Univ. Grenoble Alpes **UFR IM²AG**
MoSIG 1 **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.

Exercise 1 — Natural Operational Semantic of Proc (3 points)

Let us consider the **Proc** program depicted in Figure 1a.

1. **(2 points)** Compute the natural operational semantics of the program by either computing its derivation tree or giving the procedure and variable environments before and after each program location. Consider an initial variable environment $\rho_0 = [\,]$ and state $\sigma_0 = [\,]$. For this, consider dynamic scope for variables and procedures.
2. **(1 point)** Is the semantics of the program with static scope for variables and procedures different? If yes, indicate the final state reached by the program. Otherwise, justify why they do not differ.

Exercise 2 — Axiomatic Semantic (5 points)

1. Prove that the following Hoare triple is valid $\{n \geq 1\}\ S\ \{p = m \times (n+1)\}$, where S is the program in Figure 1b.
   For this, you may use the invariant $I \equiv p = \ldots \wedge acc \leq \ldots$, to be completed.

Exercise 3 — Natural Operational Semantics - case construct (17 points)

We are interested in language **While** and its natural operational semantics as defined in the course. We want to extend the language to account for switch-case statements. An example is given in Figure 1c. In code snippet, `a` is an arithmetic expression, `S1`,…,`Sk` are statements, and `n1`,...,`nk` are numerals (denotations of natural numbers). Moreover, the `break` statements at the end of each case are optional. The default case is also optional.

The informal semantics of this statement is to evaluate the value of arithmetic expression `a` and go through the case list. Going over the case list is done in order. The first time the numeral attached to a case statement denotes a natural number which value is the value of the arithmetic expression, then the corresponding statement is selected for execution. The statement is executed, which modifies the current state. If there is a break statement at the end of the statement, the execution of the switch-case construct is over. If there is no break statement, the execution proceeds with the next case element of the list (and a case statement can be executed later). Once there is no remaining case statements, if no break statement has been encountered, the statement associated with default case is executed.

In this exercise, we first formalize the syntax of the switch-case statements and then provide them with a natural operational semantics. For this, we start by considering the following incomplete grammar.

```
swtich-statement ::= swith (a) { case_list default_option }
case_list ::= ...
break_option ::= ...
default_option ::= default: S | epsilon
```

where `epsilon` denotes the empty sequence, `case_list` is a list of case statements, `break_option` and `default_option` are optional break and default case statements (either the corresponding statement or empty).

1. **(1 point)** Give an example of switch-case statement, with at least two cases and a default one.
2. **(1 point)** Recall that a `case_list` is a non-empty list of case statements, where each case statement is of the form `case n: S break_option`, where `case` is a keyword, `n` is a denotation of a natural number and `break_option` is a symbol referring to the rule for optional break statements. Define the grammar rule of the `case_list`, as per the grammar sketched before.
3. **(1 point)** An optional break statements consists of either the `break` keyword or is empty. Give the corresponding grammar rule by completing the following one: `break_option ::= ...`.
4. **(2 points)** We consider now the semantics. In the rules, we will distinguish the execution of the `case_list` apart from the complete switch-case statement. We start with the definition of the semantics for the `case_list`. Statements in the case list execute in an extended configuration where, in addition to the usual statement and state, there is a value $v \in \mathbb{Z}$ which corresponds to the evaluation of the arithmetic expression `a` in `switch(a)`. This value is part of the (starting) non-final configurations. Moreover, after a `case_list` executes (in final configuration), we should record with a boolean whether a break statement has been encountered (in one of the case statements) to later determine whether the default case should be executed.
   Define the sets of all possible non-terminal configurations and terminal configurations.
5. **(6 points)** In executing a statement of the form `case n: S break_option` with a value v for the arithmetic expression `a`, there are two cases: either the natural number denoted by `n` is equal to v and the statement executes, otherwise the execution proceeds with the next case statement in the list. When the case statement executes, the statement modifies the state. Depending on whether there is a `break` statement at the end, the information is recorded as a Boolean in the final configuration.
   One can thus distinguish three possible rules as indicated in the skeletons of rules below. Complete the rules.

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

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

6. **(6 points)** We now consider the rules for the switch-case statements, which reuse the rules found in the previous question. The switch-case statement executes as follows: the arithmetic expression is evaluated and the value is used to execute the list of case statements, then if no break statement has been encountered the default case executes (otherwise, it does not).
   Complete the following semantic rules providing a semantics to switch-case statements.

$$\frac{(\texttt{case\_list}, \sigma, \ldots) \to (\sigma', \texttt{tt})}{(\texttt{switch(a) \{case\_list default\_option\}}, \sigma) \to \ldots}$$

$$\frac{(\texttt{case\_list}, \sigma, \ldots) \to (\ldots, \ldots)}{(\texttt{switch(a) \{case\_list\}}, \sigma) \to \ldots} \qquad \frac{(\texttt{case\_list}, \sigma, \ldots) \to (\ldots, \ldots) \quad (\ldots, \ldots) \to \ldots}{(\texttt{switch(a) \{case\_list default : } S\}, \sigma) \to \ldots}$$