Université Joseph Fourier — **UFR IM2AG**
MoSIG1 — **Academic Year 2015 - 2016**

Programming Languages and Compiler Design

Final Exam - Wednesday December, 9 2015

Guidelines and information:

- Duration: 3 hours
- All documents from the lecture and tutorial are authorised.
- The grading scale is indicative.

Grammar for exercises 1 and 2

We consider a variant of the `While` programming language seen in the course.

$$\begin{array}{lcl}
P & ::= & \textbf{begin } D_v \ S \textbf{ end} \\
D_v & ::= & \epsilon \mid \textbf{var } x \ ; D_v \\
S & ::= & x := a \mid \textbf{skip} \mid S ; S \mid \textbf{if } b \textbf{ then } S \textbf{ else } S \textbf{ fi} \mid \textbf{begin } D_v \ S \textbf{ end} \\
a & ::= & n \mid x \mid a + a \\
b & ::= & \texttt{true} \mid a = a \mid \neg b \mid b \wedge b
\end{array}$$

S is a statement, the program P is a block, D_v is a sequence of variable declarations, a is an arithmetical expression and b a boolean expression. The term expression denotes either an arithmetical or a boolean expression.

Part I, Non-declared and non-initialised variables: typing

In this exercise, we are interested in detecting two kinds of errors, namely the notions of non-declared and non-initialised variables.

- A *non-declared* variable is a variable used in the expressions or statements of a program without appearing in the declaration of a program.
- A *non-initialised* variable is a declared variable, used in an expression before being assigned.

For instance, if we consider the sequence

`begin var x; var y; x := 0; y := x + z end`,

`x` is initialised and `y` is non-initialised, because z is non-declared. If we consider the sequence

`begin var x; var y; x := 0; y := x + y end`,

`y` is used before being initialised.

In order to detect these errors, we introduce a notion of type that differs from the one seen in the course. A **type** is a natural number that belongs to $\{0, 1, 2\}$. This set is endowed with a total order : $0 < 1 < 2$. In the typing rules of this exercise, we will be able to use a function `max` that returns the greatest of the natural numbers passed as input.

- A statement/expression has type 0 if all its variables are declared and are correctly initialised.
- A statement/expression has type 1 if all its variables are declared and at least of them is non-initialised.
- A statement/expression has type 2 if at least one of its variables appearing in it has not been declared.

The chosen order implies that the "worst" error is to used a non-declared variable.

Exercise 1

An environment is a function from variable names to types. A program is particular statement, evaluated in the empty environment, noted $\emptyset$. We consider the following judgments:

- $\Gamma \vdash S \mid (v, \Gamma')$ which means: in environment Γ, statement S has type v and produces environment Γ'.
- $\Gamma \vdash D_v \mid \Gamma_l$ which means: in environment Γ, sequence of variable declarations D_v produces (local) environment Γ_l'.
- $\Gamma \vdash e : \mathtt{t}$ which means: in environment Γ, expression e has type $\mathtt{t}$.

1. Give the typing rules for arithmetical expression, and then typing rules for boolean expressions.
2. Give the typing rules for assignments of the form `x := a` by distinguishing the case where `x` is declared from the case it is not declared.
3. Give the typing rules for block construction **begin** D_v S **end**.
4. Give the typing rules for the sequential composition `S1`; `S2` by proposing two semantics reflecting two approaches. In the first one, one evaluates `S2` whatever is the outcome of the evaluation of `S1` and report the worst error. In the second one, one evaluates `S2` only if the type of `S1` is 0.
5. Give the typing rules for the conditional construction **if** `b` **then** `S1` **else** `S2` **fi**.
6. Is the evaluation of example program influenced by the chosen approach for the sequential composition? Apply the rules to the two previous examples (by giving the inference trees).

