**Université Joseph Fourier** **Master Informatique**
**UFR IM$^2$AG**

# Sémantique des Langages de Programmation et Compilation

### Sémantique statique

## 1 Le langage `while` sans blocs

### Syntaxe abstraite

| *Métavariables* | *Ensembles* |
|---|---|
| $t$ | Type |
| $S$ | Instruction |
| $e$ | Expression |
| $x$ | Nom |
| $n$ | Constante entière |

$$
\begin{array}{rcl}
S & ::= & x := e \mid \textbf{skip} \mid S\ ;\ S \mid \\
 & & \textbf{if}\ e\ \textbf{then}\ S\ \textbf{else}\ S\ \textbf{fi} \mid \\
 & & \textbf{while}\ e\ \textbf{do}\ S\ \textbf{od} \\
e & := & n \mid \textbf{true} \mid \textbf{false} \mid x \mid e + e \mid e * e \mid \\
 & & e - e \mid e = e \mid e \leq e \mid \textbf{not}\ e \mid e\ \textbf{and}\ e \\
t & ::= & \textbf{int} \mid \textbf{bool}
\end{array}
$$

**Exercice 1.**
En utilisant les règles de sémantique statique vues en cours, prouvez que le programme suivant est correct dans l'environnement $\Gamma = [x1 \mapsto int, x2 \mapsto int, x3 \mapsto bool]$ :

```
x1 := 3
while not x3 do
  x1 := x2 + 1; x3 := x3 and true
```

**Exercice 2.**

- Donnez la sémantique statique de l'instruction `if e then S else S`.
- Complétez la syntaxe abstraite et la sémantique statique des expressions du langage en ajoutant l'expression $e$ ? $e$ : $e$ du langage `C`. L'expression $e_1$ ? $e_2$ : $e_3$ est une expression conditionnelle dont la valeur est celle de $e_2$ ou de $e_3$ selon que $e_1$ vaut vrai ou faux.

**Exercice 3.**
On étend le langage `while` avec le type **real** et les expressions avec la notation de constante réelle $r$. Complétez la grammaire abstraite et modifiez les règles de sémantique statique en considérant les deux cas suivants :

- la conversion d'un réel vers un entier est implicite, donc toujours autorisée,
- la conversion d'un réel vers un entier ne peut se faire qu'à l'aide d'un opérateur explicite de conversion **int2real** . Il faut alors fournir aussi une règle de sémantique pour cet opérateur.

## 2 Le langage `while` avec blocs

### Syntaxe abstraite

| *Métavariables* | *Ensembles* |
|---|---|
| $P$ | Programme |
| $B$ | Bloc |
| $Dv$ | Declaration |
| $t$ | Type |
| $S$ | Instruction |
| $e$ | Expression |
| $x$ | Nom |
| $n$ | Constante entière |
| $r$ | Constante réelle |

$$
\begin{array}{lcl}
P & ::= & B \\
B & ::= & \textbf{begin } Dv \ ; \ S \textbf{ end} \\
Dv & ::= & \textbf{var } x \ := \ e \ ; \ Dv \mid \epsilon \\
S & ::= & x := e \mid \textbf{ skip } \mid S_1 ; S_2 \mid \\
 & & \textbf{ if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi } \mid \\
 & & \textbf{ while } e \textbf{ do } S \textbf{ od } \mid B \\
e & := & n \mid \textbf{ true } \mid \textbf{ false } \mid r \mid x \mid e + e \mid e * e \mid \\
 & & e - e \mid e = e \mid e \leq e \mid \textbf{ not } e \mid e \textbf{ and } e \\
t & ::= & \textbf{ int } \mid \textbf{ bool } \mid \textbf{ real}
\end{array}
$$

**Exercice 4.**
Indiquez si les fragments de programmes suivants sont correctement typés ou non. On considèrera les deux sémantique de déclarations vues en cours (séquentielle et collatérale).

```
(* programme 1 *)
begin
   var x := 5 ;
   var y := x+5 ;
   x := y+4
end
```

```
(* programme 2 *)
begin
   var x := 5 ;
   begin
      var x := x < 3 ;
      x := not x
   end ;
   x := 4
```

```
(* programme 3 *)
begin
   var x := 5 ;
   var y := y + x ;
   x := y
end
```

