Université Bordeaux 1
Licence, module Compilation, 2016/2017

PROJET [1]

Le projet consiste à analyser le langage Pseudo-Pascal, à l'interpréter, le traduire en C3A et à écrire un interprète de C3A [2]

1 Langage source : syntaxe

Définition formelle Le langage Pseudo-Pascal (PP en abrégé) est défini (syntaxiquement) par la grammaire suivante (au format de Bison) :

```
MP: L_vart LD C

 E: E Pl E
  | E Mo E
  | E Mu E
  | E Or E
  | E Lt E
  | E Eq E
  | E And E
  | Not E
  | '(' E ')'
  | I
  | V
  | true
  | false
  | V '(' L_args ')'
  | NewAr TP '[' E ']'
  | Et


Et: V '[' E ']'
  | Et '[' E ']'


C: C Se C
  | Et Af E
```

1. v4 du 03/04/17 corrigeant v3 du 28/03/17, v2 du 27/03/17, v1 du 21/03/17 et v0 du 19/03/17
2. ce projet suit l'idée générale de [Aho-Ullman-Sethi, Compilateurs ...,Intereditions 1990], pages 811-818 ; le langage PP (et sa sémantique) est une légère variante de celui décrit par F. Pottier à l'adresse http ://pauillac.inria.fr/~fpottier/X/INF564/

```
  | V Af E
  | Sk
  | '{' C '}'
  | If E Th C El C
  | Wh E Do C
  | V '(' L_args ')'


L_args: %empty
      | L_argsnn


L_argsnn: E
        | E ',' L_argsnn


L_argt: %empty
      | L_argtnn


L_argtnn: Argt
        | L_argtnn ',' Argt


Argt: V ':' TP


TP: T_boo
  | T_int
  | T_ar TP


L_vart: %empty
      | L_vartnn


L_vartnn: Var Argt
        | L_vartnn ',' Var Argt


D_entp: Dep V '(' L_argt ')'


D_entf: Def V '(' L_argt ')' ':' TP


D: D_entp L_vart C
```

```
  | D_entf L_vart C


LD: %empty
  | LD D
```

où les symboles `I V T_ar NewAr T_boo T_int Def Dep Sk Af true false Se If Th El Var Wh Do` représentent les unités lexicales suivantes :
`I` : une suite de chiffres, non-vide, commençant par un chiffre non-nul
`V` : un identificateur de variable ou fonction ou procédure
`T_ar` : un mot de la forme `array of`
`NewAr` : un mot de la forme `new array of`

symbole	T_boo	T_int	Def	Dep	Af	Sk	true	false	Se	If	Th	El
lexeme	boolean	integer	defun	defpro	:=	skip	true	false	;	if	then	else

symbole	Var	Wh	Do	Pl	Mo	Mu	And	Or	Not	Lt	Eq
lexeme	var	while	do	+	-	*	and	or	not	<	=

(i.e. ces unités lexicales ne comportent qu'un lexème, qui est donné dans le tableau).
On appelle *commande* (ou instruction) un mot engendré par le non-terminal `C` ;
une *commande atomique* est une commande qui n'est pas décomposable sous la forme c_1 `Se` c_2 pour des commandes c_1, c_2 ;
une *expression* est un mot e engendré par le non-terminal `E` ;
une *expression tabulaire* est un mot e engendré par le non-terminal `Et`.

Un programme PP est un texte engendré par cette grammaire qui vérifie les contraintes supplémentaires suivantes :
- toute expression est typable
- toute affectation a des membres gauches et droit de même type
- tout appel de fonction (resp. procédure) a des paramètres d'appel qui ont la *même suite de types* que la suite des paramètres formels de la fonction (resp. procédure).

Quelques intuitions Les notions sous-jacentes aux non-terminaux utilisés sont les suivantes :

— `MP` : Main Program
— `E` : Expression (peut avoir une valeur de type entier, booléen ou tableau)
— `Et` : Expression de la forme Id[e1][e2]...[en] peut avoir une valeur de type entier, booléen ou tableau)
— `C` : Commande (= Instruction)
— `L_args` : Liste d'expressions (par ex. la liste des arguments actuels d'une fonction)
— `L_argt` : Liste de variables typées (par ex. la liste des arguments formels d'une fonction)
— `TP` : Type
— `L_vart` : Liste de déclarations de variables typées (par ex. la liste des variables locales d'une fonction)
— `D_entp` : en-tête d'une définition de procédure
— `D_entf` : en-tête d'une définition de fonction