Answer of exercise 1

1.

$$\frac{}{\Gamma \vdash n : 0} \qquad \frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash x : 2} \qquad \frac{\Gamma \vdash a1 : v1 \quad \Gamma \vdash a2 : v2}{\Gamma \vdash a1 + a2 : \max(v1, v2)}$$

$$\frac{}{\Gamma \vdash \mathtt{true} : 0} \qquad \frac{\Gamma \vdash b : v}{\Gamma \vdash \neg b : v} \qquad \frac{\Gamma \vdash b1 : v1 \quad \Gamma \vdash b2 : v2}{\Gamma \vdash b1 \wedge b2 : \max(v1, v2)}$$

2.

$$\frac{x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash x := a : (2, \Gamma[x \mapsto 2])} \qquad \frac{x \in \mathrm{dom}(\Gamma) \quad \Gamma \vdash x : v_x \quad \Gamma \vdash a : v_a}{\Gamma \vdash x := a : (\max(v_x, v_a), \Gamma[x \mapsto \max(v_x, v_a)])}$$

3.

$$\frac{\emptyset \vdash D_v \mid \Gamma_l \quad \Gamma \oplus \Gamma_l \vdash S \mid (v, \Gamma')}{\Gamma \vdash \textbf{begin } D_v \ S \textbf{ end} \mid (v, \Gamma'')} \qquad \frac{\Gamma[x \mapsto 1] \vdash D_v \mid \Gamma'}{\Gamma \vdash \textbf{var } x \,; D_v \mid \Gamma'} \qquad \frac{}{\Gamma \vdash \epsilon \mid \Gamma}$$

where $\forall x \in \mathbf{Var} : \Gamma''(x) = \begin{cases} \Gamma(x) & \text{if } x \in DV(D_v) \\ \Gamma'(x) & \text{otherwise} \end{cases}$

4. • In the first approach:

$$\frac{\Gamma \vdash S_1 \mid (v_1, \Gamma_1) \quad \Gamma_1 \vdash S_2 \mid (v_2, \Gamma_2)}{\Gamma \vdash S_1; S_2 \mid (\max(v_1, v_2), \Gamma_2)}$$

   • In the second approach:

$$\frac{\Gamma \vdash S_1 \mid (0, \Gamma_1) \quad \Gamma_1 \vdash S_2 \mid (v_2, \Gamma_2)}{\Gamma \vdash S_1; S_2 \mid (v2, \Gamma_2)} \quad \frac{\Gamma \vdash S_1 \mid (v, \Gamma_1) \quad v \neq 0}{\Gamma \vdash S_1; S_2 \mid (v, \Gamma_1)}$$

5.

$$\frac{\Gamma \vdash b : v \quad \Gamma \vdash S_1 \mid (v_1, \Gamma_1) \quad \Gamma \vdash S_2 \mid (v_2, \Gamma_2)}{\Gamma \vdash \textbf{if } \texttt{b} \textbf{ then } \texttt{S}_1 \textbf{ else } \texttt{S}_2 \textbf{ fi} \mid (\max(v, v_1, v_2), \Gamma_1 \sqcup \Gamma_2)}$$

$\forall x \in \mathbf{Var} : \Gamma_1 \sqcup \Gamma_2(x) = \begin{cases} \max(\Gamma_1(x), \Gamma_2(x)) & \text{if } x \in \mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) \\ 2 & \text{if } x \in \Big(\mathrm{dom}(\Gamma_1) \cap \overline{\mathrm{dom}(\Gamma_2)}\Big) \cup \Big(\mathrm{dom}(\Gamma_2) \cap \overline{\mathrm{dom}(\Gamma_1)}\Big) \end{cases}$

Note that, for $x \in \overline{\mathrm{dom}(\Gamma_1)} \cap \overline{\mathrm{dom}(\Gamma_2)}$, the function is not defined.

6. We apply the typing rules on the examples. For both examples, we obtain the following inference tree (parameterised by T_e^S).

$$\dfrac{\dfrac{\dfrac{[x \mapsto 1, y \mapsto 1] \vdash \epsilon \mid [x \mapsto 1, y \mapsto 1]}{[x \mapsto 1] \vdash \text{var y;} \mid [x \mapsto 1, y \mapsto 1]}}{\emptyset \vdash \text{var x; var y;} \mid [x \mapsto 1, y \mapsto 1]} \quad \dfrac{\dfrac{x \in \mathrm{dom}([x \mapsto 1, y \mapsto 1]) \quad \dfrac{}{[x \mapsto 1, y \mapsto 1] \vdash 0 : 0}}{[x \mapsto 1, y \mapsto 1] \vdash \texttt{x := 0} \mid (0, [x \mapsto 0, y \mapsto 1])} \quad T_e^S}{[x \mapsto 1, y \mapsto 1] \vdash \texttt{x := 0; S} \mid (2, [x \mapsto 0, y \mapsto 1])}}{\emptyset \vdash \texttt{begin var x; var y; x := 0; S end} \mid (2, \emptyset)}$$

