Théorie des langages

Langage et Grammaire Hors-Contexte

Jérôme Delobelle

jerome.delobelle@u-paris.fr

LIPADE - Université de Paris

Grammaires Hors-Contexte

1. Définitions
2. Grammaires réduites
3. Grammaires propres
4. Formes normales
5. Clôture des langages algébriques

Définitions

Grammaire Hors-Contexte

Grammaire Hors-Contexte ou algébrique

Une **grammaire** $G = \langle V, \Sigma, P, S \rangle$ est **hors-contexte** (ou **algébrique**) si :

- V est un alphabet
- $\Sigma \subseteq V$ est l'ensemble des symboles terminaux
- $V \setminus \Sigma$ est l'ensemble des symboles non terminaux
- $S \in V \setminus \Sigma$ est le symbole de départ
- $P \subseteq ((V \setminus \Sigma) \times V^*)$ est l'ensemble (fini) de règles de production

Une grammaire est hors-contexte si **toutes** ses règles de production sont hors-contexte.

Rappel : une règle hors-contexte est de la forme suivante :

$$A \to \beta, \text{ où } A \in V \setminus \Sigma, \ \beta \in V^*$$

Grammaire Hors-Contexte

Les notions vu précédemment dans le cours sur les grammaires restent valables pour les grammaires algébriques :

- Dérivation en une étape
- Dérivation en plusieurs étapes
- Mots/langage généré(s) par une grammaire
- Arbres de dérivation
- Grammaire ambigüe

Langage Hors-Contexte

Langage algébrique

Un langage L est **algébrique** s'il existe une grammaire algébrique G telle que

$$\mathcal{L}(G) = L$$

Langage Hors-Contexte

Langage algébrique

Un langage L est **algébrique** s'il existe une grammaire algébrique G telle que

$$\mathcal{L}(G) = L$$

- **Conséquence :** Tout langage rationnel est algébrique.
  - ATTENTION : la réciproque est fausse
- Un langage **algébrique non rationnel**
  - n'est pas reconnu par un automate fini
  - n'est pas décrit par une expression régulière
  - il n'existe pas de grammaire régulière pour l'engendrer.

Grammaire hors contexte : exemple 1

Soit $G = \langle V, \Sigma, P, S \rangle$ avec

- $\Sigma = \{a, b\}$
- $V \setminus \Sigma = \{S\}$
- Axiome S
- 2 règles de production :
  - $S \to aSb$
  - $S \to \epsilon$
- $\mathcal{L}(G) = \{a^n b^n | n \geq 0\}$

Grammaire hors contexte : mots bien parenthésés

Description du langage contenant tous les mots bien parenthésés en ne nous focalisant uniquement sur les parenthèses (le langage de Dyck).

Supposons que S représente une expression bien parenthésée.
Soit $G = \langle V, \Sigma, P, S \rangle$ avec

- $\Sigma = \{(,)\}$
- $V \setminus \Sigma = \{S\}$
- Axiome S
- $P = \{S \to \epsilon, S \to SS, S \to (S)\}$

Grammaire hors contexte : langage PASCAL

$$
\begin{aligned}
\textit{instruction} \quad \rightarrow \quad & \epsilon \\
& | \textit{variable} := \textit{expression} \\
& | \textbf{begin } \textit{liste-instructions } \textbf{end} \\
& | \textbf{if } \textit{expression } \textbf{then } \textit{instruction} \\
& | \textbf{if } \textit{expression } \textbf{then } \textit{instruction } \textbf{else } \textit{instruction} \\
& | \textbf{case } \textit{expression } \textbf{of } \textit{liste-case } \textbf{end} \\
& | \textbf{while } \textit{expression } \textbf{do } \textit{instruction} \\
& | \textbf{repeat } \textit{instruction } \textbf{until } \textit{expression} \\
& | \textbf{for } \textit{varid} := \textit{liste-pour } \textbf{do } \textit{instruction} \\
& | \textit{identificateur-procedure} \\
& | \textit{identificateur-procedure}(\textit{liste-expressions}) \\
& | \textbf{goto } \textit{etiquette} \\
& | \textbf{with } \textit{liste-variables-record } \textbf{do } \textit{instruction} \\
& | \textit{etiquette} : \textit{instruction}
\end{aligned}
$$

