

1 Introduction

Nous rappelons les règles de typages vues en cours pour un langage impératif (section 2) et un langage fonctionnel (section 3). Ces règles sont fondées sur la construction d'un environnement Γ qui décrit une fonction des noms vers les types. Cette fonction représente une abstraction de la table des symboles. Durant sa construction, on vérifie qu'un nom n'est déclaré qu'une seule fois. C'est la partie identification des noms. La section 4 donne les exercices.

2 Un langage impératif \mathcal{L}

2.1 Grammaire abstraite

$$\begin{aligned} S & ::= x := a \mid \mathbf{skip} \mid S_1; S_2 \mid \mathbf{if } b \mathbf{ then } S_1 \mathbf{ else } S_2 \mid \mathbf{while } b \mathbf{ do } S \\ a & ::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \\ b & ::= \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \end{aligned}$$

2.2 Sémantique statique standard de \mathcal{L}

Types : $\mathbf{Int}, \mathbf{Bool} \ni \tau$

Environnement : Un environnement est une fonction des noms de variables vers les types.

$$\mathbf{Env} = \mathbf{Noms} \longrightarrow \mathbf{Types} \ni \Gamma$$

On considère les prédicats suivants :

- $\Gamma \vdash S$ qui signifie dans l'environnement Γ la commande S est bien typée.
- $\Gamma \vdash a : \mathbf{Int}$ qui signifie dans l'environnement Γ l'expression a a le type \mathbf{Int} .
- $\Gamma \vdash b : \mathbf{Bool}$ qui signifie dans l'environnement Γ l'expression b a le type \mathbf{Bool} .

2.3 Les entiers naturels

$$\frac{}{\Gamma \vdash n : \mathbf{Int}} \quad \frac{}{\Gamma \vdash x : \mathbf{Int}} \text{ (si } \Gamma(x) = \mathbf{Int}) \quad \frac{\Gamma \vdash a_1 : \mathbf{Int} \quad \Gamma \vdash a_2 : \mathbf{Int}}{\Gamma \vdash a_1 + a_2 : \mathbf{Int}}$$

2.4 Les booléens

$$\frac{}{\Gamma \vdash \mathbf{true} : \mathbf{Bool}} \quad \frac{\Gamma \vdash b_1 : \mathbf{Bool} \quad \Gamma \vdash b_2 : \mathbf{Bool}}{\Gamma \vdash b_1 \wedge b_2 : \mathbf{Bool}} \quad \frac{\Gamma \vdash a_1 : \mathbf{Int} \quad \Gamma \vdash a_2 : \mathbf{Int}}{\Gamma \vdash a_1 = a_2 : \mathbf{Bool}}$$

2.5 Les commandes

$$\frac{\Gamma \vdash a : \mathbf{Int} \quad \Gamma \vdash x : \mathbf{Int}}{\Gamma \vdash x := a} \quad \frac{}{\Gamma \vdash \mathbf{skip}} \quad \frac{\Gamma \vdash S_1 \quad \Gamma \vdash S_2}{\Gamma \vdash S_1; S_2} \quad \frac{\Gamma \vdash b : \mathbf{Bool} \quad \Gamma \vdash S}{\Gamma \vdash \mathbf{while } b \mathbf{ do } S}$$

2.6 Extension : Bloc

On rajoute une règle pour les commandes :

$$S ::= \dots \mid \mathbf{begin } D_V ; S \mathbf{ end}$$

et pour les déclarations :

$$D_V ::= \mathbf{var } x := a : \mathbf{Int} ; D_V \mid \epsilon$$

2.6.1 Construction de l'environnement (déclarations)

On évalue les déclarations dans l'environnement global Γ , on construit un environnement local Γ_l . On introduit une nouvelle règle pour les déclarations :

$\Gamma \vdash D_V \mid \Gamma_l$ qui signifie les déclarations D_V modifient l'environnement Γ en Γ_l .

Rappel :

- $DV(D_v)$ dénote l'ensemble des variables déclarées dans D_v .
- $\Gamma[y \mapsto \tau]$ dénote l'environnement Γ' tel que :

$$\Gamma'(x) = \begin{cases} \Gamma(x) & \text{si } x \neq y \\ \tau & \text{sinon} \end{cases}$$

$$\frac{}{\Gamma \vdash \epsilon \mid \Gamma} \quad \frac{\Gamma \vdash a : \mathbf{Int} \quad \Gamma[x \mapsto \mathbf{Int}] \vdash D_V \mid \Gamma_l \quad x \notin DV(D_V)}{\Gamma \vdash \mathbf{var } x := a : \mathbf{Int} ; D_V \mid \Gamma_l}$$

L'environnement local Γ_l est obtenu à partir de l'environnement global Γ en modifiant Γ pour les variables locales.