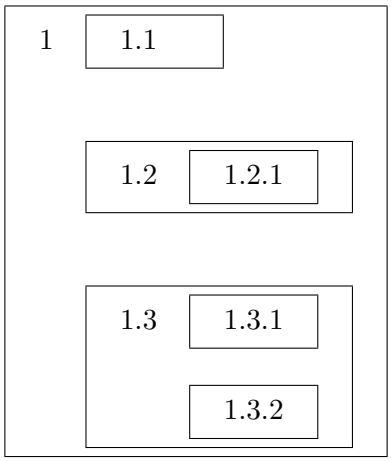
**Exercice 5.**
Calculez la valeur de l'environnement $\Gamma$ aux points d'observations 1 et 2 du programme partiel ci-dessous. Considérez les deux méthodes de définition de la sémantique des déclarations données en cours, séquentielle et collatérale.

```
begin
   var x := 2 ;
   var y := true ;
   var z = x + 2 ;
   begin
      var u := x + 6 ;
      var v := not y ;
      x := u (* point d'observation 1 *)
   end
   y := false (* point d'observation 2 *)
end
```

**Exercice 6.**
Modifiez la sémantique donnée en cours si la déclaration a la forme :

— **var** $x : t$
— **var** $x := e : t$

**Exercice 7.**
On veut associer à chaque bloc une identification unique. Pour cela, on utilise une numérotation arborescente des blocs :




A chaque bloc est ainsi associé une séquence d'entiers $ch \in \mathbb{N}^*$. Soit $n \in \mathbb{N}$ nous notons $ch.n$ la séquence constituée de $ch$ auquel on a ajouté l'entier $n$.

L'environnement est modifié de la façon suivante : à chaque couple (identificateur, bloc) on associe un type. Ainsi $\textbf{Env} = \textbf{Noms} \times \mathbb{N}^* \longrightarrow \textbf{Types}$

On pourra utiliser les prédicats suivants dans lesquels $ch \in \mathbb{N}^*$, $n, n' \in \mathbb{N}$ :

— $\Gamma, ch \vdash e : t$ qui signifie : dans l'environnement $\Gamma$, le numéro du bloc courant étant $ch$, l'expression $e$ a le type $t$
— $\Gamma, ch \vdash D_v \mid \Gamma'$ qui signifie : la déclaration $D_v$ augmente l'environnement $\Gamma$ pour former $\Gamma'$, les identificateurs sont associés au numéro $ch$
— $\Gamma, ch, n \vdash S : n'$ qui signifie : dans l'environnement $\Gamma$, le numéro du bloc courant étant $ch.n$, $S$ est correct du point de vue des types et le numéro du prochain bloc de même niveau est $ch.n'$

Définissez la nouvelle sémantique.

# 3 Le langage `while` avec procédures

### Syntaxe abstraite

| *Métavariables* | *Ensembles* |
|---|---|
| $P$ | Programme |
| $B$ | Bloc |
| $D$ | Declaration |
| $t$ | Type |
| $S$ | Instruction |
| $e$ | Expression |
| $x, p$ | Nom |
| $n$ | Constante entiere |
| $r$ | Constante réelle |

$$
\begin{array}{lcl}
P & ::= & B \\
B & ::= & \textbf{begin } Dv\ ;Dp\ ;\ S \textbf{ end} \\
Dv & ::= & \textbf{var } x\ :=\ e\ ;\ Dv \mid \epsilon \\
Dp & ::= & \textbf{proc } p \textbf{ is } S\ ;Dp \mid \epsilon \\
S & ::= & x := e \mid \textbf{ skip } \mid S\ ;\ S \mid \\
 & & \textbf{if } e \textbf{ then } S \textbf{ else } S \textbf{ fi } \mid \\
 & & \textbf{while } e \textbf{ do } S \textbf{ od } \mid B \mid \textbf{ call } x \\
e & := & n \mid \textbf{ true } \mid \textbf{ false } \mid r \mid x \mid e + e \mid e * e \mid \\
 & & e - e \mid e = e \mid e \leq e \mid \textbf{ not } e \mid e \textbf{ and } e \\
t & ::= & \textbf{int } \mid \textbf{ bool } \mid \textbf{ real}
\end{array}
$$

