DEUG MIAS2. Institut Galilée
Option : Langages formels et compilation. 1999–2000
TD : Feuille 5.

Le but de cette série d'exercices est d'illustrer les notions mises en cause par le projet.

• L'automate d'analyse $LALR(1)$ de la grammaire de *Galileo* comporte 76 états : vous ne pourrez donc pas vraiment l'utiliser et il faudra faire appel à un outil dont nos machines ne disposent pas : la ¡¡compréhension¿¿ du processus d'analyse syntaxique de type LR. L'ordre dans lequel interviennent les règles, lors d'une analyse, est essentiel (et c'est peut être la principale difficulté du projet !). Pour cela, vous pouvez commencer par construire l'arbre d'analyse, puis considérer les règles dans l'ordre convenable (lorsque vous aurez l'habitude, vous pourrez faire de longues analyses sans vous appuyer sur un arbre).

• De même, il est difficile d'utiliser concrètement la pile d'analyse. Plutt que cette pile, il sera pratique :

- d'indexer chaque occurrence des éléments du vocabulaire au fur et à mesure qu'ils apparaissent dans l'analyse, par exemple, une réduction par la règle 22, pourra se présenter sous la forme

  `ES5` → `ES1 - T4`,

- de noter les éléments d'une production par les symboles indexés ci–dessus, au lieu d'utiliser les `$i` de Yacc (dans l'exemple précédent, on notera : `ES5` pour `$$` , `ES1` pour `$1` et `T4` pour `$3`),
- d'observer que dans la notation `$i` de Yacc, `i` détermine une position dans la pile d'analyse et qu'une affectation `$i := a` correspond au rangement de la valeur de `a` à cette position. Avec nos conventions, vous pourrez noter, par exemple `T4 := a` pour `$3 := a`, mais vous devrez vous méfier des ambiguïtés que cela comporte.

• Le P–code est produit au fur et à mesure : lorsque la réduction par une règle permet d'émettre une (ou plusieurs) P–instruction(s), celle(s)–ci vien(nen)t se placer à la suite de celles qui sont déjà écrites (le cas des P–instructions de saut (`UJP p, FJP p et TJP p`) dont on ne connat pas toujours la destination `p` au moment où on les écrit, sera vu en détail). La suite des P–instructions émises lors d'une analyse `Xi` → … → … est appelée ¡¡*le* P*–code de* `Xi`¿¿ (ou de l'analyse en question).

Exercice 1. Les expressions.

Soit `Expr` une expression dont la valeur est `Val(Expr)` et soit `E` → … → `Expr` la dérivation de `Expr`.
On veut vérifier que l'effet global de l'exécution du P–code de `E` est l'empilement de `Val(Expr)` ; plus précisément, si `SP` est l'indice du sommet de `PILE` juste avant l'exécution, alors, l'exécution peut être résumé par

```
SP := SP + 1 PILE[SP] := Val(Expr).
```

Pour cela, calculez le P–code de l'analyse de `12 + 5 * 10` et exécutez–le.

Nous verrons d'autres exemples par la suite.

Exercice 2. La déclaration des variables.

Pendant la partie ¡¡déclaration¿¿, chaque identificateur est rangé dans la table et la place nécessaire pour contenir les données correspondantes est réservée au début de `PILE`. Par la suite, chaque identificateur sera caractérisé par son ¡¡*déplacement*¿¿ c'est–à–dire par la distance à parcourir, en partant de la base de `PILE`, pour atteindre le début des données qu'il représente.

Calculez la table et le P–code des déclarations suivantes, et exécutez–le.

```
variables
    x : entier
    y : entier
    t : tableau[5]
fin
```

(Ces déclarations et la table correspondante nous serviront dans toute la suite.)

Exercice 3. Les variables et les affectations.

Les variables sont définies par les règles 16 et 17. Une variable est caractérisée par une adresse, c'est–à–dire un déplacement.
L'exécution du P–code d'une variable a pour effet global d'empiler son adresse.
On veut vérifier que l'exécution d'une instruction d'affectation (règle 10) a pour effet global de mettre la valeur affectée à l'adresse de la variable.

Pour cela, calculez le P–code des deux affectations suivantes, et exécutez–les.

- `x := 3`
- `y := 12 + 5 * 10`
- `t[4] := 12 + 5 * 10`

Exercice 4. Résumons.

L'exécution du P–code d'une expression définie par l'une des règles 28 et 30 a pour effet d'empiler la valeur (s'il y en a une!) qui se trouve à l'adresse de la variable correspondante.
Calculez et exécutez le P–code des affectations suivantes :

- `t[x] := y + x * t[4]`
- `t[x - 1] := ( t[3] * t[4] ) * 2`

Exercice 5. Les instructions conditionnelles.

La syntaxe des instructions conditionnelles simples est définie par la règle

- 11 : $I \to$ **si** E **alors** Li **fin**

mais la règle qui a été retenue pour l'analyse syntaxique

- 11 : $I \to$ **si** E $Malors$ Li **fin**

comporte un non–terminal de ¡¡marquage¿¿ et une règle de production

- 33 : $Malors \to$ **alors**

rétablissant la syntaxe voulue.

P-code de E
P-code de Li
? ?

Sans marquage.

	P-code de E
a :	FJP ?
	P-code de Li

Avec marquage.

Les figures montrent l'aspect des P–codes produits par la règle initiale et par la règle ¡¡marquée¿¿. Avec cette dernière, le P–code de $Malors$ vient se placer entre ceux de E et de Li : l'adresse de destination de l'instruction de saut `FJP` n'est pas encore connue mais on enregistre son adresse `a` dans une ¡¡liste de sorties¿¿ pour pouvoir compléter cette P–instruction lorsque ce sera possible. Remarquez que Li elle–même peut comporter des sorties : elles seront nécessairement dirigées vers la même adresse.

Calculez et exécutez le P–code de la liste d'instructions suivante :

```
si x < y alors
    si x < t[3] alors x := 0 fin ;
    y := t[4]
fin ;
ecrire( x + y)
```

Remarque : le marquage de la règle 9, relative aux listes d'instructions, correspond au fait que les instructions d'une telle liste doivent être exécutées séquentiellement.

Exercice 6.

Appliquez la méthode précédente pour traiter la boucle (qui n'est pas dans le langage qui vous est proposé) dont la syntaxe est $I \rightarrow$ **repeter** Li **jusqua** E et dont la signification vous est bien connue.

Calculez et exécutez le P–code de la liste d'instructions suivante :

```
x := -1 ;
y := 1 ;
repeter
    x := x + 1 ;
    t[x] := y + 2 ;
    si x > 1 alors y := y * (x - 1 ) fin
jusqua x = 5
```

Exercice 7. Evaluation partielle des opérations booléennes.

On rappelle que, pour *Galileo*, toute expression peut s'interpréter comme une expression booléenne : fausse si elle est égale à `0` et vraie sinon.

Le projet propose des P–instructions capables de calculer la conjonction et la disjonction. Il y a cependant une autre façon de procéder. Par exemple on sait que :

- si `A` est vraie alors `(A et B) = B`
- si `A` est fausse alors `(A et B)` est fausse : dans ce cas, il n'est pas nécessaire de calculer la valeur de `B` pour conclure.

• Appliquez la méthode de marquage à la règle $E \to E$ **et** E pour exploiter cette remarque.

• Transposez ce qui précède au cas de la disjonction.

Vous pouvez utiliser une P–instruction `TJP` analogue à `FJP` pour laquelle le faux est remplacé par le vrai.