Université Joseph Fourier
MoSIG 1

SLPC

UFRIMAG
Year 2014-2015

# Exam of Wednesday 10 December 2014

**Duration : 3 hours - All documents authorized**. - Exercises are independent.

## Exercise 1

We consider the language described by the following syntax:

$$e \quad ::= \quad n \mid x \mid x := e \mid e; e \mid e \ \texttt{op} \ e \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{repeat}\ e\ \texttt{until}\ e$$

### Comments

- In this syntax, $x$ denotes an identifier ($x \in \texttt{Var}$) and $n$ is a constant integer (positive or negative ($n \in \mathbb{Z}$).
- All statements are expressions, and, consequently, returns a value.
- op denotes any arithmetical or logical operator, such as an operator in the following set : $\{+, -, *, \wedge, \vee, <, \leq, \neq, \geq, >\}$
- Statement $\texttt{repeat}\ e_1\ \texttt{until}\ e_2$ first evaluates statement $e_1$ and then, if statement $e_2$ evaluates to 0, then statement terminates and the returned value is the one of $e_1$ otherwise, we execute again the statement.
- Statement $\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3$ first evaluates statement $e_1$. If the result of this evaluation is 1 then statement $e_2$ is evaluated, otherwise $e_3$ is evaluated.

We note **Eexp** the set of such defined expressions.

### Natural Operational Semantics

Configurations are pairs $(\mathbf{Eexp} \times \mathbf{State}) \cup (\mathbb{Z} \times \mathbf{State})$.

### Questions

1. Give the operational semantics for conditional statement.
2. Give the operational semantics for sequential composition.
3. Give the operational semantics for repeat statements.

## Exercise 2

We consider the language described by the following syntax.

$$\begin{array}{lcl}
p & ::= & d : e \\
d & ::= & d_0; d \mid \epsilon \\
d_0 & ::= & \texttt{let}\ x\ :\ t = e \mid \texttt{let rec}\ x\ :\ t = e \\
t & ::= & \texttt{int} \mid t \rightarrow t \\
e & ::= & n \mid x \mid x(e) \mid\ \texttt{fun}\ x : t \rightarrow e \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid e\ \texttt{op}\ e
\end{array}$$

As in the previous exercise, an operator op is taken from the the set $\{+, -, *, \wedge, \vee, <, \leq, \neq, \geq, >\}$.

We note **Eexp'** the set of such expressions and **Decl** the set of declarations. Here, $p$ denotes a program, $d$ a sequence of declarations, $t$ a type and $e$ an expression.

### 2.A Type checking

We inspire from the typing rules of language While :

- An *environment* Env is a partial function $\texttt{Var} \rightarrow \texttt{Type}$, initially empty for the program. A program is well-typed if, by building an environment from the declarations, the expression part is correctly typed:

$$\frac{\emptyset \vdash d \mid \Gamma \qquad \Gamma \vdash e : t}{\emptyset \vdash d; e : t}$$

- The set of declarations is used to build the environment.

$$\Gamma \vdash \epsilon \mid \Gamma \qquad \frac{\Gamma \vdash d_0 \mid \Gamma' \qquad \Gamma' \vdash d \mid \Gamma''}{\Gamma \vdash d_0 \ ; \ d \mid \Gamma''}$$

#### Questions

1. Give the rules for building the type environment, by completing the two following rules.

$$\frac{\cdots}{\Gamma \vdash \texttt{let } x : t = e \mid \Gamma[\cdots]} \qquad \frac{\cdots}{\Gamma \vdash \texttt{let rec } x : t = e \mid \Gamma[\cdots]}$$

2. Give the typing rule for $x(e)$, that is the application of a function $x$ to its argument $e$.
3. Give the typing rule for $\texttt{fun } x : t \rightarrow e$.

We consider the two following programs:

| | |
|---|---|
| let f: int → int = fun $x$ : int → $x+1$ ;<br>let f: int → int = fun $x$ : int →<br>if x=0 then 1<br>else $x * f(x-1)$;<br>f(3) | let f: int → int = fun $x$ : int → $x+1$ ;<br>let rec f: int → int = fun $x$ : int →<br>if x=0 then 1<br>else $x * f(x-1)$;<br>f(3) |

#### Questions

1. Apply the previous rules to these programs.
2. Give the rule for building the type environment in the case where $d_0$ is of the form (we suppress type annotations) : $\texttt{let } x = e$.
3. (**Bonus**) Give the rule for building the type environment in the case where $d_0$ is of the form (we suppress type annotations) : $\texttt{let rec } x = e$.

