

Génération de code pour un langage impératif d'ordre supérieur

1 Introduction

Dans ce qui suit, nous considérons une extension du langage `while` par ajout de procédures et fonctions. Pour simplifier la présentation, les blocs ne pourront apparaître que au niveau de la définition d'une procédure ou fonction. De manière générale, un langage est qualifié d'ordre supérieur lorsque le passage en paramètre de procédures est licite. Nous ne traitons pas des objets alloués dynamiquement comme les tableaux à bornes dynamiques ou les pointeurs.

La sémantique naturelle du langage `while` a été étudiée précédemment. Lorsque nous introduisons des blocs et des procédures, il se pose le problème de liaison des indentificateurs libres, ceux qui ne sont pas définis localement. Nous avons vu que, pour accéder aux objets non locaux nous avons le choix entre une liaison statique ou dynamique. Dans ce qui suit, nous choisissons une liaison statique pour l'accès aux objets non locaux. Par ailleurs, nous utilisons une sémantique dite standard, fondée sur une pile d'exécution pour la gestion des procédures récursives. Informellement, la sémantique est la suivante : avant chaque activation d'une procédure, on empile l'environnement local qui sera dépilé en fin d'activation.

Pour réaliser cette gestion en pile, nous utilisons deux registres `FP` (*Frame Pointer*), base de l'environnement local et `SP` (*Stack Pointer*) pointeur de pile.

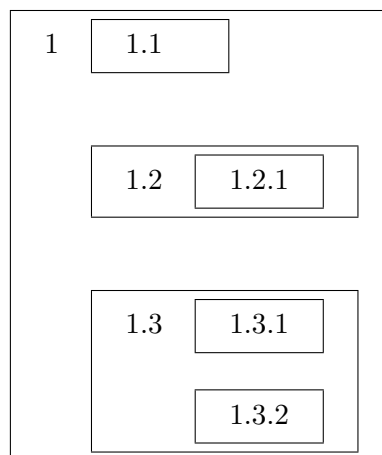
Nous rappelons brièvement comment est faite l'indentification des noms, puis nous donnons des indications sur la génération de code pour une procédure.

2 Identification des noms

On rappelle l'exercice 7, de la feuille consacrée à la vérification des types.

2.1 Syntaxe

On considère le langage `while` avec chaînage statique. 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, ainsi qu'une information sur la nature de l'identificateur indiquant si c'est une variable,

un paramètre ou un procédure. On note \mathbb{N}^* les séquences d'entiers séparés par un point : par exemple $1 \cdot 2 \cdot 3 \in \mathbb{N}^*$.

$\mathbf{Env} = \mathbf{Noms} \times \mathbb{N}^* \longrightarrow \{\mathbf{proc}, \mathbf{var}, \mathbf{par}\} \times \mathbf{Types}$

On considère la syntaxe suivante :

B	$::=$	$\mathbf{begin} D_V; D_P; S \mathbf{end}$	(* bloc *)
D_V	$::=$	$\mathbf{var} x : t; D_V \mid \epsilon$	(*déclarations de variables *)
D_P	$::=$	$\mathbf{proc} p(pf) \mathbf{is} C : t; D_P \mid \epsilon$	(*déclarations de procédures *)
pf	$::=$	$x : t, pf \mid \epsilon$	(*paramètres formels *)
S	$::=$	$x := e \mid \mathbf{skip} \mid S_1; S_2 \mid$ $\mathbf{if} e \mathbf{then} S_1 \mathbf{else} S_2$ $\mathbf{while} e \mathbf{do} S \mid \mathbf{call} p(pe)$	(*commandes *)
pe	$::=$	$e, pe \mid \epsilon$	(*paramètres effectifs *)
C	$::=$	$B \mid S$	(*corps de procédure*)

2.2 Construction de l'environnement

2.2.1 Construction avec les variables

Les configurations sont de la forme $\mathbf{Env} \times D_V \times \mathbb{N}^* \cup \mathbf{Env}$. Ce qui signifie que dans l'environnement \mathbf{Env} avec l'élément $ch \in \mathbb{N}^*$ on construit un nouvel environnement par l'intermédiaire des déclarations de variables D_V . Les règles sont de la forme

$$(\mathbf{Env}, \epsilon, ch) \longrightarrow \mathbf{Env}$$

$$\frac{(\mathbf{Env}[(x, ch) \mapsto (\mathbf{var}, t)], D_V, ch) \longrightarrow \mathbf{Env}'}{(\mathbf{Env}, \mathbf{var} x : t; D_V, ch) \longrightarrow \mathbf{Env}'} (x \notin \mathbf{Def}(D_V) \wedge (x, ch) \notin \mathbf{Dom}(\mathbf{Env}))$$

