Univ. Grenoble Alpes
MoSIG 1

UFR IM^{2}AG
Year 2021-2022

Programming Language Semantics and Compiler Design
—
Midterm Exam of Wednesday 27 October

- **Duration**: 1h20.
- 3 sheets of A4 paper are authorized.
- Any electronic device is forbidden.
- The grading scale is indicative.
- Exercises are **independent**.
- **The care of your submission will be taken into account.**
- It is recommended to read each exercise till the end before answering. **Indicate your group number on your submission.**
- If you don't know how to answer to some question, you may assume the result and proceed with the next question.
- The maximal grade is obtained with 20 points.

```
begin
   var x := 42;
   var y := 21;
   proc p is x := x * 2
   proc q is y := y * 2;
   call p;
   begin
      proc q is x := x * 2;
      call q
   end
end
```

(a) Program for Exercise 1.

```
p := m ;
acc  := 1 ;
while acc <= n do
   p := p + m ;
   acc  := acc + 1
od
```

(b) Program for Exercise 2.

```
switch (a) {
   case n1:
      S1;
      break;
   case n2:
      S2;
   ...
   case nk:
      Sk;
      break;
   default:
      S
}
```

(c) Example of program in Exercise 3.

Figure 1: Some code snippets.

Answer of exercise 1

1. It can be obtained following the same principle as in the course.
2. The semantics of the program does not change with static scope for variables and procedures. Statement `call q` calls the same procedure in both cases.

Answer of exercise ??

1. Let us define S_0, S_1 as the following sub-programs:
   - S_0: `p := m; acc := 1`, and
   - S_1: `p := p + m; acc := acc + 1`, respectively.

   The invariant is: $I \equiv p = acc \times m \wedge acc \leq n + 1$.
   We first show that the invariant propagates through the loop body and show that the condition obtained after the loop body implies the postcondition:

$$\dfrac{\dfrac{\dfrac{\{I \wedge acc \leq n\}\ p := p + m\ \{p = (acc+1) \times m \wedge acc \leq n\} \quad \{p = (acc+1) \times m \wedge acc \leq n\}\ acc := acc + 1\ \{I\}}{\{acc \leq n \wedge I\}\ S_1\ \{I\}}}{\{I\}\ \texttt{while}\ acc \leq n\ \texttt{do}\ S_1\ \texttt{od}\ \{I \wedge n < acc\}}}{\{I\}\ \texttt{while}\ acc \leq n\ \texttt{do}\ S_1\ \texttt{od}\ \{prod = m \times (n+1)\}}$$

   We consider the initialization and essentially show that before the loop, the initialization ensures the invariant.

$$\frac{\{m = m\}\ p := m\ \{p = m\} \quad \{p = m\}\ acc := 1\ \{I\}}{\dfrac{\{m = m\}\ S_0\ \{I\}}{\{n \geq 1\}\ S_0\ \{I\}}}$$

Finally, using the rule for sequential composition, we obtain:

$$\frac{\{n \geq 1\}\ S_0\ \{I\} \quad \{I\}\ \texttt{while}\ acc \leq n\ \texttt{do}\ S_1\ \texttt{od}\ \{prod = m \times (n+1)\}}{\{n \geq 1\}\ S\ \{prod = m \times (n+1)\}}$$

Answer of exercise ??

1. An example of such program is given below:

```
switch (x + 2) {
   case 3:
      skip;
      break;
   case 2:
      x := x * y;
   default:
      x := 0
}
```

2. The grammar for `case_list` is given below:

   `case_list ::= case n: S; break_option case_list | case n: S`

   where `n` is a denotation of a natural number and `S` is a statement.
3. The grammar for `break_option` is given below:

   `break_option ::= break; | epsilon`

4. Non-terminal configurations are extended with a an integer that is the semantics of the arithmetic expression present in the `swtich` part. That is, the set of non-terminal configurations is a subset of **Stm** $\times$ **State** $\times\ \mathbb{Z}$. In terminal configurations, one can find the state as well as a Boolean for recording whether a `break` statement has been encountered. That is, the set of terminal configurations is a subset of **State** $\times\ \mathbb{B}$.
5. The rules are given below:

$$\frac{(\texttt{case\_list}, \sigma, v) \to (\sigma', b)}{(\texttt{case n} : \texttt{S}; \texttt{break\_option case\_list}, \sigma, v) \to (\sigma', b)}\ \mathcal{N}(n) \neq v$$

$$\frac{(S, \sigma) \to \sigma'}{(\texttt{case n} : \texttt{S}; \texttt{break;}\ \texttt{case\_list}, \sigma, v) \to (\sigma', \mathbf{tt})}\ \mathcal{N}(n) = v$$

$$\frac{(S, \sigma) \to \sigma' \quad (\texttt{case\_list}, \sigma', v) \to (\sigma'', b)}{(\texttt{case n} : \texttt{S}; \texttt{case\_list}, \sigma, v) \to (\sigma'', b)}\ \mathcal{N}(n) = v$$

6. The semantic rules are given below:

$$\frac{(\texttt{case\_list}, \sigma, \mathcal{A}[a]\sigma) \to (\sigma', \mathbf{tt})}{(\texttt{switch}(a)\ \{\texttt{case\_list default\_option}\}, \sigma) \to \sigma'}$$

$$\frac{(\texttt{case\_list}, \sigma, \mathcal{A}[a]\sigma) \to (\sigma', \mathbf{ff})}{(\texttt{switch}(a)\ \{\texttt{case\_list}\ \}, \sigma) \to \sigma'}$$

$$\frac{(\texttt{case\_list}, \sigma, \mathcal{A}[a]\sigma) \to (\sigma', \mathbf{ff}) \quad (S, \sigma') \to \sigma''}{(\texttt{switch}(a)\ \{\texttt{case\_list default} : S\}, \sigma) \to \sigma''}$$