Maîtrise d'informatique
Module de Projet 1

Examen – Compilateur Pascal

Philippe MARQUET

Janvier 1998

Documents autorisés — Durée : 2 heures

On définit le langage PPx comme une extension du langage PP1 (grammaire page 26 du polycopié) autorisant la déclaration de types simples permettant la manipulation de tableaux d'entiers à une dimension. Les types tableaux sont déclarés suivant la règle :

```
BLOCK    : :=  CONSTS TYPES VARS INSTS
TYPES    : :=  type TYPE {; TYPE }; | ε
TYPE     : :=  ID-TYPE (NUM)
```

Une définition de type `TYPE` déclare `ID-TYPE` comme type tableaux à une dimension de taille `NUM` ; les tableaux de ce type seront indicés de 0 à `NUM-1`.

On modifie la règle `VARS` pour autoriser la déclaration de variables entières simples ou de variables tableaux d'entiers :

```
VARS    : :=  var VAR {; VAR }; | ε
VAR     : :=  ID-INT { , ID-INT } : int | ID-TAB { , ID-TAB } : ID-TYPE
```

`ID-INT`, `ID-TAB`, et `ID-TYPE` représentent des identificateurs qui dénotent respectivement une variable simple, une variable tableau et un type tableau.

Exemple :

```
type ARRAY1 (12) ; ARRAY2 (5) ;
var  A, B, C : int ;
     X1, X2 : ARRAY1 ;
     Y1, Y2, Y3 : ARRAY2 ;
```

On modifie les règles `FACT`, `AFFEC`, et `LIRE` pour autoriser la référence à un élément d'une variable de type tableau ; on ajoute l'affectation directe entre tabeaux et la promotion des scalaires aux tableaux :

```
VAL-INT    : :=  ID-INT | ID-TAB ( EXPR )
VAL-TAB    : :=  ID-TAB
FACT       : :=  VAL-INT | NUM | ( EXPR )
LIRE       : :=  read ( VAL-INT { , VAL-INT } )
AFFEC      : :=  VAL-INT := EXPR
               | VAL-TAB := ID-TAB
               | VAL-TAB := ( ID-TYPE ) EXPR
```

L'affectation se dérive sous trois formes :

- la première forme est l'affectation scalaire habituelle ;
- la seconde forme est l'affectation directe entre tableaux de même type. Chacun des éléments du tableau partie gauche de l'affectation est affecté de la valeur de l'élément correspondant du tableau partie droite de l'affectation ;
- la troisième forme est la promotion explicite d'une valeur scalaire à une valeur tableau du type `ID-TYPE` qui doit aussi être le type du tableau partie gauche de l'affectation. La valeur résultat de la partie gauche est un tableau du type `ID-TYPE` dont toutes les valeurs sont identiques à la valeur de l'expression scalaire `EXPR` ; ce tableau est affecté au tableau partie gauche de l'affectation.

Id : xam98.tex,v 1.3 2000/11/28 18 :34 :54 marquet Exp

Exemple utilisant les déclarations précédentes :

```
A  := B + C  ;
X1 := (ARRAY1) A ;
X2 := X1 ;
Y1 := (ARRAY2) X2 (B) + X1 (X2 (B)) ;
```

Exercice 1 (Table des symboles)

1. Donner une struture possible pour la table des symboles du langage PPx. Illustrer l'utilisation de cette struture sur l'exemple de déclarations donné ci-dessus.
2. Écrire les procédures `TYPES` et `VARS` qui analysent les déclarations de PPx et construisent la table des symboles.

On suppose que la procédure `VAL` d'analyse de la référence à une valeur :
`VAL   : :=  VAL-INT | VAL-TAB`
génère le code qui laisse sur la pile l'adresse de cette valeur et retourne dans un paramètre le type de la valeur analysée :

```
PROCEDURE VAL (var VT : VTYPE) ;
```

Exercice 2 (Typage des valeurs)

1. Donner une définition du type `VTYPE` utilisé par la procédure `VAL` pour retourner sa valeur.
2. Écrire le code de la procédure `AFFEC` d'analyse syntaxique et sémantique de l'affectation.

