Partiel de Compilation

Promo 2004 - Avril 2002

Une rédaction simple mais convaincante et une mise en évidence des résultats seront très appréciées des correcteurs...

1 Typologie des Langages

Soit le programme suivant, écrit pour mettre en valeur les différences entre *Mode*s de passage d'arguments aux routines :

```
var t    : integer
    foo  : array [1..2] of integer;

procedure shoot_my (x : Mode integer);
begin
  foo[1] := 51;
  t      := 2;
  x      := x + 69;
end;

begin
  foo[1] := 1;
  foo[2] := 2;
  t      := 1;
  shoot_my (foo [t]);
end.
```

Compléter le tableau suivant (sur votre copie) en explicitant les valeurs de `foo[1]`, `foo[2]`, et `t` à la fin du *programme*.

Mode (Passage par...)	`foo[1]`	`foo[2]`	`t`
Valeur			
Valeur-Résultat (Algol W)			
Valeur-Résultat (Ada)			
Référence			
Nom (Algol 60)			

On rappelle qu'en Algol W, l'adresse de la lvalue est calculée au retour, alors qu'en Ada elle l'est à l'appel. On rappelle également que les erreurs de calcul en Ing1 ne sont pas admissibles.

2 Parsage LL

1. Écrire la grammaire naïve de l'arithmétique avec les terminaux «1» (le seul nombre), «-» la soustraction binaire, «/» la division, «(» et «)» les parenthèses.
2. En plusieurs étapes, en faire une grammaire adaptée au parsage LL(1).
3. Expliquer pourquoi l'implémentation LL(1) de cette grammaire sera sûrement plus performante que le parsage par automate LALR(1).

3 Parsage LALR(1)

Considérons la grammaire suivante, écrite selon la syntaxe Bicc/Yason.

```
%%
sentence: "PROC"  argument '.'  | "PROC"  parameter ';'
        | "MACRO" argument ';'  | "MACRO" parameter '.'   ;
argument: "VARIABLE";
parameter: "VARIABLE";
%%
```

1. Quel est son type de Chomsky ?
2. Est-elle ambiguë ?
3. Est-elle déterministe ?
4. Est-elle LR(1) ?
5. Sa compilation par Bicc/Yason échoue avec deux conflits réduction/réduction. Le rapport de génération de l'automate est fourni plus bas.
   Expliquer ce conflit, i.e., déchiffrer le rapport et expliquer dans quel cas on tombe sur ce conflit.
6. Expliquer l'*origine* de ce conflit, i.e., expliquer pourquoi cette grammaire contient un conflit.
7. Est-ce qu'en dépit du conflit, le parseur est sain ? (On rappelle que par exemple dans le cas du `else` qui pendouille, le conflit est inoffensif.) Justifier.

Le rapport de Bison est :

```
L'état 4 contient 2 conflicts réduction/réduction.

Grammaire

  Numéro, Ligne, Règle
    0   2 $axiom -> sentence $
    1   2 sentence -> "PROC" argument '.'
    2   2 sentence -> "PROC" parameter ';'
    3   3 sentence -> "MACRO" argument ';'
    4   3 sentence -> "MACRO" parameter '.'
    5   4 argument -> "VARIABLE"
    6   5 parameter -> "VARIABLE"

Terminaux, suivis des règles où ils apparaissent

$ (0) 0
'.' (46) 1 4
';' (59) 2 3
error (256)
"PROC" (258) 1 2
"MACRO" (259) 3 4
"VARIABLE" (260) 5 6

Non-terminaux, suivis des règles où ils apparaissent

$axiom (8)
    à gauche: 0
sentence (9)
```

```
    à gauche: 1 2 3 4, à droite: 0
argument (10)
    à gauche: 5, à droite: 1 3
parameter (11)
    à gauche: 6, à droite: 2 4

état 0
    $axiom  ->  . sentence $   (règle 0)

    "PROC"      décalage et aller à l'état 1
    "MACRO"     décalage et aller à l'état 2

    sentence    aller à l'état 3

état 1
    sentence  ->  "PROC" . argument '.'   (règle 1)
    sentence  ->  "PROC" . parameter ';'   (règle 2)

    "VARIABLE"  décalage et aller à l'état 4

    argument    aller à l'état 5
    parameter   aller à l'état 6

état 2
    sentence  ->  "MACRO" . argument ';'   (règle 3)
    sentence  ->  "MACRO" . parameter '.'   (règle 4)

    "VARIABLE"  décalage et aller à l'état 4

    argument    aller à l'état 7
    parameter   aller à l'état 8

état 3
    $axiom  ->  sentence . $   (règle 0)

    $           décalage et aller à l'état 9

état 4
    argument  ->  "VARIABLE" .   (règle 5)
    parameter  ->  "VARIABLE" .   (règle 6)

    '.'         réduction par la règle 5 (argument)
    '.'         [réduction par la règle 6 (parameter)
    ';'         réduction par la règle 5 (argument)
    ';'         [réduction par la règle 6 (parameter)
    $défaut     réduction par la règle 5 (argument)

état 5
    sentence  ->  "PROC" argument . '.'   (règle 1)

    '.'         décalage et aller à l'état 10

état 6
    sentence  ->  "PROC" parameter . ';'   (règle 2)
```

```
    ';'         décalage et aller à l'état 11

état 7
    sentence  ->  "MACRO" argument . ';'   (règle 3)

    ';'         décalage et aller à l'état 12

état 8
    sentence  ->  "MACRO" parameter . '.'   (règle 4)

    '.'         décalage et aller à l'état 13

état 9
    $axiom  ->  sentence $ .   (règle 0)

    $défaut     accepter

état 10
    sentence  ->  "PROC" argument '.' .   (règle 1)

    $défaut     réduction par la règle 1 (sentence)

état 11
    sentence  ->  "PROC" parameter ';' .   (règle 2)

    $défaut     réduction par la règle 2 (sentence)

état 12
    sentence  ->  "MACRO" argument ';' .   (règle 3)

    $défaut     réduction par la règle 3 (sentence)

état 13
    sentence  ->  "MACRO" parameter '.' .   (règle 4)

    $défaut     réduction par la règle 4 (sentence)
```

Les actions entre crochets sont celles qui, du fait d'un conflit, ne seront pas retenues.