle c \rangle ::= \mathbf{do} \langle be \rangle_1 \rightarrow \langle c \rangle_1 \square \langle be \rangle_2 \rightarrow \langle c \rangle_2 \square \langle be \rangle_3 \rightarrow \langle c \rangle_3 \mathbf{od}$$

where **do** and **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 ($\langle be \rangle_1 \dots \langle be \rangle_3$) and multiple loop bodies ($\langle c \rangle_1 \dots \langle c \rangle_3$). 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.

Propose an axiomatic proof rule corresponding to this command. Do not assume a certain deterministic evaluation order. Explain your proof rules briefly.

5. The following program takes a positive integer n as input, computes the factorial function $n!$, and returns the result as the value of f . Recall that $0! = 1$ and $n! = 1 * \dots * n$ for $n > 0$.

Find the loop invariant and a termination function and prove the **total** correctness of this program.

```
{n ≥ 0}
f := 1;
i := n;
while i > 1 do
    f := f * i;
    i := i - 1
end
{f = n!}
```

Clearly separate the proof into the three parts before, in, and after the while loop, and indicate the verification conditions resulting from the rule of consequence.