TD01 - GRAMMAIRES HORS CONTEXTE

Conventions

Pour répondre aux questions ci-dessous, respectez les conventions suivantes :

- Utilisez des lettres *MAJUSCULES* pour les symboles non terminaux, et des lettres `minuscules` ou symboles spéciaux pour les symboles terminaux.
- Chaque règle de la grammaire aura exactement un symbole non terminal en partie gauche.
- Vous pouvez écrire $A \to \alpha \mid \beta$ si des règles $A \to \alpha$ et $A \to \beta$ partagent leur partie gauche.
- La flèche simple $\to$ dénote une règle de grammaire, la flèche double $\Rightarrow$ dénote une dérivation.
- L'axiome est le symbole qui correspond à la partie gauche de la première règle.

1. Listes & Séquences

Exercice 1. *Nombres*

(1) Écrire deux grammaires permettant de générer les nombres entiers positifs, par exemple `14592` ou encore `543`.

(2) Donner les dérivations gauche et droite du mot `543` pour chacune des deux grammaires.

(3) Dessiner l'arbre de dérivation (ou les arbres de dérivations) du mot `543` pour chacune des deux grammaires.

Exercice 2. *Listes avec séparateur*

(1) Écrire une grammaire permettant de générer des listes, éventuellement vides ε, de chiffres séparés par deux-points, par exemple `1:4:5:9:2` ou encore `5:4:3`.

(2) Donner les dérivations gauche et droite du mot `5:4:3`.

(3) Dessiner l'arbre de dérivation (ou les arbres de dérivations) du mot `5:4:3`.

Exercice 3. *Listes avec séparateur et parenthèses*

(1) Écrire une grammaire permettant de générer des listes, éventuellement vides `()`, de chiffres séparés par deux-points et encadrés par des parenthèses, par exemple `(1:4:5:9:2)` ou encore `(5:4:3)`.

(2) Dessiner l'arbre de dérivation (ou les arbres de dérivations) du mot `(5:4:3)`.

Exercice 4. *Listes imbriquées*

(1) Écrire une grammaire permettant de générer des listes, éventuellement vides `()`, de chiffres séparés par deux-points et encadrés par des parenthèses. Les éléments de ces listes sont soit des chiffres, soit des listes. Par exemple, `(1:(2:(3:4)))` ou encore `(5:(4:3):())`.

(2) Donner les dérivations gauche et droite du mot `(5:(4:3):())`.

(3) Dessiner l'arbre de dérivation (ou les arbres de dérivations) du mot `(5:(4:3):())`.

2. Expressions

Exercice 5. *Associativité*

(1) Écrire une grammaire, la plus simple possible, éventuellement ambiguë, permettant de générer des expressions composées de chiffres et de l'opérateur de soustraction, par exemple `1-2-3`.

(2) Dessiner les arbres syntaxiques du mot `2-1-1`. Parmi les arbres générés, lequel vous semble correct ? Autrement dit, lequel de ces arbres correspond à l'interprétation habituelle de la soustraction associative à gauche $(2 - 1) - 1 = 0$?

(3) On souhaite empêcher la génération d'arbres incorrects de l'exemple précédent, qui pourraient être faussement interprétés comme $2 - (1 - 1) = 2$. Comment peut-on modifier la grammaire pour y arriver ?

Exercice 6. *Priorité*

(1) Écrire une grammaire, la plus simple possible (ambiguë), permettant de générer des expressions composées de chiffres et des opérateurs d'addition et de multiplication, par exemple `1+2+3` ou `3+2*6`.

(2) Dessiner les arbres syntaxiques du mot `3+2*6`. Parmi les arbres générés, lequel vous semble correct ? Autrement dit, lequel de ces arbres correspond à l'interprétation habituelle de la multiplication prioritaire par rapport à l'addition $3 + (2 \times 6) = 15$?

(3) On souhaite empêcher la génération d'arbres incorrects de l'exemple précédent, qui pourraient être faussement interprétés comme $(3 + 2) \times 6 = 30$. Comment peut-on modifier la grammaire pour y arriver ?

3. Ambigüité

Définition. *Soit $\mathcal{L}$ un langage hors-contexte. Soit G une grammaire définissant $\mathcal{L}$. G est dite* ambigüe *s'il existe un mot m de $\mathcal{L}$ tel que m admet plus d'un arbre de dérivation G.*

N.B. Le problème de l'ambigüité en général d'une grammaire est indécidable, ce qui signifie qu'il n'est pas possible d'élaborer un algorithme qui soit capable de dire, après un temps fini, si une grammaire qu'on lui soumet est ambigüe ou non.

Exercice 7.
Soit la grammaire :

$$P \longrightarrow \varepsilon \mid (\ P \ ) \mid P \ P$$

(1) Montrez qu'elle est ambigüe.

(2) Proposez une grammaire équivalente qui ne l'est pas.

Exercice 8.
Soit la grammaire :

$$\begin{array}{lcl} INSTR & \rightarrow & \texttt{si}\ EXPR\ \texttt{alors}\ INSTR \\ & \mid & \texttt{si}\ EXPR\ \texttt{alors}\ INSTR\ \texttt{sinon}\ INSTR \\ & \mid & \texttt{i} \\ EXPR & \rightarrow & \texttt{e} \end{array}$$

(1) Dessiner les arbres de dérivation du mot `si e alors si e alors i sinon i`.

Les langages de programmation choisissent de faire correspondre chaque `sinon` avec le `si` non clos le plus proche. L'idée est que l'instruction apparaissant entre un `alors` et un `sinon` doit être *close* : elle ne doit pas se terminer par un `alors` ouvert ou non clos. Une instruction close est soit une instruction `si alors sinon` ne contenant aucune instruction ouverte, soit tout autre sorte d'instruction non conditionnelle (le `i` dans notre grammaire).

(2) Modifier la grammaire précédent afin d'implémenter la règle du rattachement du `sinon` avec le `si` non clos le plus proche.

(3) Dessiner l'arbre de dérivation du mot `si e alors si e alors i sinon i`.

4. Autres langages

Exercice 9. *Langage $\mathcal{L}_1 = \{a^*b\}$*
Soit l'alphabet $\Sigma = \{a, b\}$ et le langage $\mathcal{L}_1 = \{a^*b\}$ sur Σ. Écrivez une grammaire de $\mathcal{L}_1$.

Exercice 10. *Langage $\mathcal{L}_2 = \{a^n b^n \mid n \in \mathbb{N}\}$*
Soit l'alphabet $\Sigma = \{a, b\}$ et le langage $\mathcal{L}_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ sur Σ. Écrivez une grammaire de $\mathcal{L}_2$.

Exercice 11. *Langage $\mathcal{L}_3 = \{a^n b^p \mid n > p$ où $(n, p) \in \mathbb{N} \times \mathbb{N}\}$*
Soit l'alphabet $\Sigma = \{a, b\}$ et le langage $\mathcal{L}_3 = \{a^n b^p \mid n > p$ où $(n, p) \in \mathbb{N} \times \mathbb{N}\}$ sur Σ. Écrivez une grammaire de $\mathcal{L}_3$.