Université de Bordeaux. Collège Sciences et Techniques.
Licence d'Informatique, module Compilation, 2016/2017.

Compilation

Examen du 25/04/17

Exercice 1 (10 pts) **Grammaires et attributs**
On considère la grammaire G suivante (au format de Bison) :

```
expr: expr op expr | N ;
```

où `expr` est un non-terminal et `N`, `op` sont des lettres terminales.
1- Cette grammaire est-elle ambiguë ? Est-elle LALR(1) ?

On considère les trois grammaires suivantes, sur les mêmes ensembles de lettres non-terminales et terminales :
G_1 :

```
expr : expr op N | N ;
```

G_2 :

```
expr : N op expr | N ;
```

G_3 :

```
expr : N op expr |expr op N | N ;
```

2- Parmi les grammaires G_i ($i \in \{1, 2, 3\}$), lesquelles engendrent le *même langage* que la grammaire G ? lesquelles sont non-ambiguës ?
3- La table de l'analyseur syntaxique LR généré par BISON à partir de G est donnée dans l'annexe 1. Quel arbre de dérivation cet analyseur construit-il à partir du mot `N op N op N` ?

On souhaite interpréter des expressions de la forme `N op N ... op N` pour l'opérateur d'exponentiation, noté $**$, sur les entiers. L'opération d'exponentiation est définie par $a{**}b := a^b$, c'est-à-dire : $\forall a \in \mathbb{N}, \forall b \in \mathbb{N}$

$$a{**}0 = 1, \;\; a{**}(b+1) = (a{**}b) * a.$$

4- Que valent les expressions :

$$e_1 := (2{**}2){**}3, \;\; e_2 := 2{**}(2{**}3)?$$

Ont-elles la même valeur ?
Rappelons qu'une opération binaire $\odot$ sur un ensemble D est *associative* ssi

$$\forall x \in D, \forall y \in D, \forall z \in D, (x \odot y) \odot z = x \odot (y \odot z).$$

L'opération $**$ sur l'ensemble des entiers $\mathbb{N}$ est-elle associative ?
5- On choisit d'interpréter l'expression non-parenthésée $2{**}2{**}3$ comme $2{**}(2{**}3)$.
Quel arbre de syntaxe abstraite doit être associé à $2{**}2{**}3$? (faire un dessin).
De façon générale une expression $N_1{**}...N_{p-2}{**}N_{p-1}{**}N_p$ est interprétée comme

$$N_1{**}(...(N_{p-2}{**}(N_{p-1}{**}N_p))...).$$

6- Supposons que l'on utilise une des grammaires G ou G_i ($i \in \{1,2,3\}$), avec un analyseur lexical qui envoie le token [1] N à partir d'un numéral et le token `op` à partir du mot `**`. Pour chacune des quatre grammaires G, G_1, G_2, G_3, expliquer en quoi elle est bien (ou mal) adaptée à l'écriture (au moyen d'attributs synthétisés) d'un interpréteur de ces expressions.
7- Choisir l'une des grammaires G, G_1, G_2, G_3, et écrire des actions sémantiques permettant d'afficher la valeur d'une expression exponentielle, en fin d'analyse.
8- Le professeur Cosinus a (malencontreusement) choisi la grammaire G_1 et se demande s'il peut, par un choix d'attributs adéquats, évaluer les expressions exponentielles. Pouvez-vous l'aider ? c'est-à-dire écrire des actions sémantiques, associées aux règles de G_1, permettant d'évaluer les expressions exponentielles.

On considère la grammaire H suivante (au format de Bison) :

```
expr: expr op1 expr | expr op2 expr | N ;
```

où `expr` est un non-terminal et `N`, `op1`, `op2` sont des lettres terminales.
Cette grammaire est utilisée avec un analyseur lexical qui envoie le token N à partir d'un numéral, le token `op1` à partir du mot `+` et le token `op2` à partir du mot `**`.
9- La grammaire H est-elle ambiguë ? Est-elle LALR(1) ?

On souhaite évaluer les expressions engendrées par cette grammaire (nous les dénommerons "H-expressions") en interprétant `+` par l'addition des entiers, `**` par l'exponentiation des entiers. On maintient le choix de la question 5.
De plus on choisit d'interpréter a`+`b`**`c de la même façon que a`+(`b`**`c`)`,
et a`**`b`+`c de la même façon que `(`a`**`b`)+`c.
10- L'expression 2+3+4 a-t-elle plusieurs arbres de dérivation dans H ?
Ce phénomène est-il gênant pour utiliser H dans le but d'évaluer les H-expressions ?
11- Écrire un fichier BISON permettant d'évaluer les H-expressions.

