TD 7 : Grammaires algébriques

Exercice 1 (Langage local d'arbres)**.** Soit Σ un alphabet et t un terme de $T(\Sigma)$. On note $\mathsf{r}(t)$ pour le symbole qui étiquette la racine de t et $b(t)$ l'ensemble des *branchements* de t, définis par

$$\mathsf{r}(a) = a \qquad \mathsf{b}(a) = \emptyset \qquad \text{si } a \in \Sigma_0$$

$$\mathsf{r}(f(t_1, \ldots, t_n)) = f \quad \mathsf{b}(f(t_1, \ldots, t_n)) = \{f(\mathsf{r}(t_1), \ldots, \mathsf{r}(t_n))\} \cup \bigcup_{i=1}^{n} \mathsf{b}(t_i) \qquad \text{si } f \in \Sigma_n$$

Par exemple $\mathsf{b}(\ f(g(a), f(a,b))\ ) = \{f(g,f), g(a), f(a,b)\}$.

Un langage d'arbres L est *local* si et seulement s'il existe deux ensembles $R \subseteq \Sigma$ de symboles racines et $B \subseteq \mathsf{b}(T(\Sigma))$ de branchements tels que $t \in L$ ssi $\mathsf{r}(t) \in R$ et $\mathsf{b}(t) \subseteq B$. Posons

$$L(R, B) = \{t \in T(\Sigma \mid \mathsf{r}(t) \in R \text{ et } \mathsf{b}(t) \subseteq B\} \ ;$$

on a alors qu'un langage L est local si et seulement si $L = L(\mathsf{r}(L), \mathsf{b}(L))$.

1. Montrer que le langage $\{\ f(g(a), g(b))\ \}$ n'est pas local.
2. Montrer que tout langage local d'arbres est reconnaissable.
3. Soit $\Sigma = \Sigma_0 \uplus \Sigma_{>0}$. Montrer qu'un langage d'arbres sur Σ est local avec $|R| = 1$ si et seulement s'il est l'ensemble des arbres de dérivation d'une grammaire algébrique.
4. Montrer que tout langage reconnaissable d'arbres est l'image d'un langage local d'arbres par un morphisme d'arbre alphabétique. En déduire que le langage $\mathsf{Fr}(L)$ des frontières d'un langage reconnaissable d'arbres est algébrique.

Exercice 2 (Lemme d'Ogden)**.**

1. Montrer que le langage $\{a^n b^n c^m d^m \mid n, m \geq 0\}$ est algébrique mais pas linéaire. On pourra considérer le mot $a^n b^n c^n d^n$ avec tous les d distingués.
2. Montrer que le langage $\{a^n b^n c^m \mid n, m \geq 0\} \cup \{a^n b^m c^m \mid n, m \geq 0\}$ est linéaire et inhéremment ambigu. On pourra appliquer le lemme aux mots $a^{n+n!} b^n c^n$ puis $a^n b^n c^{n+n!}$ et exhiber deux arbres de dérivations différents pour le mot $a^{n+n!} b^{n+n!} c^{n+n!}$.

Exercice 3 (Langages de programmation)**.**

1. La première utilisation des grammaires algébriques pour décrire la syntaxe d'un langage de programmation a été faite pour le langage Algol 60. Néanmoins, la modélisation par une grammaire algébrique seule n'est pas possible : le programme suivant

```
begin
  real x;
  y := z
end
```

n'est *sémantiquement* correct que si les identifiants `x`, `y` et `z` coïncident. Cependant un identifiant ALGOL 60 est une chaîne arbitrairement longue du langage rationnel $l(l+c)^*$ où l désigne n'importe quelle lettre majuscule ou minuscule de l'alphabet latin, et c n'importe quel chiffre de 0 à 9.

Montrer que ce niveau de correction ne peut pas être assuré par une grammaire algébrique seule.

2. Voici un extrait de la grammaire d'ALGOL 60 :

$$\begin{aligned}
\langle \textit{statement} \rangle &\to \langle \textit{unconditional statement} \rangle \\
&\mid \langle \textit{conditional statement} \rangle \\
\langle \textit{unconditional statement} \rangle &\to \langle \textit{for statement} \rangle \\
\\
\langle \textit{conditional statement} \rangle &\to \langle \textit{if statement} \rangle \\
&\mid \langle \textit{if statement} \rangle \ \texttt{else} \ \langle \textit{statement} \rangle \\
\langle \textit{if statement} \rangle &\to \langle \textit{if clause} \rangle \ \langle \textit{unconditional statement} \rangle \\
\langle \textit{if clause} \rangle &\to \texttt{if} \ \langle \textit{boolean expression} \rangle \ \texttt{then}
\end{aligned}$$

Comme vous pouvez l'observer, les fondateurs d'ALGOL 60 ont pris des précautions pour éviter l'ambiguïté de la construction `if/then/else`, en interdisant totalement d'emboîter des instructions `if/then` ! Comment pourrait-on modifier cette grammaire (et du coup le langage généré) pour permettre la dérivation de la phrase suivante sans introduire d'ambiguïté ?

```
if a > 1 then if b > 2 then c := 0 else c := 1
```

3. La grammaire d'ALGOL 60 comportait aussi les règles

$$\begin{aligned}
\langle \textit{for statement} \rangle &\to \langle \textit{for clause} \rangle \ \langle \textit{statement} \rangle \\
\langle \textit{for clause} \rangle &\to \texttt{for} \ \langle \textit{variable} \rangle := \langle \textit{for list} \rangle \ \texttt{do} \\
\langle \textit{for list} \rangle &\to \langle \textit{for list element} \rangle \\
&\mid \langle \textit{for list} \rangle \ , \ \langle \textit{for list element} \rangle \\
\langle \textit{for list element} \rangle &\to \langle \textit{arithmetic expression} \rangle \\
&\mid \langle \textit{arithmetic expression} \rangle \ \texttt{step} \ \langle \textit{arithmetic expression} \rangle \\
&\quad \texttt{until} \ \langle \textit{arithmetic expression} \rangle
\end{aligned}$$

Montrer que l'extrait de grammaire constitué des deux fragments est ambigu. Proposer une grammaire *équivalente* non ambiguë.