### 2.B Operational semantics

The semantics domains are :

- Domain of variables: Var,
- Domain of integers: $\mathbb{Z}$.
- Domain of functions: Fun = Var × **Eexp'**
- Domain of values: Val = $\mathbb{Z} \cup$ Fun
- Domain of states: State = Var → Val

Configurations of declarations belong to (**Decl** × **State**) ∪ **State** Configurations of expressions belong to (**Eexp'**×**State**)∪Val. The set of declarations is used to build the memory. We want to define the operational semantics of this language.

## Questions

1. Give the operational semantics for the application of a function to its argument $x(e)$.
2. Give the operational semantics for $\texttt{fun}\ x : t \rightarrow e$
3. **Bonus** Give the rule to construct the memory by completing the two following rules.

$$\frac{\cdots}{(\texttt{let}\ x\ :\ t\ =\ e, \sigma) \longrightarrow \cdots} \qquad \frac{\cdots}{(\texttt{let rec}\ x\ :\ t\ =\ e, \sigma) \longrightarrow \cdots}$$

## Exercise 3

Using Hoare logic, prove that the following Hoare triple is valid: $\{b \geq 0\}\ S\ \{p = b^2\}$ where $S$ :

```
c := b ;
p := 0 ;
while !(c = 0) do
  p := p + b ;
  c := c-1 ;
od
```

## Exercise 4 : Optimization

We consider the control-flow graph in figure 1 :

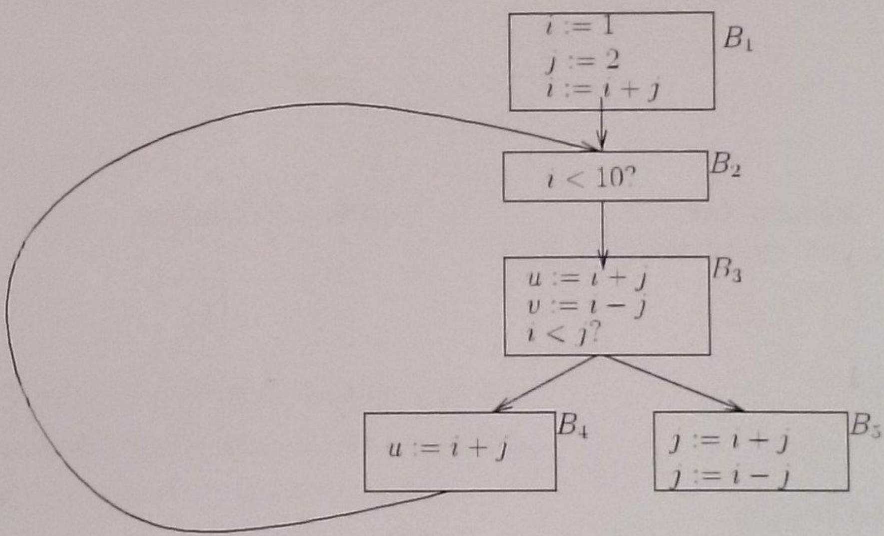


Figure 1: Initial control-flow graph

### Available expressions : questions

1. Compute the sets $Gen(b)$ and $Kill(b)$ for each basic block $b$.
2. Compute the sets $In(b)$ and $Out(b)$ for each basic block $b$.
3. Suppress redundant computations.

**In the following part, we consider the modified CFG.**

**Active Variables: questions**

1. Compute the sets $Gen(b)$ and $Kill(b)$ for each basic block $b$.
2. Compute the sets $In(b)$ and $Out(b)$ for each basic block $b$.
3. Suppress useless statements.

## Exercise 5 : Code Generation

We consider the program below

1. Draw the statck during the execution of `Q2` with only the static and dynamic links.
2. In procedure `P1`, give the sequence of code associated to `y=y1;`.
3. In procedure `P1`, give the sequence of code associated to `P2();`.
4. In procedure `P2`, give the sequence of code associated to `y= Q2 (5,&x1,F2)`.
5. In procedure `Q2`, give the sequence of code associated to `x1=*y+x+z`.
6. In procedure `Q2`, give the sequence of code associated to `return(p(z))`.
7. **Bonus** After the execution of the program, give the value associated to `x1`.

```
#include <stdio.h>
typedef int (*intprocint) ( int);

main() {
  int x1 ;
  int y1 ;
  int P1() {
    int y ;
    int F2(int x) {return (x+1);}
    int Q2 (int x, int *y,intprocint p) {
      int z ;
      z=2;
      x1 = *y+x+z;
      return(p(z));
    }
    void P2() {
      y= Q2 (5,&x1,F2);
    } /*  end P2 */
    y=y1;
    P2();
    return (y+10);
  } /* end P1 */
  x1=11;
  y1=111;
  x1=3+ P1();
}/* end main */
```