Exercice 2 (10 pts) **Interprétation des tableaux**

Rappels (tp6)

On souhaite interpréter les instructions de construction, d'affectation et d'évaluation des tableaux du langage TPPascal (la version réduite de PP, sans fonction ni procédure, étudiée au TP6). Les algorithmes mis en oeuvre par l'interpréteur seront décrits (dans l'énoncé et dans les copies) en "C libre" .
On définit 3 tableaux statiques : ADR,TAL,TAS, de tailles respectives ADMAX,ADMAX,TASMAX, qui stockent respectivement l'adresse de début d'un tableau, la taille du tableau et les cases du tableau. Un tableau de dimension 0 est simplement un entier. Un tableau T de dimension $k+1$ est représenté par un entier t :
- un tableau nil est représenté par l'entier 0 (et `ADR`$[0] = 0$)
- si T ne vaut pas nil, il est représenté par un entier t, et pour tout entier $i <$ `TAL`$[t]$, la variable $T[i]$ est représentée par `TAS[ADR[t] + i]`.
De plus, la première adresse libre dans `ADR` (resp. `TAS`) est stockée dans la variable de type entier `padrl` (resp. `ptasl`). Ainsi :

1. le code d'unité lexicale

- on évalue `X[Y]` par `TAS[ADR[X] + valeur(Y)]`
- on exécute `X[Y]:= Z` par `TAS[ADR[X] + valeur(Y)]:= valeur(Z)`
- on exécute `NewAr(e)` (qui représente une expression `new array of array ...of integer[e]`) par :

```
taille=valeur(e);            /* taille du tableau              */
res=padrl;
ADR[res]=ptasl;
TAL[res]=taille;
padrl++;ptasl+=taille;     /* mise a jour allocateur  memoire   */
return(res);
```

Un exemple

On note $\varphi : \{0,1\}^* \to \{0,1\}^*$ l'homomorphisme de monoïdes défini par

$$\varphi(0) := 01, \;\; \varphi(1) := 10.$$

Par exemple :

$$\varphi(0) = 01, \;\; \varphi(001) = 010110, \;\; \varphi(100) = 100101.$$

1- Vérifier que, si $w \in \{0,1\}^*$ est un mot de longueur ℓ, alors $\varphi(w)$ est un mot de longueur $2 \cdot \ell$.
On considère la suite de mots $(t_n)_{n\in\mathbb{N}}$ définie par :

$$t_0 := 0, \;\; t_{n+1} := \varphi(t_n).$$

2- Vérifier que,

$$t_1 := 01, \;\; t_2 := 0110, \;\; t_3 := 01101001.$$

Que vaut t_4 ?
3- Exprimer par une formule simple l'entier $\ell_n := |t_n|$ (la longueur du mot t_n) en fonction de n.
On considère le programme `motthue.pp` (voir annexe 2).
4- Vérifier que, à la fin de la boucle `while (p < n)`, la propriété en commentaire est valide i.e. :

$$L = 2^p, W = \varphi^p(0).$$

5- En ce point du programme, la valeur de la variable `ptasl` de l'interpréteur est 2^{p+1}. Pouvez-vous expliquer pourquoi ?

On remplace maintenant, dans le programme `motthue.pp`, l'affectation `n:=3` de la ligne 10 par `n:=20`. On se place, dans l'exécution du programme `motthue.pp`, à la dernière instruction de la boucle `while (p < n)`. On sait que `ptasl` vaut 2^{p+1}.
6- Quelle est la zone du tableau `TAS` qui reste accessible au programme, c'est-à-dire que le programme pourra lire et modifier ?
Lorsque p vaut 20 quelle est la taille de la zone du tableau `TAS` devenue inaccessible ?

Ramasse-miettes

On souhaite remédier à cette perte d'espace mémoire par un mécanisme de *ramasse-miettes*. On ne traitera, dans les questions 7,8,9, que le cas des tableaux (dynamiques) de dimension 1.
On introduit un nouveau tableau `REF` de taille `ADMAX` et un ensemble[2] `ADRL` qui contient les "adresses libres" de `ADR` :
- `REF[i]` stocke le nombre de variables du programme qui ont pour valeur i,
- `ADRL` stocke l'ensemble des indices $i \in [0, \texttt{ADMAX} - 1]$ tels que `REF[i]`=0 (la variable entière `padrl` est désormais inutile).
L'instruction `T := T'` (où `T,T'` sont des variables de type tableau de dimension1) est alors interprétée par :

```
REF[T'] = REF[T']+1;
REF[T] = REF[T]-1;
if (REF[T]==0)
    {
     insert(T,ADRL);/* inserer l'entier T dans ADRL*/
     TAL[T]=0;
    };
T=T';
```

7- Réécrire l'interprétation de l'expression `NewAr(e)`, de façon à renvoyer la plus petite adresse libre de `ADRL` et à mettre à jour les structures `ADR,TAL,REF,ADRL` et l'entier `ptasl`.

