TD COMPILATION — FEUILLE 2

Grammaire hors-contexte et analyse LL(1)

Exercice 1. On s'intéresse aux expressions booléennes de deux langages de programmation, Ada et Pascal. On se limitera au vocabulaire terminal suivant : { **true**, **false**, **not**, **and**, **or**, **(**, **)** }.

▷ **Question 1** *En* Pascal, *l'opérateur* **and** *est plus prioritaire que l'opérateur* **or**. *D'autre part* **and** *et* **or** *sont associatifs. Par exemple l'expression* **true and false or true** *est correcte et s'évalue à la valeur vraie. Donner une grammaire non ambigüe de ce langage et définir par récurrence l'évaluation des expressions.*

▷ **Question 2** *En* Ada *les opérateurs* **and** *et* **or** *ont même priorité mais ne peuvent être combinés l'un avec l'autre. D'autre part* **and** *et* **or** *sont associatifs. Par exemple l'expression* **true and false or true** *est incorrecte mais les expressions suivantes sont permises :* **true and (false or true)**, **(true and false) or true**, **true and false and true**. *Même question.*

Exercice 2.

▷ **Question 1** *Lesquelles, parmi les grammaires suivantes, sont LL(1) ? Que peut-on dire des langages ?*

1. $S \to ABc \qquad A \to a \mid \epsilon \qquad B \to bS \mid \epsilon$
2. $S \to BbB \qquad B \to bS \mid \epsilon$

▷ **Question 2** *Soit le langage* $L = \{1^n 0^m \mid 0 \leq n \leq m\}$. *Donner une grammaire hors-contexte LL(1) engendrant* L.

Exercice 3. Donner des grammaires LL(1) engendrant les mêmes langages que les grammaires ci-dessous. On justifiera le caractère LL(1) des grammaires proposées.

▷ **Question 1**

$$
\begin{array}{lcl}
place & \to & place\ \mathbf{.}\ \mathbf{idf} \mid place\ \mathbf{(}\ suite\_exp\ \mathbf{)} \mid \mathbf{idf} \\
suite\_exp & \to & exp \mid exp\ \mathbf{,}\ suite\_exp \\
exp & \to & \mathbf{num} \mid place
\end{array}
$$

avec $VT_1 = \{\ \mathbf{.}, \mathbf{(}, \mathbf{)}, \mathbf{idf}, \mathbf{,}, \mathbf{num}\ \}$

▷ **Question 2**

$$
\begin{array}{lcl}
I & \to & place\ \mathbf{:=}\ exp \mid \mathbf{case}\ exp\ \mathbf{of}\ LC\ \mathbf{end\ case\,;} \\
LC & \to & LCD\ LO \\
LO & \to & \mathbf{when\ others} \Rightarrow I \mid \epsilon \\
LCD & \to & C\ LCD \mid C \\
C & \to & \mathbf{when}\ exp \Rightarrow I
\end{array}
$$

avec $VT_2 = VT_1 \cup \{\ \mathbf{:=}, \mathbf{case}, \mathbf{of}, \mathbf{end}, \mathbf{when}, \mathbf{;}, \mathbf{others}, \Rightarrow\ \}$. *Les non-terminaux place et exp sont définis comme précédemment.*

Exercice 4.

▷ **Question 1** *Montrer que tout langage régulier est LL(1).*

▷ **Question 2** *Proposer une version itérative de la procédure d'analyse LL(1) pour un non-terminal défini par* $A \rightarrow \omega A \mid \alpha$.

Exercice 5.

▷ **Question 1** *En ne considérant que des opérateurs arithmétiques binaires, donner une grammaire hors-contexte LL(1) des expressions postfixées, définies sur le vocabulaire* $VT_1 = \{$ **num**, **+**, **-**, ***** $\}$.

▷ **Question 2** *Écrire un programme qui reconnaît les expressions postfixées. On utilisera une fonction* `lire_mot` *qui retourne un élément du type énuméré* { **num**, **add**, **moins**, **mult**, **eof** }. *Le terminal* **eof** *représente la fin de fichier.*

▷ **Question 3** *Compléter l'analyseur afin d'ajouter, en même temps que l'analyse, l'évaluation des expressions. On supposera que dans le cas de la reconnaissance des constantes entières la variable* **val** *contient la valeur de cette constante. La variable* **val** *est mise à jour par la fonction* `lire_mot`.

▷ **Question 4** *Reprendre les questions précédentes pour les expressions infixées avec parenthèses* ($VT_2 = VT_1 \cup \{$ **(**, **)** $\}$).