

# Correction du Partiel de Compilation

## 1 Hiérarchie de Chomsky

### 1.1 Langages

Pour chacun des langages suivants, préciser son type dans la hiérarchie de Chomsky (son rang, e.g., 3, et son nom, e.g., langage régulier). Proposer une grammaire non ambiguë qui représente ce langage (les grammaires longues peuvent être esquissées et se terminer par "...").

1. L'ensemble des logins de l'ENST.

**Correction:** Il s'agit d'un langage fini, rang 4 : les logins sont composés au plus de 8 caractères. Une grammaire longue mais finie est :

```
S -> yvon
S -> dax
S -> demaille
...
S -> peyrade
```

2. L'ensemble des nombres binaires (écrits de "0" et de "1").

**Correction:** Langage infini, mais trivialement régulier —type 3— puisque représenté par l'expression régulière  $(0|1)^+$ . On peut par exemple en donner une grammaire linéaire à droite :

```
B -> 0S    B -> 1S
B -> 0     B -> 1
```

3. L'ensemble des sommes ("+") de nombres binaires.

**Correction:** Ça reste régulier, type 3, puisque représenté par l'expression régulière "num(+num)\*" où num représente une expression régulière représentant les entiers binaires, comme par exemple celle de la question précédente. On obtient  $(0|1)^+(+(0|1)^+)^*$ .

Un grammaire hors-contexte et non régulière serait :

```
S -> B + S
S -> B
```

avec B comme dans la question précédente. Cette grammaire est un fragment de la grammaire classique de l'arithmétique, bien connue pour ne pas être ambiguë.

4. L'ensemble des sommes de nombres binaires avec parenthèses ("(", ")").

**Correction:** Il est bien connu qu'on est sorti des langages réguliers, pour tomber dans les langages hors-contexte —type 2. En effet, une grammaire hors-contexte serait :

```
S -> B + S
S -> B
S -> (S)
```

## 1.2 Grammaires

Pour chacune des grammaires suivantes, (i) préciser son type dans la hiérarchie de Chomsky, (ii) si elle est ambiguë, (iii) le langage qu'elle engendre, (iv) le type du langage dans la hiérarchie. Justifier votre réponse.

1.  $P ::= P \text{ inst } ';' \mid /* \text{ empty } */$

**Correction:** Linéaire à gauche, donc régulière. Elle est non ambiguë, et engendre une suite de zéro ou plusieurs *instr terminés* par des `;`. Ce langage est infini, régulier (puisqu'engendré par une grammaire régulière) : type 3.

2.  $P ::= P1 \mid /* \text{ empty } */$   
 $P1 ::= P1 ';' \text{ inst } \mid \text{ inst}$

**Correction:** Linéaire à gauche, donc régulière. Elle est non ambiguë.  $P1$  engendre une liste de une ou plusieurs *instr séparés* par des `;`. Donc, cette grammaire engendre une liste de zéro ou plusieurs *instr séparés* par des `;`. Ce langage est infini, régulier (puisqu'engendré par une grammaire régulière) : type 3.

3.  $P ::= P1 \mid /* \text{ empty } */$   
 $P1 ::= P1 ';' P1 \mid \text{ inst}$

**Correction:** Cette grammaire est très visiblement une version ambiguë de la grammaire précédente. On pourrait dire que dans la grammaire l'opérateur `;` est associatif à gauche, ici il est associatif à droite et à gauche, i.e., une phrase comme `inst ; inst ; inst` peut se lire comme `(inst ; inst) ; inst` ou `inst ; (inst ; inst)`. Le langage, lui, reste évidemment de type 3.

4.  $S ::= P$   
 $P ::= p P Q R$   
 $P ::= p q R$   
 $R Q ::= Q R$   
 $q Q ::= q q$   
 $q R ::= q r$   
 $r R ::= r r$

**Correction:** Cette grammaire est visiblement monotone, non hors contexte. Bien qu'il ne soit pas simple de le montrer formellement, une "exécution" de cette grammaire à la main montre qu'elle n'est pas ambiguë. On reconnaît l'exemple de grammaire engendrant  $a^n b^n c^n$ , i.e., le langage des mots commençant par un certain nombre (non nul) de `a`, puis d'autant de `b`, et enfin autant de `c`. Mais avec `p`, `q` et `r`. Ce langage est bien connu comme l'exemple type des langages hors-contextes, comme vu en cours.

Les non terminaux sont écrits en majuscules, les terminaux en minuscules ou entre guillemets, et `/* empty */` désigne  $\epsilon$ , le mot vide.

## 2 Parsage LALR(1)

Considérons la grammaire suivante, écrite selon la syntaxe Bison.

```
%%  
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 Bison/Yacc échoue avec deux conflits réduction/réduction. Le rapport de génération de l'automate contient :

```

State 4 contains 2 reduce/reduce conflicts.
[...]
State 0 $axiom -> . sentence $ (rule 0)

        "PROC"      shift, and go to state 1
        "MACRO"     shift, and go to state 2
        sentence    go to state 3

State 1 sentence -> "PROC" . argument '.' (rule 1)
        sentence -> "PROC" . parameter ';' (rule 2)

        "VARIABLE" shift, and go to state 4
        argument    go to state 5
        parameter   go to state 6

State 2 sentence -> "MACRO" . argument ';' (rule 3)
        sentence -> "MACRO" . parameter '.' (rule 4)

        "VARIABLE" shift, and go to state 4
        argument    go to state 7
        parameter   go to state 8

State 3 $axiom -> sentence . $ (rule 0)

        $          shift, and go to state 9

State 4 argument -> "VARIABLE" . (rule 5)
        parameter -> "VARIABLE" . (rule 6)

        '.'        reduce using rule 5 (argument)
        '.'        [reduce using rule 6 (parameter)]
        ';'        reduce using rule 5 (argument)
        ';'        [reduce using rule 6 (parameter)]
        $default   reduce using rule 5 (argument)
[...]

```

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

6. 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.

### 3 Parsage LL

Écrire la grammaire naïve de l'arithmétique avec les terminaux "1" (le seul nombre), "-" la soustraction binaire, "/" la division, "(" et ")" les parenthèses. En plusieurs étapes, en faire

une grammaire adaptée au passage LL(1). Expliquer pourquoi l'implémentation LL(1) de cette grammaire sera sûrement plus performante que le passage par automate LALR(1).