— `D` : définition d'une procédure ou fonction
— `LD` : liste de définitions de procédures ou fonctions

Une difficulté ? L'analyseur lexical ne distinguera pas un nom de variable d'un nom de fonction ou de procédure. En revanche l'analyseur syntaxique pourra reconnaître qu'un identificateur (qui est facteur d'une entête de déclaration de fonction ou de procédure) est un nom de fonction (ou de procédure). L'analyseur sémantique pourra vérifier (grace à la table des symboles) qu'un identificateur suivi d'une suite d'expressions entre parenthèses est le nom d'une fonction (ou d'une procédure).

2 Langage source : sémantique

Les objets manipulés par un programme PP ont une *dimension* qui est un entier. Un objet de dimension 0 est un entier sur 32 bits i.e. un éléments de $\mathbb{Z}/n\mathbb{Z}$, avec $n := 2^{32}$ ou un booléen. Un objet de dimension $d \geq 1$ est une liste d'objets de dimension $d-1$.
Le tas est un arbre non-ordonné : la racine a une nombre fini de fils et chaque fils est un objet de dimension $d \geq 1$.
La sémantique de PP est définie ci-dessous. Le procédé de définition employé s'appelle une sémantique *opérationnelle à grands pas*.

Idée générale
La sémantique définit l'évolution de l'état d'une machine abstraite sous l'effet d'une commande (ou d'une expression) de PP. Un *état* est un triplet (G, H, E) où G est un environnement global (un vecteur de valeurs pour les variables globales), H ("heap", en Anglais) est un tas et E est un environnement local (un vecteur de valeurs pour les variables locales d'une fonction/procédure).
Une commande c fait passer la machine d'un état (G, H, E) à un état (G', H', E') ce que l'on note

$$G, H, E/c \to G', H', E'$$

Une expression e fait passer la machine d'un état (G, H, E) à un état (G', H', E'), et produit la valeur v ce que l'on note

$$G, H, E/e \to G', H', E'/v.$$

Les règles (voir le paragraphe suivant) décrivent comment, connaissant certaines transitions de la machine, on peut en déduire d'autres. L'ensemble des transitions de la machine est le *plus petit* ensemble de transitions satisfaisant les règles.

Sémantique des opérateurs
Les opérateurs d'arité k ($k \in \{1, 2\}$) sur les entiers (resp. les booléens) ont la sémantique habituelle, qui est une application $(\mathbb{Z}/n\mathbb{Z})^k \to \mathbb{Z}/n\mathbb{Z}$ (dans le cas de `Pl`, `Mo`,`Mu`) ou bien $(\mathbb{Z}/n\mathbb{Z})^k \to \mathbb{B}$ (dans le cas de `Lt`, `Eq`) ou bien $\mathbb{B}^k \to \mathbb{B}$ (dans le cas de `And`, `Or`, `Not`).

Sémantique des expressions

$$\text{Constante}\quad G,H,E/k \to G,H,E/k \qquad \frac{x \in dom(E)}{G,H,E/x \to G,H,E/E(x)}\ \text{Variable locale} \qquad \frac{x \in dom(G) \setminus dom(E)}{G,H,E/x \to G,H,E/G(x)}\ \text{Variable globale}$$

$$\frac{G,H,E/e \to G',H',E'/v}{G,H,E/uop\ e \to G,H,E/[\![uop]\!](e)}\ \text{Opérateur unaire} \qquad \frac{S/e_1 \to S'/v_1 \quad S'/e_2 \to S''/v_2}{S/e_1\ bop\ e_2 \to S''/[\![bop]\!](v_1,v_2)}\ \text{Opérateur binaire}$$

$$\frac{S/e_1 \to S'/\ell \quad S'/e_2 \to S''/n \quad S''=G'',H'',E'' \quad H''(\ell)=v_0\dots v_{p-1} \quad 0\le n<p}{S/e_1[e_2] \to S''/v_n}\ \text{Lecture dans un tableau}$$

$$\frac{S/e \to G',H',E'/n \quad n\ge 0 \quad \ell \notin dom(H') \quad H''=H' \cup \{\ell \mapsto (default(\tau))^n\}}{S/\text{new array of } \tau[e] \to G',H'',E'/\ell}\ \text{Allocation d'un tableau}$$