Grammaire hors contexte : langage PASCAL

instruction $\rightarrow$ ϵ
| *variable* := *expression*
| **begin** *liste-instructions* **end**
| **if** *expression* **then** *instruction*
| **if** *expression* **then** *instruction* **else** *instruction*
| **case** *expression* **of** *liste-case* **end**
| **while** *expression* **do** *instruction*
| **repeat** *instruction* **until** *expression*
| **for** *varid* := *liste-pour* **do** *instruction*
| *identificateur-procedure*
| *identificateur-procedure*(*liste-expressions*)
| **goto** *etiquette*
| **with** *liste-variables-record* **do** *instruction*
| *etiquette* : *instruction*

- Symboles terminaux : **begin**, **if**, **then**, **else**, **end**, **case**, **of**, **while**...
- Les variables (ou symboles non terminaux) sont définies dans d'autres règles

$\Rightarrow$ Grammaire Pascal : plus de 300 règles

Analyse syntaxique

- Les langages de programmation sont des langages algébriques. Ils sont spécifiés par des grammaires algébriques.
- Dans sa phase d'analyse syntaxique, un compilateur teste si un programme est bien un élément du langage de programmation dans lequel il est écrit.
- Pour un programme donné, le compilateur essaie de reconstituer l'arbre de dérivation (ou arbre syntaxique) de bas en haut en procédant par réductions successives : on parle alors d'analyse ascendante.
- A l'issue de cette phase, on sait si notre programme est syntaxiquement correct ou pas

Grammaires réduites

Grammaires réduites

Symbole non terminal utile

Soit $G = \langle V, \Sigma, P, S \rangle$. Un symbole non terminal $X \in V \setminus \Sigma$ est

- **Productif** si $L_G(X) \neq \emptyset$
- **Accessible** s'il existe des mots α, β tels que $S \xrightarrow[G]{*} \alpha X \beta$
- **Utile** s'il est productif, qu'il existe des mots α, β tels que $S \xrightarrow[G]{*} \alpha X \beta$ et tels que α et β ne contiennent que des symboles non terminaux productifs.

Grammaires réduites

Symbole non terminal utile

Soit $G = \langle V, \Sigma, P, S \rangle$. Un symbole non terminal $X \in V \setminus \Sigma$ est

- **Productif** si $L_G(X) \neq \emptyset$
- **Accessible** s'il existe des mots α, β tels que $S \xrightarrow[G]{*} \alpha X \beta$
- **Utile** s'il est productif, qu'il existe des mots α, β tels que $S \xrightarrow[G]{*} \alpha X \beta$ et tels que α et β ne contiennent que des symboles non terminaux productifs.

Grammaire réduite

Une grammaire est **réduite** si tous ses symboles non terminaux sont utiles.

Grammaires réduites

Symbole non terminal utile

Soit $G = \langle V, \Sigma, P, S \rangle$. Un symbole non terminal $X \in V \setminus \Sigma$ est

- **Productif** si $L_G(X) \neq \emptyset$
- **Accessible** s'il existe des mots α, β tels que $S \xrightarrow[G]{*} \alpha X \beta$
- **Utile** s'il est productif, qu'il existe des mots α, β tels que $S \xrightarrow[G]{*} \alpha X \beta$ et tels que α et β ne contiennent que des symboles non terminaux productifs.

Grammaire réduite

Une grammaire est **réduite** si tous ses symboles non terminaux sont utiles.

En supprimant les symboles inutiles dans une grammaire, on ne change pas le langage engendré.

Algorithme de calcul des symboles non terminaux productifs

1. Calculer l'ensemble V_0 des symboles non terminaux X pour lesquels il existe une règle $X \to \alpha$, $\alpha \in \Sigma^*$
2. Calculer l'ensemble V_{i+1} formé de V_i et des symboles non terminaux X pour lesquels il existe une règle $X \to \alpha$, $\alpha \in (\Sigma \cup V_i)^*$
3. Arrêter lorsque $V_{i+1} = V_i$. Cet ensemble est l'ensemble des symboles non terminaux productifs.

