TD2 : Grammaires et analyse syntaxique

Exercice 1

On a vu que le langage des palindromes (les mots qui peuvent se lire de gauche à droite et de droite à gauche) n'est pas régulier. Donner une grammaire qui le reconnaît sur l'alphabet $\{a, b, c\}$.

Exercice 2

Décrire le langage reconnu par la grammaire suivante :

$$S \to aSbS \mid bSaS \mid \varepsilon$$

Exercice 3

On considère la grammaire suivante :

$$S \to AB \mid Ca \qquad A \to aAb \mid \varepsilon \qquad B \to bB \mid b \qquad C \to cC \mid d$$

1. Calculer Premier et Suivant pour cette grammaire.
2. Donner la table d'analyse syntaxique de cette grammaire.
3. La grammaire est-elle LL(1) ?

Exercice 4

La *notation polonaise inversée* (NPI) est une manière d'écrire des expressions arithmétiques sans ambiguïté et sans parenthèses. L'idée consiste à placer les opérateurs ($+$ et $\times$) après les deux opérandes. Par exemple, $1\,2\,+$ représente l'opération écrite habituellement $1 + 2$.

1. Évaluer $1\,2\,3 + 4 \times +$.
2. Donner un algorithme permettant d'évaluer une expression en NPI en la lisant une seule fois, de gauche à droite. *Indication : utiliser une pile.*
3. Donner une grammaire reconnaissant les expressions en NPI (on représentera les entiers par le terminal *id*).
4. En s'inspirant de la question 2, fournir un algorithme d'analyse syntaxique pour cette grammaire, c'est-à-dire qui, étant donnée une expression en NPI, construit un arbre de dérivation qui la génère.

Exercice 5

Dans le cadre de l'analyse ascendante, on essaye de construire un arbre de dérivation en partant de ses feuilles. Comme pour les expressions NPI, on va utiliser une pile ainsi que les deux opérations suivantes :

- *shift* : on empile la prochaine lettre du mot w
- *reduce* : on trouve $\alpha_1, \ldots, \alpha_n$ en haut de la pile et on les remplace par A quand on a une règle $A \to \alpha_1 \cdots \alpha_n$ dans la grammaire.

À chaque fois qu'on applique une opération *reduce*, on ajoute un noeud dans l'arbre de dérivation qu'on construit. Ce noeud sera étiqueté par A, et aura pour fils $\alpha_1, \ldots, \alpha_n$.

1. Reformuler l'algorithme de l'exercice précédent en utilisant les opérations shift et reduce.

On considère le mot $a + a \times a$, accepté par grammaire G_1 suivante :

$$E \to E + T \mid T \qquad T \to T \times F \mid F \qquad F \to (E) \mid id$$

2. On applique shift, reduce, reduce, shift. Que devient la pile ? Peut-on finir l'analyse ?
3. On applique shift, reduce, reduce, reduce, shift, shift, reduce, reduce, reduce. Que devient la pile ? Peut-on finir l'analyse ?

Pour savoir quelle opération utiliser, on va construire l'*automate des items*. Un *item* est un objet de la forme $A \to \alpha_1 \cdots \alpha_p \bullet \alpha_{p+1} \cdots \alpha_n$. Il signifie qu'on a déjà réussi à reconstituer $\alpha_1 \cdots \alpha_p$ et qu'il reste à reconstruire $\alpha_{p+1} \cdots \alpha_n$ pour pouvoir appliquer une opération *reduce* avec la règle $A \to \alpha_1 \cdots \alpha_p$.

On considère la grammaire G_2 suivante

$$S \to E\$ \qquad E \to (E + E) \mid (E \times E) \mid id$$

4. Quels sont les items de cette grammaire ?

On va construire un automate dont les états sont les items, et tel que :

- Les états initiaux sont les items de la forme $A \to \bullet\alpha_1 \cdots \alpha_n$ (*on commence la lecture de* $\alpha_1 \cdots \alpha_n$) où A est l'axiome.
- Les états finaux sont les items de la forme $A \to \alpha_1 \cdots \alpha_n\bullet$ (*on a fini de reconnaître* $\alpha_1 \cdots \alpha_n$).
- Pour chaque terminal ou non-terminal a, on a une transition étiquetée par a de $A \to u \bullet av$ vers $A \to ua \bullet v$.
- Pour chaque non-terminal B, on a un transition étiquetée par ε de $A \to u \bullet Bv$ vers tous les items de la forme $B \to [\ldots]$.

5. Construire l'automate des items.
6. Éliminer les transitions ε comme vu en cours.
7. Déterminiser l'automate.

L'automate des items permet de décider quelle opération utiliser : on utilise *shift* quand on parcourt une transition de l'automate, et on utilise *reduce* quand on arrive dans un état final.

8. Effectuer l'analyse du mot $((id + id) \times id)\$$.

On dit que la grammaire est LR(0) quand l'automate des items n'a pas de conflit:

- conflit shift-reduce: quand il y a une transition qui part d'un état final, on a le choix entre suivre cette transition (shift) ou appliquer reduce.

- conflit reduce-reduce: quand un état final contient deux items finaux, on a le choix de quelle règle réduire.

Ici, il n'y a pas de conflit, donc G_2 est LR(0).

9. G_1 est-elle LR(0) ?

Exercice 6

On considère le langage $L = \{a^n b^n c^n, n > 0\}$. Le but de cet exercice est de montrer qu'il ne peut pas être reconnu par une grammaire. Supposons par l'absurde qu'il existe une grammaire reconnaissant L. Notons N son nombre de non-terminaux, et d le nombre maximum de symboles à droite d'une règle de production.

1. Montrer que si un mot $\alpha \in L$ a un arbre de dérivation de hauteur h, alors $|\alpha| \leqslant d^h$.
2. On pose $p = d^{N+1}$ et $\alpha = a^p b^p c^p$. Montrer que tout arbre de dérivation pour w contient une branche contenant deux occurrences du même non-terminal.
3. En déduire qu'on peut décomposer α en $uvwxy$ de sorte que $vx \neq \varepsilon$ et $uv^k wx^k y \in L$ pour tout entier k.
4. En déduire une contradiction, et que L ne peut pas être reconnu par une grammaire.
5. Montrer que $\{a^n b^n c^m, n, m > 0\}$ et $\{a^m b^n c^n, m, n > 0\}$ sont reconnus par des grammaires et en déduire que l'intersection de deux langages reconnus par des grammaires n'est pas forcément reconnue par une grammaire.