2.6.2 Bloc

On évalue la partie commande du bloc dans l'environnement Γ modifié par Γ_l .

$$\frac{\Gamma \vdash D_V \mid \Gamma_l \quad \Gamma_l \vdash S}{\Gamma \vdash \mathbf{begin } D_V ; S \mathbf{end}}$$

2.7 Extension : procédures

On modifie la syntaxe de telle sorte qu'il n'y ait plus de distinction entre expressions booléennes et arithmétiques.

$$\begin{aligned} e &::= n \mid r \mid \mathbf{true} \mid \mathbf{false} \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \mid e_1 = e_2 \mid e_1 \leq e_2 \mid \neg e \mid e_1 \wedge e_2 \\ S &::= \dots \mid \mathbf{begin } D ; S \mathbf{end} \mid \mathbf{call } x \mid \mathbf{call } x(e) \\ D &::= \mathbf{var } x := e : b_t ; D \mid \mathbf{proc } x \mathbf{is } S ; D \mid \mathbf{proc } x(y : b_t) \mathbf{is } S ; D \mid \epsilon \\ b_t &::= \mathbf{Int} \mid \mathbf{Real} \mid \mathbf{Bool} \end{aligned}$$

On redéfinit l'environnement de la manière suivante :

$$\mathbf{Env} = \mathbf{Noms} \longrightarrow \mathbf{Types} \ni \Gamma$$

$$\mathbf{Types} ::= \mathbf{Int} \mid \mathbf{Real} \mid \mathbf{Bool} \mid \mathbf{Proc} \mid \{\mathbf{Proc}\} \times \mathbf{Types}$$

On introduit un nouveau prédicat $\Gamma \vdash D_V$ qui signifie que dans l'environnement Γ , la déclaration D_V est bien typée.

Construction de l'environnement	Vérification des déclarations
$\Gamma \vdash \epsilon \mid \Gamma$	$\Gamma \vdash \epsilon$
$\frac{\Gamma[x \mapsto b_t] \vdash D \mid \Gamma_l \quad x \notin DV(D)}{\Gamma \vdash \mathbf{var } x := e : b_t ; D \mid \Gamma_l}$	$\frac{\Gamma \vdash e : b_t \quad \Gamma \vdash D}{\Gamma \vdash \mathbf{var } x := e : b_t ; D}$
$\frac{\Gamma[x \mapsto \mathbf{Proc}] \vdash D \mid \Gamma_l \quad x \notin DV(D)}{\Gamma \vdash \mathbf{proc } x \mathbf{is } S ; D \mid \Gamma_l}$	$\frac{\Gamma \vdash S \quad \Gamma \vdash D}{\Gamma \vdash \mathbf{proc } x \mathbf{is } S ; D}$
$\frac{\Gamma[x \mapsto (\mathbf{Proc}, b_t)] \vdash D \mid \Gamma_l \quad x \notin DV(D)}{\Gamma \vdash \mathbf{proc } x(y : b_t) \mathbf{is } S ; D \mid \Gamma_l}$	$\frac{\Gamma[y \mapsto b_t] \vdash S \quad \Gamma \vdash D}{\Gamma \vdash \mathbf{proc } x(y : b_t) \mathbf{is } S ; D}$
	$\frac{\Gamma \vdash D_V \mid \Gamma_l \quad \Gamma_l \vdash D_V \quad \Gamma_l \vdash S}{\Gamma \vdash \mathbf{begin } D_V ; S \mathbf{end}}$

3 Un langage fonctionnel

Type $\tau ::= \text{Bool} \mid \text{Int} \mid \text{Real} \mid \tau \longrightarrow \tau$.

3.1 Syntaxe

$$e ::= c \mid \mathbf{op} \mid x \mid \mathbf{fun} \ x \cdot e \mid (e \ e) \mid (e, e) \mid \mathbf{let} \ x = e \ \mathbf{in} \ e$$

3.2 Sémantique

Les règles de typage sont les suivantes : **Types** $\exists \tau ::= \mathbb{T} \mid \tau \longrightarrow \tau \mid \tau \times \tau$

$$\overline{\Gamma \vdash c : \mathbf{TC}(c)}$$
$$\overline{\Gamma \vdash \mathbf{op} : \mathbf{TC}(op)}$$
$$\overline{\Gamma \vdash x : \Gamma(x)}$$
$$\frac{\Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \mathbf{fun} \ x \cdot e : \tau_1 \mapsto \tau_2}$$
$$\frac{\Gamma \vdash e_1 : \tau_1 \mapsto \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1 e_2) : \tau_2}$$
$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma[x \mapsto \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2 : \tau_2}$$
$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

4 Exercices

Exercice 1

Modifier les règles de sémantique en prenant en compte les règles de conversion implicites entre entiers et réels, à savoir que si un opérateur binaire a un opérande entier et un opérande réel, alors l'opérande entier est converti en réel. Discuter du fait de disposer dans le langage d'un opérateur explicite de conversion de type.