Algorithme de calcul des symboles non terminaux productifs

1. Calculer l'ensemble V_0 des symboles non terminaux X pour lesquels il existe une règle $X \to \alpha$, $\alpha \in \Sigma^*$
2. Calculer l'ensemble V_{i+1} formé de V_i et des symboles non terminaux X pour lesquels il existe une règle $X \to \alpha$, $\alpha \in (\Sigma \cup V_i)^*$
3. Arrêter lorsque $V_{i+1} = V_i$. Cet ensemble est l'ensemble des symboles non terminaux productifs.

$$
\begin{array}{lcl}
S & \to & A|X \\
A & \to & a \\
X & \to & XY \\
Y & \to & b
\end{array}
$$

$V_0 = \{A, Y\}$ car $A \to a$ et $Y \to b$
$V_1 = V_0 \cup \{S\} = \{A, Y, S\}$ car $S \to A$
$V_2 = V_1 \cup \emptyset = V_1$

$$V_P = \{A, Y, S\}$$

Algorithme de calcul des symboles non terminaux accessibles

1. Poser $W_0 = \{S\}$ (l'axiome de départ)
2. Calculer l'ensemble W_{i+1} formé de W_i et des symboles non terminaux X pour lesquels il existe une règle $Y \to \alpha X \beta$, avec $Y \in W_i$ et $\alpha, \beta \in (\Sigma \cup V)^*$
3. Arrêter lorsque $W_{i+1} = W_i$. Cet ensemble est l'ensemble des symboles non terminaux accessibles.

Algorithme de calcul des symboles non terminaux accessibles

1. Poser $W_0 = \{S\}$ (l'axiome de départ)
2. Calculer l'ensemble W_{i+1} formé de W_i et des symboles non terminaux X pour lesquels il existe une règle $Y \to \alpha X \beta$, avec $Y \in W_i$ et $\alpha, \beta \in (\Sigma \cup V)^*$
3. Arrêter lorsque $W_{i+1} = W_i$. Cet ensemble est l'ensemble des symboles non terminaux accessibles.

$$
\begin{array}{lcl}
S & \to & U \\
U & \to & aYb \\
X & \to & a \\
Y & \to & Z \\
Z & \to & c|\epsilon
\end{array}
$$

$$
\begin{aligned}
W_0 &= \{S\} \\
W_1 &= W_0 \cup \{U\} = \{S, U\} \text{ car } S \to U \\
W_2 &= W_1 \cup \{Y\} = \{S, U, Y\} \text{ car } U \to aYb \\
W_3 &= W_3 \cup \{Z\} = \{S, U, Y, Z\} \text{ car } Y \to Z \\
W_4 &= W_3 \cup \emptyset = W_3
\end{aligned}
$$

$$V_A = \{S, U, Y, Z\}$$

Algorithme de réduction d'une grammaire

1. Déterminer l'ensemble des symboles non terminaux productifs
2. Supprimer les symboles non terminaux non productifs, ainsi que les règles dans lesquels ils figurent
3. Si l'axiome S est improductif, la grammaire réduite a pour seul symbole non terminal S, et un ensemble de règles vide
4. Si l'axiome S est productif, déterminer tous les symboles non terminaux accessibles à partir de S. Ceci permet d'obtenir l'ensemble des symboles non terminaux utiles.
5. Supprimer tous les autres symboles non terminaux, ainsi que les règles dans lesquelles ils figurent.

La grammaire ainsi obtenue est réduite.

Algorithme de réduction d'une grammaire : exemple

- On considère la grammaire ayant les règles suivantes :

$$\begin{aligned} S &\rightarrow a|X \\ X &\rightarrow XY \\ Y &\rightarrow b \end{aligned}$$

- Symboles non terminaux productifs : $\{S, Y\}$. X est donc improductif, on peut le supprimer
- On obtient la grammaire

$$\begin{aligned} S &\rightarrow a \\ Y &\rightarrow b \end{aligned}$$

- L'ensemble des symboles non terminaux accessibles est $\{S\}$. On supprime Y

$$S \rightarrow a$$

- La grammaire est réduite.