2.2.2 Construction avec les procédures

Les configurations sont de la forme $\mathbf{Env} \times D_P \times \mathbb{N}^* \cup \mathbf{Env}$. Ce qui signifie que dans l'environnement \mathbf{Env} avec l'élément $ch \in \mathbb{N}^*$ on construit un nouvel environnement par l'intermédiaire des déclarations de procédures D_P .

Les règles sont de la forme

$$(\mathbf{Env}, \epsilon, ch) \longrightarrow \mathbf{Env}$$

$$\frac{(\mathbf{Env}[(x, ch) \mapsto (\mathbf{proc}, \tau \longrightarrow t)], D_P, ch) \longrightarrow \mathbf{Env}'}{(\mathbf{Env}, \mathbf{proc} x(pf) \mathbf{is} C : t; D_P, ch) \longrightarrow \mathbf{Env}'} \left(\begin{array}{l} x \notin \mathbf{Def}(D_P) \wedge (x, ch) \notin \mathbf{Dom}(\mathbf{Env}) \wedge \\ \tau = \mathcal{T}(pf) \end{array} \right)$$

La fonction \mathcal{T} est définie de la manière suivante :

$\mathcal{T}(\epsilon)$	$=$	\mathbf{void} si la liste de paramètres formels est vide
$\mathcal{T}(x : t)$	$=$	t
$\mathcal{T}(x : t; pf)$	$=$	$t \times \mathcal{T}(pf)t$ si la liste de paramètres formels est non vide

2.2.3 Vérification des déclarations de procédures

Les configurations sont de la forme $\mathbf{Env} \times D_P \times \mathbb{N}^* \times \mathbb{N} \cup \{\top, \perp\}$. Ce qui signifie que dans l'environnement \mathbf{Env} avec l'élément $ch \in \mathbb{N}^*$ on construit un nouvel environnement par l'intermédiaire des déclarations de procédures D_P . L'entier cpt servira à discriminer les blocs d'un

même niveau (dans la figure de la page 1, on distingue ainsi $1 \cdot 1$ de $1 \cdot 2$; de même, $1 \cdot 3 \cdot 1$ de $1 \cdot 3 \cdot 2$).

Les règles sont de la forme

$$(\text{Env}, \epsilon, ch, cpt) \longrightarrow \top$$

$$\frac{(\text{Env}, pf, ch.cpt) \longrightarrow \text{Env}_1 \quad (\text{Env}_1, C, ch.cpt, cpt) \longrightarrow (\top, cpt') \quad (\text{Env}, D_P, ch, cpt') \longrightarrow (\top, cpt'')}{(\text{Env}, \text{proc } x(pf) \text{ is } C : t; D_P, ch, cpt) \longrightarrow (\top, cpt'')}$$

2.2.4 Bloc

$$\frac{(\text{Env}, D_V, ch) \longrightarrow \text{Env}_1 \quad (\text{Env}_1, D_P, ch) \longrightarrow \text{Env}_2 \quad (\text{Env}_2, D_P, ch, 1) \longrightarrow (\top, cpt) \quad (\text{Env}_2, S, ch, cpt) \longrightarrow (\top, cpt')}{(\text{Env}, \text{begin } D_P; D_V; S \text{ end}, ch, cpt) \longrightarrow (\top, cpt + 1)}$$

3 Environnement d'exécution d'une procédure

Nous décrivons brièvement comment est organisée la mémoire lors de l'exécution d'un programme.

3.1 Représentation de l'environnement

3.1.1 Partie statique et partie dynamique

- La partie statique est la partie dont la taille est connue à la compilation. Ce sont par exemple les variables de type simple, les tableaux de taille fixe, etc.
- La partie dynamique est la partie dont la taille est connue à l'exécution. Comme nous l'avons dit, nous n'abordons pas ce problème dans ce document.

3.1.2 Représentation de la partie statique

Cette zone contient la zone des variables, la zone des paramètres et les données de liaison, qui sont les informations nécessaires à la gestion de la communication entre environnements.