Where, for the first example, T_e^S is the following inference tree:

$$\dfrac{y \in \mathrm{dom}([x \mapsto 0, y \mapsto 1]) \quad \dfrac{\dfrac{x \in \mathrm{dom}([x \mapsto 0, y \mapsto 1])}{[x \mapsto 0, y \mapsto 1] \vdash x : 0} \quad \dfrac{z \notin \mathrm{dom}([x \mapsto 1, y \mapsto 1])}{[x \mapsto 0, y \mapsto 1] \vdash z : 2}}{[x \mapsto 0, y \mapsto 1] \vdash \texttt{x} + \texttt{z} : 2}}{[x \mapsto 0, y \mapsto 1] \vdash \texttt{y} := \texttt{x} + \texttt{z} \mid (2, [x \mapsto 0, y \mapsto 1])}$$

Where, for the second example, T_e^S is the following inference tree:

$$\dfrac{y \in \mathrm{dom}([x \mapsto 0, y \mapsto 1]) \quad \dfrac{\dfrac{x \in \mathrm{dom}([x \mapsto 0, y \mapsto 1])}{[x \mapsto 0, y \mapsto 1] \vdash x : 0} \quad \dfrac{y \in \mathrm{dom}[x \mapsto 1, y \mapsto 1]}{[x \mapsto 0, y \mapsto 1] \vdash y : 1}}{[x \mapsto 0, y \mapsto 1] \vdash \texttt{x} + \texttt{y} : 1}}{[x \mapsto 0, y \mapsto 1] \vdash \texttt{y} := \texttt{x} + \texttt{y} \mid (1, [x \mapsto 0, y \mapsto 1])}$$

Part II : Non-declared and non-initialised variables - Natural Operational Semantics

In this part, we consider the error detection presented in the previous part, but from an operational semantics point of view which will report errors at runtime. More precisely, we consider the operational semantics without performing typing analysis first.

Let us recall that we have two sets of partial functions,

- the environment:
$$\mathbf{Env}_V = \mathbf{Var} \to \mathbf{Loc} \ni \rho$$
- the memory
$$\mathbf{Store} = \mathbf{Loc} \overset{part.}{\to} \mathbb{Z} \cup \{\perp\} \ni \sigma$$

We consider, as in the course, a stack of local environments (noted $\hat{\rho}$).

Hence, a non-declared variable is not in the definition domain of $\hat{\rho}$ and a non-initialised variable x is such that $\sigma(\hat{\rho}(x)) = \perp$. That is, the effect of a declaration is to i) create an address for the declared variable (as in the course) and ii) to modify memory by associating $\perp$ to the address of the declared variable.

The purpose of this exercise is to extend the natural operational semantics seen in the course to consider these errors. The analysis should follow the below constraints.

- A non-declared variable x makes the program stop, terminating in a configuration $(\texttt{stop}, x)$.
- A non-initialised variable x makes the program stop, terminating in a configuration $(\texttt{stop}, x)$.

To simplify the analysis, we do not distinguish the termination of the program due to a non-declared variable from the termination due to a non-initialised variable. Moreover, the program should stop at the first encountered error exhibiting the incriminated variable.

Exercise 2 Expressions

Configurations range over the set $\mathbf{Aexp} \times {\mathbf{Env}_V}^* \times \mathbf{Store} \cup \mathbb{Z} \cup \{\texttt{stop}\} \times \mathbf{Var}$. A final configuration is either a value or a pair $(\texttt{stop}, x)$

1. Complete the rule $(x, \hat{\rho}, \sigma) \longrightarrow \cdots$
2. Complete the rule $(a_1 + a_2, \hat{\rho}, \sigma) \longrightarrow \cdots$

Answer of exercise 2

1.
$$\frac{x \in \mathrm{dom}(\hat{\rho}) \quad \sigma(\hat{\rho}(x)) \neq \perp}{(x, \hat{\rho}, \sigma) \to \sigma(\hat{\rho}(x))} \qquad \frac{x \in \mathrm{dom}(\hat{\rho}) \quad \sigma(\hat{\rho}(x)) = \perp}{(x, \hat{\rho}, \sigma) \to (\texttt{stop}, x)} \qquad \frac{x \notin \mathrm{dom}(\hat{\rho})}{(x, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}$$