Pour toute fonction f déclarée par

$$f(x_1:\tau_1,\dots,x_n:\tau_n):\tau \text{ var } x'_1:\tau'_1,\dots,\text{var } x'_m:\tau'_m c$$

$$\text{Appel de fonction}\quad \frac{\begin{array}{c}\forall j\in[1,n], S_{j-1}/e_j \to S_j/v_j,\ \ S_n=G',H',E_n \\ E'=(x_j\mapsto v_j)\cup(x'_j\mapsto default(\tau'_j))\cup(f\mapsto default(\tau)) \\ G',H',E'/c \to G'',H'',E''v=E''(f)\end{array}}{S_0/f(e_1,\dots,e_n) \to G'',H'',E_n/v}$$

Sémantique des commandes (ou instructions)

$$\text{Séquence}\quad \frac{S_0/c_1 \to S_1,\ \ S_1/c_2 \to S_2}{S_0/c_1\ \texttt{Se}\ c_2 \to S_2}$$

$$\frac{S/e \to G',H',E'/v \quad x\in dom(E')}{S/x:=e \to G',H',E'[x\mapsto v]}\ \text{Affectation : variable locale} \qquad \frac{S/e \to G',H',E'/v \quad x\in dom(G')\setminus dom(E')}{S/x:=e \to G'[x\mapsto v],H',E'}\ \text{Affectation : variable globale}$$

$$\frac{S/e \to S'/\text{true} \quad S'/c\ \text{Se}\ \text{Wh}\ e\ \text{Do}\ c \to S''}{S/\ \text{Wh}\ e\ \text{Do}\ c \to S''}\ \text{While true} \qquad \frac{S/e \to S'/\text{false}}{S/\ \text{Wh}\ e\ \text{Do}\ c \to S'}\ \text{While false}$$

$$\frac{S/e \to S'/\text{true} \quad S'/c_1 \to S''}{S/\text{IfeThc}_1\text{Elc}_2 \to S''}\ \text{IfThEl true} \qquad \frac{S/e \to S'/\text{false} \quad S'/c_2 \to S''}{S/\text{IfeThc}_1\text{Elc}_2 \to S''}\ \text{IfThEl false}$$

$$\text{Écriture dans un tableau}\quad \frac{S/e_1 \to S'/\ell \quad S'/e_2 \to S''/n \quad S'/e_3 \to G''',H''',E'''/v \quad H''(\ell)=v_0\dots v_{p-1} \quad 0\le n<p}{S/e_1[e_2]:=e_3 \to G''',H'''[\ell\mapsto v_0,\dots v_{n-1}vv_{n+1}\dots v_{p-1}],E'''}$$

Pour toute procédure f déclarée par

$$f(x_1:\tau_1,\dots,x_n:\tau_n) \text{ var } x'_1:\tau'_1,\dots,\text{var } x'_m:\tau'_m c$$

Appel de procédure

$$\frac{\begin{array}{c}\forall j \in [1,n], S_{j-1}/e_j \to S_j/v_j, \quad S_n = G', H', E_n \\ E' = (x_j \mapsto v_j) \cup (x'_j \mapsto default(\tau'_j)) \\ G', H', E'/c \to G'', H'', E''\end{array}}{S_0/f(e_1, \ldots, e_n) \to G'', H'', E_n}$$

Programme

$$\frac{G = (x_j \mapsto default(\tau_j)) \quad G, \emptyset, \emptyset/c \to G', H', E'}{\text{var } x_1 : \tau_1, \ldots, \text{var } x_n : \tau_n d_1 d_2 \ldots d_n c \to G', H'}$$

Précisions :
- dans les règles While true , While false , la commande c est supposée *atomique* et dans les règles IfThEl true , IfThEl false , la commande c_2 est supposée *atomique*.
- on suppose que, dans chaque déclaration de fonction (ou de procédure), l'ensemble des paramètres est disjoint de l'ensemble des variables locales ; le nom de la fonction est vu comme une variable locale, dont le type est le type de la fonction
- si le même nom de fonction/procédure fait l'objet de plusieurs déclarations, alors les *profils* de ces déclarations doivent être identiques et c'est la dernière déclaration qui définit la sémantique de cette fonction (ou procédure).