Exercice 2

Donner un exemple de programme sous forme d'arbre et calculer les informations de type associées. Par exemple, considérons l'arbre $(+, x, (+, y, z))$ en utilisant les environnements suivants

- $\Gamma = [x \mapsto \text{Int}, y \mapsto \text{Int}, z \mapsto \text{Int}]$
- $\Gamma = [x \mapsto \text{Réel}, y \mapsto \text{Int}, z \mapsto \text{Int}]$
- $\Gamma = [x \mapsto \text{Réel}, y \mapsto \text{Réel}, z \mapsto \text{Réel}]$
- $\Gamma = [x \mapsto \text{Int}, y \mapsto \text{Réel}, z \mapsto \text{Réel}]$
- $\Gamma = [x \mapsto \text{Int}, y \mapsto \text{Int}, z \mapsto \text{Réel}]$

Exercice 3

On ajoute dans au langage **while** avec blocs et procédures la possibilité d'avoir des appels de procédures dans les expressions.

$$\begin{aligned} S & ::= x := e \mid \mathbf{skip} \mid S_1; S_2 \mid \\ & \quad \mathbf{if} \ e \ \mathbf{then} \ S_1 \ \mathbf{else} \ S_2 \\ & \quad \mathbf{while} \ e \ \mathbf{do} \ S \mid \mathbf{begin} \ D; S \ \mathbf{end} \mid \mathbf{call} \ x \mid \mathbf{call} \ x(e) \\ D & ::= d \mid D; D \mid \epsilon \\ d & ::= \mathbf{var} \ x := e : \tau \mid \mathbf{proc} \ x \ \mathbf{is} \ S : \tau \mid \mathbf{proc} \ x(y : \tau) \ \mathbf{is} \ S : \tau \\ \tau & ::= \mathbf{Void} \mid \mathbf{Int} \mid \mathbf{Real} \mid \mathbf{Bool} \\ e & ::= n \mid r \mid \mathbf{true} \mid \mathbf{false} \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \mid e_1 = e_2 \mid e_1 \leq e_2 \mid \neg e \mid e_1 \wedge e_2 \mid \mathbf{call} \ x \mid \mathbf{call} \ x(e) \end{aligned}$$

Modifier les règles de typage.

Exercice 4

On considère le langage fonctionnel du paragraphe 3. 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

Donner le type, lorsque c'est possible, des expressions suivantes :

- (a1 a2)
- (a2 a1)
- (a3 a4)
- (a5 a4)
- ((a5 a4) a1)
- ((a5 a4) a3)

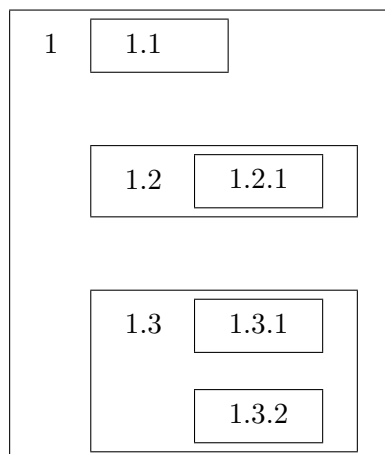
Exercice 5

- Donner un algorithme de typage d'une expression.
- Appliquer les règles de typages sur
 - ((fun x · x) 3)

- ((fun x · x) true)
- let x = 1 in ((fun y : Int · y) x).
- let x = fun x · x in let y = (f 1) in let z = (= (1, 2)) in (f z)
- let f = fun x · x + 3 in let g = fun x · x in (+((f(g 2)), (g 3)))
- ((fun x · (x x)) true)

Exercice 6

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



L'environnement est modifié ainsi : à chaque identificateur, à chaque bloc, on associe un type :

$$\mathbf{Env} = \mathbf{Noms} \times \mathbb{N}^* \longrightarrow \mathbf{Types} \ni \Gamma$$

Une configuration pour les commandes sera de la forme :

$$(\mathbf{Env} \times \mathbf{Stm} \times \mathbb{N}^* \times \mathbb{N}) \cup \mathbb{N}$$

Définir la nouvelle sémantique sur la grammaire suivante :

$$\begin{aligned}
 P &::= B \\
 B &::= | \text{begin } D ; S \text{ end} \\
 e &::= n \mid r \mid \text{true} \mid \text{false} \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \mid e_1 = e_2 \mid e_1 \leq e_2 \mid \neg e \mid e_1 \wedge e_2 \\
 S &::= x := e \mid S ; S \mid \text{skip} \mid \text{if } e \text{ then } S \text{ else } S \mid \text{while } e \text{ do } S \text{ od} \mid \text{begin } D ; S \text{ end} \\
 D &::= \text{var } x := e : b_t ; D \mid \epsilon \\
 b_t &::= \text{Int} \mid \text{Real} \mid \text{Bool}
 \end{aligned}$$