Algorithme de réduction d'une grammaire : exemple 2

- On considère la grammaire ayant les règles suivantes :

$$\begin{aligned} S &\rightarrow XYZW \\ X &\rightarrow cX \\ Y &\rightarrow ab \\ Z &\rightarrow cYa|WSW \\ W &\rightarrow \epsilon \end{aligned}$$

- Symboles non terminaux productifs : $\{Y, Z, W\}$. S est donc improductif.
- La grammaire réduite a donc pour seul symbole non terminal S, et un ensemble de règles vide

Grammaires propres

Grammaires propres

Une grammaire propre ne contient :

- **Pas de règle unitaire** (ou **renommage**) : G ne contient pas de production de la forme $A \to B$, avec $A, B \in V \setminus \Sigma$ (symboles non terminaux)
- **Pas de ϵ-règle** : G ne contient pas de règle du type $A \to \epsilon$

Grammaires propres

Théorème

Tout langage algébrique $L = L \setminus \{\epsilon\}$ peut être engendré par une grammaire propre.

Grammaires propres

Théorème

Tout langage algébrique $L = L \setminus \{\epsilon\}$ peut être engendré par une grammaire propre.

- Un symbole non terminal X est annulable si $X \xrightarrow{*} \epsilon$
- X est annulable si :
  - Il existe une règle $X \to \epsilon$, ou
  - Il existe une règle $X \to Y_1 \dots Y_n$, avec $Y_1, \dots, Y_n$ des symboles non terminaux annulables

Algorithme de calcul des symboles non terminaux annulables

1. Calculer l'ensemble N_0 des symboles non terminaux X telles qu'il existe une règle $X \to \epsilon$
2. Calculer l'ensemble N_{i+1} formé de N_i et des symboles non terminaux X telles qu'il existe une règle $X \to \alpha$ avec $\alpha \in N_i^*$
3. Arrêter lorsque $N_{i+1} = N_i$. Cet ensemble est l'ensemble des symboles non terminaux annulables.

Algorithme de calcul des symboles non terminaux annulables

1. Calculer l'ensemble N_0 des symboles non terminaux X telles qu'il existe une règle $X \to \epsilon$
2. Calculer l'ensemble N_{i+1} formé de N_i et des symboles non terminaux X telles qu'il existe une règle $X \to \alpha$ avec $\alpha \in N_i^*$
3. Arrêter lorsque $N_{i+1} = N_i$. Cet ensemble est l'ensemble des symboles non terminaux annulables.

$$
\begin{array}{lcl}
S & \to & A|B \\
A & \to & XZ \\
B & \to & b \\
Z & \to & X \\
X & \to & x|\epsilon \\
Y & \to & \epsilon
\end{array}
$$

$$
\begin{aligned}
N_0 &= \{X, Y\} \text{ car } X \to \epsilon \text{ et } Y \to \epsilon \\
N_1 &= N_0 \cup \{Z\} = \{X, Y, Z\} \text{ car } Z \to X \\
N_2 &= N_1 \cup \{A\} = \{X, Y, Z, A\} \text{ car } A \to XZ \\
N_3 &= N_3 \cup \{S\} = \{X, Y, Z, A, S\} \text{ car } S \to A \\
N_4 &= N_3 \cup \emptyset = N_3
\end{aligned}
$$

$$V_{An} = \{X, Y, Z, A, S\}$$

Algorithme d'élimination des ϵ-règles

1. Calculer l'ensemble N des symboles non terminaux annulables
2. Remplacer chaque règle $X \to \alpha$ par toutes les règles obtenues en remplaçant, de toutes les façons possibles, les occurrences de symboles non terminaux annulables par le mot vide. S'il y a n occurrences de symboles non terminaux annulables dans α, cela donne 2^n règles.
3. Supprimer les ϵ-règles. La grammaire obtenue est équivalente à la grammaire de départ au mot vide près, et est sans ϵ-règle.