Le registre FP, base de l'environnement local, est utilisé pour mettre à jour le sommet de pile au moment de l'activation et de la sortie de la procédure. C'est le *chaînage dynamique*. Autrement dit, FP contient l'adresse d'un emplacement mémoire dont le contenu est l'adresse de la base de l'environnement d'appel.

le **chaînage statique** sert à retrouver l'environnement de définition de l'objet.

Afin de résoudre les problèmes liés à la portée lexicale des identificateurs, nous utilisons l'information associée à l'environnement qui détermine les noms de manière unique (on aurait pu identifier chaque nom en le préfixant par la concaténation de ses procédures englobantes).

Au moment de la génération de code, on ajoute à l'environnement **Env** pour chaque variable et chaque paramètre un déplacement par rapport à **FP**. Ce déplacement peut être calculé au moment de l'identification des noms, par une simple modification de la sémantique présentée à la section précédente. On a donc, pour chaque identificateur **x** et chaque élément **ch**, **Env(x, ch) = (mode, type, int)** où le mode est soit var, par ou proc et le dernier entier le déplacement par rapport à FP (ce champ est significatif dans le cas où le mode est soit var soit par). Le type sert à calculer le déplacement.

Remarque Nous aurions pu choisir, au lieu de $\mathbf{Env} = \mathbf{Noms} \times \mathbb{N}^* \longrightarrow \{\mathbf{proc}, \mathbf{var}, \mathbf{par}\} \times \mathbf{Types}$
 $\mathbf{Env} = \mathbf{Noms} \longrightarrow 2^{\mathbb{N}^* \times \{\mathbf{proc}, \mathbf{var}, \mathbf{par}\} \times \mathbf{Types}}$

Le premier cas correspond à la notion de table locale en compilation tandis que le deuxième cas, à la notion de table globale. Ainsi, si on considère un couple $(x, ch) \in \text{Dom}(\mathbf{Env})$, nous obtenons une identification unique de x .

Identificateur Dans la suite, un identificateur est donné par un couple (x, ch) , c'est-à-dire (nom, environnement de définition). On peut définir un ordre sur les identificateurs de la manière suivante :

$$(x, \leq y \text{ ssi } (x, x_1 \cdots x_j) \leq (y, y_1 \cdots y_n) \text{ ssi } j \leq n \wedge \forall i \leq j \cdot x_i = y_i$$

On peut alors définir le plus grand symbole inférieur à P et à Q.

3.2 Communication entre environnements

Soit p une procédure, identifiée par (p, ch_p) et x un nom, identifié par (x, ch_x) . On dit que x est *accessible* à p si

- $\exists n \in \mathbb{N} \ ch_x = ch_p \cdot n$ (variable locale), ou
- $(x, ch_x) \leq (p, ch_p)$ (variable globale)

3.2.1 Les paramètres

Les paramètres peuvent être dans une zone mémoire fixe, dans les registres ou dans la pile. Nous utilisons la pile pour ranger les paramètres.

3.3 Accès à un objet local

Soit p la procédure de référence.

x est une **variable locale**, alors son adresse est déterminée par FP-depl où, depl est le déplacement que l'on trouve dans la table des symboles, $\mathbf{Env}(x, ch_x) = (\mathbf{var}, \mathbf{t}, \mathbf{depl})$.

x est un **paramètre de la procédure**, alors son adresse est déterminée par FP+depl où, depl est le déplacement que l'on trouve dans la table des symboles, $\mathbf{Env}(x, ch_x) = (\mathbf{par}, \mathbf{t}, \mathbf{depl})$.

x est une **procédure locale**, alors son adresse est l'étiquette x et son chaînage statique est FP.

3.4 Accès à un objet global

Soit p la procédure de référence.

- L'accès à l'environnement de définition se fait en utilisant $\mathbf{Env}(x, ch_x)$ et $\mathbf{Env}(p, ch_p)$.
- L'accès à l'élément se fait ensuite comme dans le cas précédant (objet local).

3.5 Convention pour la pile

On suppose que les variables et les paramètres sont représentés sur 4 octets. Un paramètre procédural sera composé de 4 octets pour l'adresse du code de la procédure et de 4 octets pour le lien statique.