2.
$$\frac{\exists x \in \mathbf{Var} : (a_1, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}{(a_1 + a_2, \hat{\rho}, \sigma) \to (\texttt{stop}, x)} \qquad \frac{(a_1, \hat{\rho}, \sigma) \to v_1 \quad \exists x \in \mathbf{Var} : (a_2, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}{(a_1 + a_2, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}$$
$$\frac{(a_1, \hat{\rho}, \sigma) \to v_1 \quad (a_2, \hat{\rho}, \sigma) \to v_2}{(a_1 + a_2, \hat{\rho}, \sigma) \to v_1 + v_2}$$

In the following, we suppose that the semantics of Boolean expressions is extended is the same way the semantics of arithmetical expressions was extended.

Exercise 3 Statements

Configurations range over the set $\mathbf{Stm} \times {\mathbf{Env}_V}^* \times \mathbf{Store} \cup \mathbf{Store} \cup \{\texttt{stop}\} \times \mathbf{Var}$. A final configuration is either a memory σ or a pair $(\texttt{stop}, x)$

1. Give the semantics of assignments by completing the rule: $(x := a, \hat{\rho}, \sigma) \longrightarrow \cdots$

2. Give the semantics of conditional statements **if** `b` **then** `S1` **else** `S2` **fi**.

3. Give an example of program that yields a "non-initialised variable" error during type verification, but that does not yield error at runtime. **Inference and derivation trees are not requested.**

4. Give an example of program that yields a "non-declared variable" error during type verification, but that does not yield error at runtime. **Inference and derivation trees are not requested.**

Answer of exercise 3

1.

$$\frac{}{(x := a, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}\ x \notin \mathrm{dom}(\hat{\rho})$$

$$\frac{\exists y \in \mathbf{Var} : (a, \hat{\rho}, \sigma) \to (\texttt{stop}, y)}{(x := a, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}\ x \in \mathrm{dom}(\hat{\rho}) \qquad \frac{(a, \hat{\rho}, \sigma) \to v}{(x := a, \hat{\rho}, \sigma) \to \sigma[\hat{\rho}(x) \mapsto v]}\ x \in \mathrm{dom}(\hat{\rho})$$

2.

$$\frac{(b, \hat{\rho}, \sigma) \to \texttt{tt} \quad (\texttt{S1}, \hat{\rho}, \sigma) \to \sigma_1}{(\textbf{if}\ \texttt{b}\ \textbf{then}\ \texttt{S1}\ \textbf{else}\ \texttt{S2}\ \textbf{fi}, \hat{\rho}, \sigma) \to \sigma_1} \qquad \frac{(b, \hat{\rho}, \sigma) \to \texttt{ff} \quad (\texttt{S2}, \hat{\rho}, \sigma) \to \sigma_2}{(\textbf{if}\ \texttt{b}\ \textbf{then}\ \texttt{S1}\ \textbf{else}\ \texttt{S2}\ \textbf{fi}, \hat{\rho}, \sigma) \to \sigma_2}$$

$$\frac{(b, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}{(\textbf{if}\ \texttt{b}\ \textbf{then}\ \texttt{S1}\ \textbf{else}\ \texttt{S2}\ \textbf{fi}, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}$$

$$\frac{(b, \hat{\rho}, \sigma) \to \texttt{tt} \quad \exists x \in \mathbf{Var} : (\texttt{S1}, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}{(\textbf{if}\ \texttt{b}\ \textbf{then}\ \texttt{S1}\ \textbf{else}\ \texttt{S2}\ \textbf{fi}, \hat{\rho}, \sigma) \to (\texttt{stop}, x)} \quad \frac{(b, \hat{\rho}, \sigma) \to \texttt{ff} \quad \exists x \in \mathbf{Var} : (\texttt{S2}, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}{(\textbf{if}\ \texttt{b}\ \textbf{then}\ \texttt{S1}\ \textbf{else}\ \texttt{S2}\ \textbf{fi}, \hat{\rho}, \sigma) \to (\texttt{stop}, x)}$$