On décide (lorsque la valeur de `ptasl` est plus grande que la moitié de l'entier `TASMAX`) de "tasser le TAS vers la gauche".
8- Écrire une suite d'instructions qui place dans un tableau `INDL`, de taille `ADMAX`, la liste des indices $i \in [0, \texttt{ADMAX} - 1]$ tels que `TAL[i]` $\neq 0$, dans un ordre tel que :

`ADR[INDL[0]]<ADR[INDL[1]]< ...<ADR[INDL[k]]<ADR[INDL[k+1]]< ...`

(on pourra utiliser une fonction de tri disponible dans une bibliothèque ; on pourra convenir que `INDL[k]>=TASMAX` si k est strictement plus grand que le nombre d'entiers i tels que `TAL[i]` $\neq 0$).
9- On souhaite placer le tableau placé à l'indice `INDL[j]` (dans `ADR`) dans la zone du tableau `TAS` d'indices compris entre $(\sum_{k=0}^{j-1} \texttt{TAL[INDL[k]]}) + 1$ et $(\sum_{k=0}^{j-1} \texttt{TAL[INDL[k]]}) + \texttt{TAL[INDL[j]]}$.
Écrire une procédure `TASSERG()` qui modifie les tableaux `ADR` et `TAS` de façon à réaliser ce nouveau placement.
10- Reprendre les questions 7,8,9 dans le cas général où les tableaux peuvent avoir une dimension $d \geq 0$ quelconque.

2. nous laissons de côté l'implémentation précise des ensembles

ANNEXE 1

```
État 5 conflits: 1 décalage/réduction

Grammaire

    0 $accept: expr $end
    1 expr: expr '+' expr
    2     | NUMBER


Terminaux, suivis des règles où ils apparaissent

$end (0) 0
'+' (43) 1
error (256)
NUMBER (258) 2


Non-terminaux, suivis des règles où ils apparaissent

$accept (5)
    à gauche: 0
expr (6)
    à gauche: 1 2, à droite: 0 1


État 0

    0 $accept: . expr $end
    1 expr: . expr '+' expr
    2     | . NUMBER

    NUMBER  décalage et aller à l'état 1

    expr  aller à l'état 2


État 1

    2 expr: NUMBER .

    $défaut  réduction par utilisation de la règle 2 (expr)


État 2

    0 $accept: expr . $end
    1 expr: expr . '+' expr

    $end  décalage et aller à l'état 3
    '+'   décalage et aller à l'état 4


État 3

    0 $accept: expr $end .

    $défaut  accepter


État 4

    1 expr: . expr '+' expr
    1     | expr '+' . expr
    2     | . NUMBER
```

```
    NUMBER  décalage et aller à l'état 1

    expr  aller à l'état 5


État 5

    1 expr: expr . '+' expr
    1     | expr '+' expr .  [$end, '+']

    '+'  décalage et aller à l'état 4

    '+'       [réduction par utilisation de la règle 1 (expr)]
    $défaut  réduction par utilisation de la règle 1 (expr)
```

ANNEXE 2

```
/* phi(0) := 01, phi(1) := 10 : morphisme de Thue-Morse */
/* calcule le mot t(n) := phi^n(0) */
var n: integer,
var p : integer,
var i: integer,
var L: integer,
var W: array of integer,
var nW: array of integer

n:= 3;
/* W := 0 */
p := 0; L := 1;
W := new array of integer[L];
W[0] := 0;
while (p < n ) do    /* W := phi(W) */
      {nW := new array of integer[2*L];
       i  := 0;
       while i < L do
             {if W[i] = 0
                 then {nW[2*i]    := 0;
                       nW[2*i +1] := 1
                       }
                 else {nW[2*i]    := 1;
                       nW[2*i +1] := 0
                       };
               i:= i+1
             };
        W := nW;
        p := p+1;
        L := 2*L
        /* L = 2^p, W = phi^p(0) */
        }
```