...	0	...	0
var _n	←SP, FP- 4*n	var _n	←SP, FP- 4*n
...		...	
var ₁	←FP-4	var ₁	←FP-4
Châinage dynamique	←FP	Châinage dynamique	←FP
Adresse de retour	←FP+4	Adresse de retour	←FP+4
Châinage statique	←FP+8	Châinage statique	←FP+8
Param _n	←FP+12	Résultat	←FP+12
...		Param _n	←FP+16
Param ₁	←FP+8+4*n	...	
		Param ₁	←FP+12+4*n

Dans la pile de gauche, nous avons une configuration d'une procédure avec n paramètres sans résultat. Dans la pile de droite, nous avons une procédure avec n paramètres et un résultat tenant sur 4 octets.

3.6 Recherche de l'environnement de définition

- Soit R un registre disponible. On a le code suivante :

$$\left. \begin{array}{l} \text{LD } R, [FP + 8] \\ \text{LD } R, [R + 8] \\ \dots \\ \text{LD } R, [R + 8] \end{array} \right\} \text{réalisé } |ch_p| - |ch_x| - 1 \text{ fois}$$

où, pour un élément de \mathbb{N} , la fonction $|\cdot|$ est définie de la manière suivante :

$$\begin{aligned} |\epsilon| &= 0 \\ |ch \cdot n| &= |ch| + 1 \end{aligned}$$

4 Génération de code pour une procédure

Pour chaque procédure, on génère

- un prologue,
- le corps de la procédure, en utilisant les fonctions déjà définies pour générer du code pour les expressions et les instructions,
- un épilogue.

4.1 Appel d'une procédure

- Dans l'appelant, on évalue et on empile chaque paramètre : 4 octets pour un entier ou un booléen et 8 octets pour une procédure (4 pour l'adresse et 4 pour le chaînage statique),
- dans l'appelant, si la procédure à un résultat, on réserve dans la pile la taille nécessaire pour ce résultat,
- dans l'appelant, on empile le chaînage statique de la procédure qui va être appelée,
- dans l'appelé, on exécute un prologue qui empile le contenu de FP et met à jour le pointeur de pile avec la taille des données locales.

4.2 Retour d'une procédure

- On restaure le contexte de l'appelant, en retrouvant sa base grâce à FP, le chaînage statique (épilogue).
- Dans l'appelant, on récupère la place utilisée pour les paramètres, le résultat et le lien statique.

5 Exemple

```
begin
  var x : int;
  var y : int;
  proc p() is x := x* 2;
  proc q() is call p ();
  proc r() is
    begin
      var x : int;
      proc p() is x := x +1;
      x := 5;
      call q();
      y :=x
    end
  x := 0;
  call r();
end
```

5.0.1 Construction de l'environnement

Procédure	Environnement défini
main	Env (x,1) = (var, int, 4) Env (y,1) = (var,int,8) Env(p,1) = (proc, void \rightarrow void) Env(q,1) = (proc, void \rightarrow void) Env(r,1) = (proc, void \rightarrow void)
main.p	\emptyset
main.q	\emptyset
main.r	Env (x,1.1)= var, int, 4) Env(p,1.1) = (proc, void \rightarrow void)

5.0.2 Génération de code

Pour le programme principal :

```
prologue(8)
SUB R0, R0, R0  -- on le registre R0 contiendra 0
ST R0, [FP-4]  -- x := 0

ADD SP, SP, -4 -- on empile FP pour l'appel de r
ST FP, [SP]
CALL main.r
ADD SP, SP, +4  -- on récupère la place sur la pile
epilogue
RET
```

Pour la procédure main.r

```
prologue(4)
ADD R2, R0, 5   -- x := 5
ST R2, [FP-4]

LD R1, [FP+8]   -- on empile le lien statique de q
ADD SP, SP, -4
ST R1, [SP]
CALL main.q
ADD SP, SP, +4

LD R2, [FP-4]   -- y := x
ST R2, [R1-8]
epilogue
RET
```

Pour la procédure main.q

```
prologue(0)
LD R1, [FP+8]   -- on empile le lien statique de p
ADD SP, SP, -4
ST R1, [SP]
CALL main.p
ADD SP, SP, +4
epilogue
RET
```

Pour la procédure main.p

```
prologue(0)
LD R1, [FP+8]   -- on récupère le lien statique de x
LD R2, [R1-4]
```

```
ADD R3,R2,R2
ST R3, [R1-4]
epilogue
RET
```