3. The following program yields an error "non-initialised variable" error during type verification, but does not yield any error at runtime. We informally apply the analysis rules by giving the obtained environment after each statement.

```
var x, y;         // [x |-> 1, y |-> 1]
var z, w;         // [x |-> 1, y |-> 1, z |-> 1, w |-> 1]
w := 0              // (1, [x |-> 1, y |-> 1, z |-> 1, w |-> 0 ])
if (false) then
   x := y         // (1, [x |-> 1, y |-> 1, z |-> 1, w |-> 0 ])
   z := 0         // (1, [x |-> 1, y |-> 1, z |-> 0, w |-> 0 ])
else
   skip           // (0, [x |-> 1, y |-> 1, z |-> 1, w |-> 0])
fi
skip              // (1, [x |-> 1, y |-> 1, z |-> 1, w |-> 0])
```

4. The following program yields an error "non-declared variable" error during type verification, but does not yield any error at runtime.

```
var x;               // [x |-> 1]
x := 0;              // (0, [x |-> 0])
if (false) then
   x := y            // (0, [x |-> 1, y \mapsto 1])
else
```

```
        skip                // (0, [x |-> 1, y \mapsto 1])
      fi
```

Partie III : Optimisation - Available expressions

We consider the control-flow graph in Figure 1 :

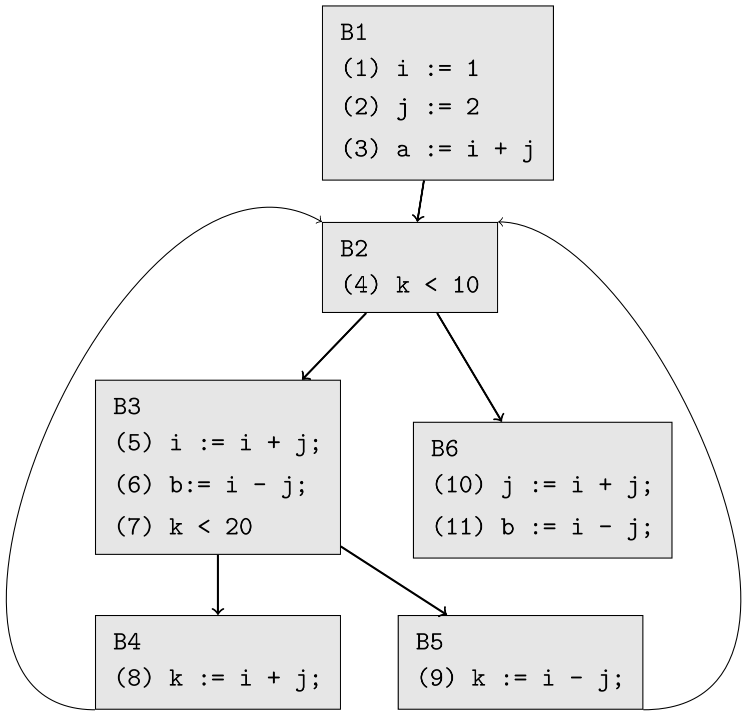

Figure 1: Initial control-flow graph

Exercise 4

We are interessed in the computation of available expressions.

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.

Answer of exercise 4

Let $e1 =$ `i + j`, $e2 =$ `i - j` be the expressions considered during analysis. We have $Out[Entry] = \emptyset$ and $Out[B_i] = \{e1, e2, e3\}, i = 0, \ldots, 4$.

1. Sets Gen and Kill are as follows:

	B1	B2	B3	B4	B5	B6
Gen	e1	$\emptyset$	e2	e1	e2	e2
Kill	Σ	$\emptyset$	Σ	$\emptyset$	$\emptyset$	Σ

2. The fix-point computation is done following the usual equations as follows:

	B1	B2	B3	B4	B5	B6
Gen	e1	$\emptyset$	e2	e1	e2	e2
Kill	Σ	$\emptyset$	Σ	$\emptyset$	$\emptyset$	Σ
Pred	$\emptyset$	B1,B4,B5	B2	B3	B3	B2
In	$\emptyset$	Σ	Σ	Σ	Σ	Σ
Out	e1	Σ	e2	Σ	Σ	e2
In	$\emptyset$	e1	Σ	e2	e2	Σ
Out	e1	e1	e2	Σ	e2	e2
In	$\emptyset$	$\emptyset$	e1	e2	e2	e1
Out	e1	$\emptyset$	e2	Σ	e2	e2
In	$\emptyset$	$\emptyset$	$\emptyset$	e2	e2	$\emptyset$
Out	e1	$\emptyset$	e2	Σ	e2	e2

3. Expression e2 is available at the entrance of B5 and is computed in B3. We can replace statement (6) by `t := i - j; b := t` and statement (9) by `k:=t`.