Algorithme d'élimination des ϵ-règles : exemple

- On considère la grammaire ayant les règles suivantes :
  - $S \rightarrow aSbS|\epsilon$
- S est un symbole non-terminal annulable car $S \rightarrow \epsilon$
- Remplacer le symbole non terminal S par le mot vide de toutes les façons possibles dans $aSbS$ donne les 4 mots $aSbS$, abS, aSb et ab
- La grammaire propre obtenue est
  - $S \rightarrow aSbS|abS|aSb|ab$

Algorithme d'élimination des ϵ-règles : exemple 2

- On considère la grammaire ayant les règles suivantes :
  - $S \rightarrow aSb|SS|\epsilon$
- La première règle donne $S \rightarrow aSb$ et $S \rightarrow ab$
- La seconde règle donne $S \rightarrow SS$
  - La règle $S \rightarrow S$ générée deux fois par notre algorithme est inutile
- La grammaire propre obtenue est
  - $S \rightarrow aSb|ab|SS$

Algorithme d'élimination des règles unitaires

1. Calculer la relation $\geq$ définie par $X \geq Y$ ssi $X \stackrel{*}{\rightarrow} Y$
2. Calculer la relation $\sim$ définie par $X \sim Y$ ssi $X \geq Y$ et $Y \geq X$
3. Choisir un symbole non terminal par classe d'équivalence, et remplacer toutes les occurrences de toutes les symboles non terminaux équivalents par ce représentant. Supprimer toutes les règles unitaires entre symboles non terminaux équivalents.
4. En commançant par les symboles non terminaux qui sont minimaux dans l'ordre $\geq$, remplacer les règles unitaires $X \rightarrow Y$ par toutes les règles $X \rightarrow \alpha$, pour tous les α tels que $Y \rightarrow \alpha$.

La grammaire obtenue est sans règles unitaires.

Algorithme d'élimination des règles-unitaires : exemple

- On considère la grammaire des expressions arithmétiques ayant les règles suivantes :
  - $E \rightarrow E + T | T$
  - $T \rightarrow T * F | F$
  - $F \rightarrow (E) | a | b | c$
- Il y a deux règles unitaires : $E \rightarrow T$ et $T \rightarrow F$
- L'ordre obtenu est $E > T > F$
- On considère les symboles non terminaux en commançant par celles qui sont minimales dans $\geq$
  - On substitue à $T \rightarrow F$ les règles $T \rightarrow (E) | a | b | c$
  - On substitue à $E \rightarrow T$ les règles $E \rightarrow T * F | (E) | a | b | c$
- On obtient :
  - $E \rightarrow E + T | T * F | (E) | a | b | c$
  - $T \rightarrow T * F | (E) | a | b | c$
  - $F \rightarrow (E) | a | b | c$

Formes normales

Forme normale de Chomsky

Forme normale de Chomsky

Une grammaire algébrique $G = \langle V, \Sigma, P, S \rangle$ est sous la **forme normale de Chomsky** si toute production est de la forme :

$$\begin{aligned} A &\to a \\ A &\to BC \end{aligned}$$

avec $A, B, C \in V \setminus \Sigma$, et $a \in \Sigma$

Intérêt de la forme normale de Chomsky :

- Les arbres d'analyse correspondants seront des arbres binaires
- Disposer d'une forme normale pour les grammaires permet de simplifier les développements de certaines preuves

Algorithme de mise en forme normale de Chomsky

On part d'une grammaire propre et réduite.