3 Langage intermédiaire

Un programme écrit dans le langage C3A (Code à 3 Adresses) consiste en une suite de lignes, chacune de la forme :

$$\textit{Etiquette} : \textit{Operateur} : \textit{Argument} : \textit{Argument} : \textit{Destination}$$

Les champs *Etiquette*, *Destination* sont des identificateurs i.e. des mots qui consistent en une lettre suivie d'un nombre quelconque de lettres ou chiffres. *Operateur* est l'un des symboles :

`Pl`, `Mo`, `Mu`, `And`, `Or`, `Lt`, `Ind`, `Not`, `Af`, `Afc`, `AfInd`, `Sk`, `Jp`, `Jz`, `St`, `Param`, `Call`, `Ret`

et *Argument* peut être soit un identificateur, soit un numéral i.e. une suite de chiffres, précédée éventuellement d'un signe ("+" ou "-").
Les variables globales sont soit de type entier soit de type tableau. Dans ce dernier cas il s'agit, pour chaque variable, d'un tableau de dimension 1, d'entiers, d'adresse fixe et de taille fixe. Cela revient à dire que le code manie :
- des *constantes* entières désignées par des "numéraux"
- des *constantes* de type tableau d'entiers, désignées par des identificateurs (du syle `TAS`,`TAILLE`...) ; ces tableaux sont de dimension 1
- des *variables* (globales ou locales) de type entier. La sémantique de ce langage est définie (informellement) comme suit :
soient $\rho_g, \rho_\ell, \rho_p$ des environnements i.e. des fonctions associant à des identificateurs entiers des valeurs entières ; on les nomme respectivement environnement *global*, environnement *local*, environnement *paramétrique*.
- `Pl` affecte à la variable *Destination* la somme du premier et du second argument
- `Mo` affecte à la variable *Destination* la différence du premier et du second argument
- `Mu` affecte à la variable *Destination* le produit du premier et du second argument

- `And` affecte à la variable *Destination* la conjonction du premier et du second argument
- `Or` affecte à la variable *Destination* la disjonction du premier et du second argument
- `Lt` affecte à la variable *Destination* l'entier 1 (resp. 0) si $Arg1 < Arg2$ (resp. $Arg1 >= Arg2$)
- `Ind` affecte à la variable *Destination* la valeur de $Arg1[Arg2]$
- `Not` affecte à la variable *Destination* la négation du premier argument
- `Af` provoque une affectation de la valeur du second argument au premier argument
- `Afc` provoque une affectation de la valeur du premier argument, qui est un numeral, à la variable *Destination*
- `AfInd`, affecte à l'emplacement $Arg1[Arg2]$ la valeur de *Destination*[3]
- `Sk` ne provoque aucune modification de l'environnement
- `Jp` provoque un saut à l'instruction étiquetée par *Destination*
- `Jz` provoque un saut à l'instruction étiquetée par *Destination*, dans le cas où le premier argument est nul
- `St` arrête l'exécution
- `Param` ajoute à ρ_p le couple ($Arg1$, valeur de $Arg2$)
- `Call` : $Arg2$ est un numéral ; affecte à ρ'_l l'environnement ρ_p (ca1), enlève de ρ_p les $Arg2$ derniers couples (ca2), affecte à ρ'_p l'environnement vide (ca3), lance (récursivement) l'exécution (ca4) à l'instruction d'étiquette $Arg1$ avec le nouveau triplet $(\rho_g, \rho'_l, \rho'_p)$.
- `Ret` revient à l'instruction qui suit le dernier `Call` exécuté, revient au couple (ρ_g, ρ_l) mémorisé à l'époque de ce dernier `Call`, (donc au ρ_p obtenu lors de ce call, étape (ca2)).

En fait, vue cette sémantique, seul le champ *Operateur* doit impérativement être correctement rempli ; les autres champs peuvent être "vides" c'est à dire consister en une suite de caractères "blanc" ou "tabulation", lorsque l'opération (de la même ligne) ne les utilise pas et ils ne sont pas la destination d'une instruction de saut (depuis une autre ligne).

4 Travail à réaliser

Partie obligatoire

1- Un *analyseur syntaxique* du langage PP : cette fonction teste que le texte donné en entrée est bien engendré par la grammaire de PP (et envoie un message d'erreur, sinon). Elle construit une table des symboles contenant toutes les informations qui seront nécessaires à un interpréteur ou un compilateur :
- nom et type des variables globales
- nom des fonctions et procédures, listes de leurs arguments (typés), type éventuel du résultat, corps.

2- Un *analyseur sémantique* du langage PP : cette fonction teste que les "contraintes supplémentaires", de nature sémantique, annoncés après la grammaire (section 1) sont satisfaites par le programme (et envoie un message d' erreur expliquant la nature de l'erreur détectée, sinon). L'analyseur sémantique achèvera le typage des expressions (déjà amorcée par les analyseurs lexicaux et syntaxique).

