# 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.
2. L'ensemble des nombres binaires (écrits de "0" et de "1").
3. L'ensemble des sommes ("+") de nombres binaires.
4. L'ensemble des sommes de nombres binaires avec parenthèses ("(", ")").

### 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 inst ';' | /* empty */`
2. ```
   P  ::= P1 | /* empty */
   P1 ::= P1 ';' inst | inst
   ```
3. ```
   P  ::= P1 | /* empty */
   P1 ::= P1 ';' P1 | inst
   ```
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
   ```

Les non terminaux sont écrits en majuscules, les terminaux en minuscules ou entre guillemets, et `/* empty */` désigne $\varepsilon$, 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 parsage LL(1). Expliquer pourquoi l'implémentation LL(1) de cette grammaire sera sûrement plus performante que le parsage par automate LALR(1).