1. On introduit un nouvel ensemble de symboles non terminaux $Z = \{Z_a | a \in \Sigma\}$
2. On ajoute les règles $Z_a \to a$ pour tout $a \in \Sigma$
3. Toute règle $X \to \alpha$ où α est de longueur 1 est conservée
   - *puisque la grammaire est propre, α est un symbole terminal*
4. Toute règle $X \to \alpha$ où $|\alpha| \geq 2$ est transformée en 2 étapes :
   i) Tout symbole terminal a dans α est remplacé par le symbole non terminal Z_a
   ii) Si $|\alpha| > 2$, soit $\alpha = Y_1 \dots Y_p$. On introduit $p-2$ nouveaux symboles non terminaux $T_1, \dots, T_{p-2}$ et on remplace la règle $X \to Y_1 \dots Y_p$ par les $p-1$ règles

$$X \to Y_1 T_1, \;\; T_1 \to Y_2 T_2, \dots, \;\; T_{p-3} \to Y_{p-2} T_{p-2}, \;\; T_{p-2} \to Y_{p-1} Y_p$$

La grammaire obtenue est en forme normale de Chomsky

Algorithme de mise en forme normale de Chomsky : exemple

- On considère la grammaire propre et réduite ayant les règles suivantes :

$$S \to aSbS|abS|aSb|ab$$

- On introduit d'abord deux nouveaux symboles non terminaux Z_a et Z_b, et on remplace les symboles terminaux. La grammaire devient

$$S \to Z_aSZ_bS|Z_aZ_bS|Z_aSZ_b|Z_aZ_b;\ Z_a \to a;\ Z_b \to b$$

- On considère à présent les règles suivantes :
  - $S \to Z_aSZ_bS$, que l'on transforme en $S \to Z_aT_1$; $T_1 \to ST_2$; $T_2 \to Z_bS$
  - $S \to Z_aZ_bS$, que l'on transforme en $S \to Z_aT_2$; $T_2 \to Z_bS$
  - $S \to Z_aSZ_b$, que l'on transforme en $S \to Z_aT_3$; $T_3 \to SZ_b$
- La grammaire devient sous forme normale de Chomsky :

$$\begin{array}{lcl} S & \to & Z_aT_1|Z_aT_2|Z_aT_3|Z_aZ_b \\ T_1 & \to & ST_2;\ T_2 \to Z_bS;\ T_3 \to SZ_b \\ Z_a & \to & a;\ Z_b \to b \end{array}$$

Forme normale de Greibach

Forme normale de Greibach

Une grammaire algébrique $G = \langle V, \Sigma, P, S \rangle$ est sous la **forme normale de Greibach** si toute production est de la forme :

$$\begin{aligned} A &\rightarrow aA_1 \dots A_n \\ A &\rightarrow a \end{aligned}$$

avec $A, A_i \in V \setminus \Sigma$, et $a \in \Sigma$

Forme normale de Greibach

Forme normale de Greibach

Une grammaire algébrique $G = \langle V, \Sigma, P, S \rangle$ est sous la **forme normale de Greibach** si toute production est de la forme :

$$\begin{aligned} A &\rightarrow aA_1 \dots A_n \\ A &\rightarrow a \end{aligned}$$

avec $A, A_i \in V \setminus \Sigma$, et $a \in \Sigma$

Intérêt de la forme normale de Greibach :

- A chaque dérivation, on détermine un préfixe de plus en plus long formé uniquement de symboles non terminaux
- Permet de construire plus aisément des analyseurs permettant de retrouver l'arbre d'analyse associé à un mot généré

Clôture des langages algébriques

Grammaires hors-contextes et langages algébriques

- Les grammaires hors-contextes génèrent des langages algébriques
- Existe-t'il des méthodes pour construire les grammaires plus facilement à partir des langages ?

Grammaires hors-contextes et langages algébriques

- Les grammaires hors-contextes génèrent des langages algébriques
- Existe-t'il des méthodes pour construire les grammaires plus facilement à partir des langages ?

$\Rightarrow$ Oui, en décomposant les langages

Clôture par union

Clôture par union

Soient L_1 et L_2 deux langages algébriques.

Alors $L_1 \cup L_2$ est un langage algébrique.

Clôture par union

Clôture par union

Soient L_1 et L_2 deux langages algébriques.

Alors $L_1 \cup L_2$ est un langage algébrique.