3- Un *interpréteur* du langage PP : il s'agit d'une fonction qui prend entrée une commande c, un environnement global G, un tas H, un environnement local E et retourne le triplet $(G', H', E') = [\![c]\!]_{\mathrm{PP}}(G, H, E)$ défini par la sémantique de PP.

3. ce troisième argument ne dénote donc plus une "destination" , dans ce cas

4- Un *traducteur* (ou compilateur) de P vers C3A.
5- Un *interpréteur* du langage C3A : il s'agit d'une fonction qui prend entrée un programme C3A P et un environnement global ρ et retourne l'environnement $[\![P]\!]_{\text{c3a}}\ (\rho)$.

Extensions(facultatives)
6- Un traducteur (ou compilateur) de C3A vers Y86 : il s'agit d'une fonction qui prend entrée un programme P (écrit en C3A) et retourne un programme Q (écrit en Y86) et un placement des variables globales $x \in V$ dans des adresses $\text{ad}(x)$ du processeur de façon que : pour tout environnements ρ_g, si on interprète Q, avec un processeur Y86, où initialement la valeur stockée dans $\text{ad}(x)$ est $\rho_g(x)$, alors, en fin de calcul, la valeur stockée dans $\text{ad}(x)$ est

$$[\![P]\!]_{c3a}(\rho_g)(x).$$

7- Un "ramasse-miettes" pour l'interprète de la question 3) i.e. on fera en sorte que les zones du tas devenues inaccessibles par les variables du programme soient réutilisées pour stocker les nouveaux objets créés par les instructions `new array of ...`

5 Modalités de réalisation

Groupes

Le projet doit être réalisé par groupes d'au plus 4 étudiants. Les projets réalisés par ≥ 5 étudiants *ne* seront *pas* pris en considération. Un responsable par groupe communiquera le projet à son enseignant de TD.

Fichiers

Chaque responsable placera dans une archive projetcompi2.tar des sources :

- `ppascal.l` (flex), `ppascal.y` (bison),
- différents modules `*.h`, `*.c` qui lui sembleront utiles ; on peut penser à un module qui traite les arbres "abstraits", un module qui traite l'analyse sémantique, un module qui traite l'interprétation de PP, un module qui traite le code intermédiaire (traduction de PP vers C3A et interprétation de C3A).
- un `Makefile` produisant différents programmes exécutables à partir de ces sources (par exemple une commande `interpp` qui interprète les programmes PP, une commande `tradpp2c3a` qui traduit les programmes PP en programmes C3A et une commande `interpp_c3a` qui traduit les programmes PP en programmes C3A puis interprète le code C3A).
- un *rapport* d'une dizaine de pages qui explique le plan de votre programme, les difficultés rencontrées, les solutions mises en oeuvre, les tests effectués. Ce document doit pouvoir être lu indépendamment des fichiers sources des programmes et doit être rédigé d'une façon aussi pédagogique que possible.

Le fichier `ppascal.y` contiendra (en commentaire) :
- les noms des étudiants auteurs du devoir.
- le nom de la fonction C (et du module qui la contient) répondant à chacune des questions 1,2,3,4,5 de la section 4.

L'archive doit être envoyée à l'enseignant du groupe de TD du responsable, avant le 10/04/17 à 23h 59.

Notation

La notation sera établie à partir du rapport (qui sera lu), des programmes écrits (qui seront testés), de leurs sources (qui seront parcourues), et d'une soutenance qui aura lieu en semaine 15 (10/04 au 14/04). La note de soutenance est individuelle ; elle dépend de la qualité de votre part d'exposé (transparents et explications orales).

Documents

On trouvera sur le site `http://dept-info.labri.fr/ENSEIGNEMENT/compi` :
- la documentation Flex et Bison
- quelques fonctions C utiles (fichiers `environ.c`, `bilquad.c`), qui restent à adapter à des variables *typées*.

Responsables

Pour tous cas particuliers, adressez-vous à votre enseignant de TD/TP :
H. Derycke, email :`henri.derycke@u-bordeaux.fr` (gr. A5)
L. Clément, email :`lionel.clement@u-bordeaux.fr` (gr. A1,A3)
G. Sénizergues, email :`geraud.senizergues@u-bordeaux.fr` (gr. A2,A4).