TD 7 : Formes normales et complexité

Exercice 1 (Forme normale de Chomsky)**.** Une grammaire algébrique $G = \langle N, \Sigma, P, S\rangle$ de langage associé non vide est dite sous *forme normale de Chomsky* si toutes les productions de R sont de la forme $X \longrightarrow YZ$ ou $X \longrightarrow a$.

1. Soit G une grammaire algébrique propre. Montrer qu'il est possible de transformer G en une grammaire G_1 engendrant le même langage que G et telle que toutes les règles de G_1 sont de l'une des deux formes suivantes : $X \longrightarrow a$ et $X \longrightarrow X_1 \dots X_n, n \geq 2$ où $X, X_1 \dots X_n$ sont des non-terminaux et a est un terminal.
2. En déduire que toute grammaire algébrique n'engendrant pas le mot vide peut être transformée en une grammaire algébrique sous forme normale de Chomsky engendrant le même langage.
3. Mettre la grammaire suivante sous forme normale de Chomsky :

$$\begin{array}{lcl} S & \longrightarrow & bA|aB \\ A & \longrightarrow & bAA|aS|a \\ B & \longrightarrow & aBB|bS|b \end{array}$$

Exercice 2 (Théorème de Chomsky et Schützenberger)**.** Le théorème dit qu'un langage L est algébrique si et seulement s'il existe un entier n, un langage rationnel R et un morphisme π tels que $L = \pi(D_n^* \cap R)$, où D_n^* dénote l'ensemble des mots bien parenthésés sur un alphabet à n paires de parenthèses.

L'intuition derrière ce théorème est que l'on peut séparer les aspects de structure (le parenthésage, c'est-à-dire la structure d'arbre) et de contrôle (le langage rationnel) d'un langage algébrique – ce dont on verra une autre interprétation avec les automates à pile.

1. Soit G une grammaire algébrique sur Σ. Proposer une grammaire algébrique G', qui explicite la structure des dérivations de G au moyen d'un alphabet Σ_n de n sortes de parenthèses, telle que

$$L(G') \subseteq D_n^* \text{ et } L(G) = \pi(L(G'))$$

   avec π une projection de $\Sigma_n^* \to \Sigma^*$.

2. Il faut maintenant trouver un langage rationnel R tel que

$$L(G') = D_n^* \cap R\ .$$

   En fait, il existe même un langage *local* (cf. feuille de TD 1, exercice 5) qui remplit ce rôle. Proposer un tel langage.

3. Montrer qu'il existe un morphisme μ de $\Sigma_n \to \Sigma_2$ tel que

$$D_n^* = \mu^{-1}(D_2^*)\ .$$

   En déduire une autre formulation du théorème.

Exercice 3 (Complexité)**.** On définit la *taille* d'une grammaire algébrique $G = \langle N, \Sigma, P, S\rangle$ comme

$$|G| = \max(\sum_{A\to\alpha\in P} |A\alpha|, |N \cup \Sigma|) \ .$$

1. Une grammaire algébrique est sous *forme quadratique* si pour toute règle $A \to \alpha$ de P, $|\alpha| \leq 2$. Montrer que, pour toute grammaire algébrique G, on peut construire une grammaire algébrique G' équivalente sous forme quadratique en temps $O(|G|)$.
2. Un symbole non terminal A de N est *utile* s'il existe une dérivation de la forme

   $$S \Rightarrow^* \alpha A \beta \Rightarrow^* w$$

   avec α et β des chaînes de symboles dans $(N \uplus \Sigma)^*$ et w un chaîne de symboles dans Σ^*.

   Montrer que pour toute grammaire algébrique G, on peut calculer l'ensemble de ses symboles non terminaux utiles en temps $O(|G|)$. En déduire un algorithme pour construire une grammaire *réduite* équivalente à G en temps linéaire.
3. En déduire une borne de complexité du calcul d'une grammaire algébrique équivalente où les règles $A \to \alpha$ de P vérifient $|\alpha| \geq 1$ (sauf potentiellement pour une règle $S \to \varepsilon$, mais alors on demande aussi $\alpha \in (\Sigma \cup N\backslash\{S\})^*$).
4. Quelle borne de complexité pouvez-vous donner pour la suppression des règles de la forme $A \to B$ avec A et B dans N ? Pour la mise sous forme normale de Chomsky ?

Exercice 4 (Intersection avec un langage reconnaissable et algorithme de Cocke, Kasami et Younger)**.**

1. Soit G une grammaire algébrique et A un automate fini. Contruire G' algébrique telle que $L(G') = L(G) \cap L(A)$.
2. Modifier la construction pour calculer *au vol* si $L(G') = \emptyset$.
3. Quelle est la complexité de la construction d'une grammaire algébrique pour le langage $L(G) \cap \{w\}$?

   Un problème avec cet algorithme est qu'il génère en général beaucoup de règles inutiles. On voudrait filtrer la génération des règles de la grammaire pour n'ajouter une règle avec un non terminal de la forme $[q, A, q']$ (où q et q' sont des états de l'automate reconnaissant w, et A un non terminal de G) que s'il existe réellement un facteur u de w dans $L_G(A) \cap L_{q,q'}$.

   Proposer un algorithme avec cet invariant. Quelle borne de complexité obtenez-vous ?

Exercice 5 (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

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

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