Preuve par construction :

- Soient $G_1 = \langle V_1, \Sigma_1, P_1, S_1 \rangle$ et $G_1 = \langle V_2, \Sigma_2, P_2, S_2 \rangle$ engendrant L_1 et L_2 et tels que $(V_1 \setminus \Sigma_1) \cap (V_2 \setminus \Sigma_2) = \emptyset$ et $S \notin V_1 \cup V2$ (et sinon on renomme)
- On construit $G = \langle V, \Sigma, P, S \rangle$ telle que :
  - $V = V_1 \cup V_2 \cup \{S\}$
  - $\Sigma = \Sigma_1 \cup \Sigma_2$
  - $P = P_1 \cup P_2 \cup \{S \to S_1 | S_2\}$

Clôture par concaténation

Clôture par concaténation

Soient L_1 et L_2 deux langages algébriques.

Alors $L_1.L_2$ est un langage algébrique.

Clôture par concaténation

Clôture par concaténation

Soient L_1 et L_2 deux langages algébriques.

Alors $L_1.L_2$ est un langage algébrique.

Preuve par construction :

- Soient $G_1 = \langle V_1, \Sigma_1, P_1, S_1 \rangle$ et $G_1 = \langle V_2, \Sigma_2, P_2, S_2 \rangle$ engendrant L_1 et L_2 et tels que $(V_1 \setminus \Sigma_1) \cap (V_2 \setminus \Sigma_2) = \emptyset$ et $S \notin V_1 \cup V2$ (et sinon on renomme)
- On construit $G = \langle V, \Sigma, P, S \rangle$ telle que :
  - $V = V_1 \cup V_2 \cup \{S\}$
  - $\Sigma = \Sigma_1 \cup \Sigma_2$
  - $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$

Clôture par opération étoile

Clôture par opération étoile

Soit L un langage algébrique.

Alors L^* est un langage algébrique.

Clôture par opération étoile

Clôture par opération étoile

Soit L un langage algébrique.

Alors L^* est un langage algébrique.

Preuve par construction :

- Soit $G_1 = \langle V_1, \Sigma_1, P_1, S_1 \rangle$ engendrant le langage L
- On construit $G = \langle V, \Sigma, P, S \rangle$ engendrant le langage L^* telle que :
  - $V = V_1 \cup \{S\}$
  - $\Sigma = \Sigma_1$
  - $P = P_1 \cup \{S \to \epsilon | S_1 S\}$

Exemple

Peut-on trouver une grammaire algébrique pour
$L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$?

Exemple

Peut-on trouver une grammaire algébrique pour
$L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$?

- L se décompose en l'union de
  - $L_1 = \{a^i b^j c^k | i = j, k \geq 0\}$
  - $L_2 = \{a^i b^j c^k | i \geq 0, j = k\}$

Exemple

Peut-on trouver une grammaire algébrique pour
$L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$?

- L se décompose en l'union de
  - $L_1 = \{a^i b^j c^k | i = j, k \geq 0\}$
  - $L_2 = \{a^i b^j c^k | i \geq 0, j = k\}$
- $L_1 = \{a^i b^j c^k | i = j, k \geq 0\}$ est la concaténation de
  - $\{a^i b^j | i = j\}$ et
  - $\{c^k | k \geq 0\}$

Exemple

Peut-on trouver une grammaire algébrique pour
$L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$?

- L se décompose en l'union de
  - $L_1 = \{a^i b^j c^k | i = j, k \geq 0\}$
  - $L_2 = \{a^i b^j c^k | i \geq 0, j = k\}$
- $L_1 = \{a^i b^j c^k | i = j, k \geq 0\}$ est la concaténation de
  - $\{a^i b^j | i = j\}$ et
  - $\{c^k | k \geq 0\}$
- $L_2 = \{a^i b^j c^k | i \geq 0, j = k\}$ est la concaténation de
  - $\{a^i | i \geq 0\}$ et
  - $\{b^j c^k | j = k\}$

Exemple, suite

- $\{a^i b^j | i = j\}$
  - $S_1 \rightarrow \epsilon | aS_1 b$

Exemple, suite

- $\{a^i b^j | i = j\}$
  - $S_1 \rightarrow \epsilon | aS_1 b$
- $\{c^k | k \geq 0\}$
  - $S_2 \rightarrow \epsilon | cS_2$