**Exercice 8.**
En utilisant les règles de sémantique vue en cours étudiez la construction des environnements et la correction du programme ci-dessous.

```
begin
  var x := 3
  proc p is x := x + 1
  proc q is call p
  begin
    proc p is x := x + 5
    call q
    call p
  end
  call p
end
```

**Exercice 9.**

1. Le programme suivant est-il correct selon les règles de sémantique vues en cours ? Le cas échéant comment y remédier ?
   ```
   begin
       proc p is call p ;
   call p
   end
   ```

2. On considère le programme ci-dessous :

```
begin
   proc p1 is
        call p2 ;
   proc p2 is
        call p1 ;
   call p1 ;
end
```

   (a) Montrez que ce programme est incorrect dans le cas de la liaison statique pour les procédures.

   (b) Modifiez les règles vues en cours pour que ce programme soit correct.
   **Indication :** chaque séquence de déclaration de procédure doit être analysée **deux fois**.

**Exercice 10.**
Nous ajoutons le fait qu'une procédure puisse avoir un paramètre :

$$\begin{array}{lcl} Dp & ::= & \textbf{proc}\ \ p\,(y\ :\ t)\ \ \textbf{is}\ \ S\,;Dp \mid \ldots \\ S & ::= & \ldots \mid \textbf{call}\ \ x\,(e) \end{array}$$

Donnez les nouvelles règles de sémantique.

**Exercice 11.**
En utilisant les règles de sémantique précédentes montrez que le programme suivant est correct :

```
begin
  var x := 3
  proc p (u : int) is x := u + 1
  begin
    var x := true
    proc p (u : bool) is not u
    call p (x)
  end
  call p (x)
end
```

**Exercice 12.**
On ajoute au langage la possibilité de définir des fonctions (procédure donnant un résultat) et ainsi d'avoir des appels de fonctions dans les expressions. Modifiez la grammaire abstraite et complétez les règles de sémantique.

# 4 Un langage fonctionnel

## Syntaxe abstraite

| *Métavariables* | *Ensembles* |
|---|---|
| $P$ | Programme |
| $\tau$ | Type |
| $e$ | Expression |
| $x$ | Nom |
| $op$ | Operateur |

$$\tau ::= \textbf{bool} \mid \textbf{int} \mid \textbf{real} \mid \tau \longrightarrow \tau$$
$$e ::= n \mid \textbf{true} \mid x \mid \textbf{fun}\ x \cdot e \mid (e\ e) \mid \textbf{let}\ x = e\ \textbf{in}\ e$$

## Sémantique

$$\frac{}{\Gamma \vdash \textbf{n} : \textbf{int}} \qquad \frac{}{\Gamma \vdash \textbf{true} : \textbf{bool}} \qquad \frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \textbf{fun}\ x \cdot e : \tau_1 \mapsto \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \mapsto \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1 e_2) : \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma[x \mapsto \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau_2}$$

**Exercice 13.**
On considère l'environnement donné par le tableau suivant :

| Nom | Type |
|---|---|
| a1 | `int` |
| a2 | `int` $\mapsto$ (`int` $\mapsto$ `int`) |
| a3 | (`int` $\mapsto$ `int`)$\mapsto$ `int` |
| a4 | `int` $\mapsto$ `int` |
| a5 | (`int` $\mapsto$ `int`)$\mapsto$ (`int` $\mapsto$ `int`) |
| a6 | `int` |

Donnez le type, lorsque c'est possible, des expressions suivantes :
- (a1 a2)
- (a2 a1)
- (a3 a4)
- (a5 a4)
- ((a5 a4) a1)
- ((a5 a4) a3)

**Exercice 14.**
Appliquez les règles de typage sur les expressions suivantes :
- ((fun $x$ · $x$) 3)
- ((fun $x$ · $x$) `true`)
- let $x$ = 1 in ((fun $y$ · $y$) $x$).
- ((fun $x$ · ($x$ $x$)) `true`)