Pour implanter les nouvelles formes de l'affectation, on enrichit le langage P-Code (page 19 du polycopié) de deux nouvelles instructions :

`MOV n` attend deux adresses sur la pile (`a1` au sous-sommet et `a2` au sommet) et réalise la recopie de `n` valeurs lues à partir de l'adresse `a2` dans les adresses successives à partir de `a1`.

`FIL n` attend une adresse `a` au sous-sommet de la pile et une valeur `v` au sommet de la pile. Elle récopie la valeur `v` dans chacune des `n` adresses qui suivent `a`.

Exercice 3 (Génération de P-Code)

1. Écrire l'extension de l'interpréteur P-Code pour les deux instructions `MOV n` et `FIL n`.
2. Donner la structure du P-Code à générer pour chacune des trois formes de l'affectation.
3. Étendre le code la procédure `AFFEC` de l'exercice 2.2 avec les instructions de génératon de code ad hoc.

On reprend dans PPx les procédures paramétrées, non imbriquées, avec déclarations locales de PP8 (chapitre 8, page 101 du polycopié) et les extensions du P-Code introduites pour la génération de code PP8. Comme dans PP8, les paramètres scalaires sont passés par adresse ou par valeur suivant la présence ou non du mot clé `var`. Les paramètres tableaux sont systématiquement passés par adresse. On convient que la procédure appelante laisse sur la pile les valeurs ou adresses des paramètres effectifs dans l'ordre inverse de leur déclaration et que la procédure appelée accède à ces paramètres par un adressage qui est un déplacement négatif par rapport au registre `BASE` à l'aide de l'instruction P-Code `LDL`. Les variables locales sont allouées sur la pile et accédées par un déplacement positif par rapport à `BASE`. Contrairement à ce qui est présenté dans le polycopié, l'instruction P-Code `RET` de retour de procédure n'accepte pas de paramètre et la procédure appelante retrouve au retour de l'appel les paramètres en sommet de pile. La procédure appellante réalise le nettoyage de la pile par un déplacement du sommet de pile (instruction P-Code `INT`).

```
program EXAM1 ;
type ARRAY1 (12) ; ARRAY2 (5) ;
var  A : int ;
     X, Y : ARRAY1 ;
procedure P (var I: int; J :int ; U, V: ARRAY1) ;
  var K : int ; T : ARRAY2 ;
  begin
    K := I + J ;
    U := V ;
    T := (ARRAY2) U (2) ;
  end ;
begin
  read (A) ;
  X := (ARRAY1) A ;
  Y := X ;
  P (A, A, X, Y) ;
end .
```

FIG. 1 – Programme PPx à traduire en P-Code.

Exercice 4 (Génération de code d'appel de procédure)

1. Dessiner l'état de la pile et des registres de la P-Machine lors de l'exécution du programme de la figure 1 avant l'exécution de la première instruction de la procédure `P`.
2. Donner le P-Code généré pour le programme de la figure 1.

On étend le langage PPx pour autoriser la manipulation par les procédures de paramètres tableaux dont le type (donc la taille) n'est pas connu à la compilation. Ces paramètres sont identifiés par un type prédéfini `assume`. Un exemple de tel programme est donné à la figure 2. La sémantique du langage assure les paramètres d'une même déclration `assume` doivent être de même type.

Exercice 5 (Question Bonus — Extension au type `assume`)
Proposez les modifications à apporter pour la compilation de cette nouvelle version de PPx.

```
program EXAM2 ;
type TI (12) ; TX (5) ;
var  R, S : int ;
     I, J : TI ;
     X, Y : TX ;
procedure PP (A, B : assume) ;
  begin
    B := A ;
    A := (assume) 0 ;
  end ;
begin
  read (R, S) ;
  I := (TI) R ;
  X := (TX) S ;
  PP (I, J) ;
  PP (X, Y) ;
end .
```

FIG. 2 – Extension de PPx au type `assume`.