Exemple, suite

- $\{a^i b^j | i = j\}$
  - $S_1 \rightarrow \epsilon | aS_1 b$
- $\{c^k | k \geq 0\}$
  - $S_2 \rightarrow \epsilon | cS_2$
- $L_1 = \{a^i b^j c^k | i = j, k \geq 0\}$
  - $S_1 \rightarrow \epsilon | aS_1 b$
  - $S_2 \rightarrow \epsilon | cS_2$
  - $S_{L_1} \rightarrow S_1 S_2$

Exemple, suite

- $\{a^i b^j | i = j\}$
  - $S_1 \to \epsilon | aS_1 b$
- $\{c^k | k \geq 0\}$
  - $S_2 \to \epsilon | cS_2$
- $L_1 = \{a^i b^j c^k | i = j, k \geq 0\}$
  - $S_1 \to \epsilon | aS_1 b$
  - $S_2 \to \epsilon | cS_2$
  - $S_{L_1} \to S_1 S_2$
- $L_2 = \{a^i b^j c^k | i \geq 0, j = k\}$
  - $S_3 \to \epsilon | aS_3$
  - $S_4 \to \epsilon | bS_4 c$
  - $S_{L_2} \to S_3 S_4$

Exemple, suite

- $L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$

Exemple, suite

- $L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$
- $G = \langle V, \Sigma, S, P \rangle$

Exemple, suite

- $L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$
- $G = \langle V, \Sigma, S, P \rangle$
- $V \setminus \Sigma = \{S_1, S_2, S_3, S_4, S_{L_1}, S_{L_2}, S\}$

Exemple, suite

- $L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$
- $G = \langle V, \Sigma, S, P \rangle$
- $V \setminus \Sigma = \{S_1, S_2, S_3, S_4, S_{L_1}, S_{L_2}, S\}$
- $\Sigma = \{a, b, c\}$

Exemple, suite

- $L = \{a^i b^j c^k | i = j \text{ ou } j = k\}$
- $G = \langle V, \Sigma, S, P \rangle$
- $V \setminus \Sigma = \{S_1, S_2, S_3, S_4, S_{L_1}, S_{L_2}, S\}$
- $\Sigma = \{a, b, c\}$
- P :

$$
\begin{array}{lcl}
S_1 & \rightarrow & \epsilon | aS_1 b \\
S_2 & \rightarrow & \epsilon | cS_2 \\
S_{L_1} & \rightarrow & S_1 S_2 \\
S_3 & \rightarrow & \epsilon | aS_3 \\
S_4 & \rightarrow & \epsilon | bS_4 c \\
S_{L_2} & \rightarrow & S_3 S_4 \\
S & \rightarrow & S_{L_1} | S_{L_2}
\end{array}
$$

Clôture par intersection et complémentation

Clôture par les opérations rationnelles

La classe des langages algébriques est close par les opérations rationnelles (union, concaténation et étoile de Kleene).

Clôture par intersection

La classe des langages algébriques n'est pas close par intersection

Contre-exemple : $L_1 = \{a^n b^n c^m\}$ et $L_2 = \{a^m b^n c^n\}$ sont deux langages hors-contexte mais $L_1 \cap L_2 = \{a^n b^n c^n\}$ n'est pas un langage hors-contexte (mais contextuel).

Clôture par complémentation

La classe des langages algébriques n'est pas close par complémentation

Corollaire déduit par application directe de la loi de De Morgan.

Problèmes décidables pour les grammaires hors-contextes

Il existe au moins un algorithme permettant de :

- décider si un mot u de Σ^* est ou n'est pas engendré par une grammaire HC ;
- déterminer si le langage engendré par une grammaire HC est vide ;
- déterminer si le langage engendré par une grammaire HC est infini.

Problèmes indécidables pour les grammaires hors-contextes

Il n'existe pas d'algorithme permettant de :

- décider si deux grammaires HC sont équivalentes ;
- décider si le langage engendré par une grammaire HC est inclus dans le langage engendré par une autre grammaire HC ;
- déterminer si une grammaire HC engendre en fait un langage régulier ;
- décider si une grammaire est ambiguë.