Automates, langages et compilation

Grammaires

Isabelle Ryl

2024 – 2025

Cours de L3 - Université Paris Dauphine-PSL

Grammaires

Grammaires

Présentation

Exemple

Revenons sur l'exemple du premier cours

- les expressions régulières. Ex :

$$(b+c)^*a(b+c)^*a(a+b+c)^*$$

- les grammaires. Ex :

$$\begin{aligned} S &\longrightarrow TaTaT \\ T &\longrightarrow Ta \mid Tb \mid Tc \mid \varepsilon \end{aligned}$$

Exemple de dérivation :

$$\begin{aligned} S &\longrightarrow TaTaT \\ &\longrightarrow TaaTaT \\ &\longrightarrow TaaTbaT \\ &\longrightarrow TaaTbaTc \\ &\longrightarrow aaTbaTc \\ &\longrightarrow aaTbac \\ &\longrightarrow aabac \end{aligned}$$

Similitude avec le langage naturel (1/2)

$\Sigma = \{le, la, chien, balle, maitre, lance, attrappe\}$

Nous pouvons imaginer décomposer une phrase P, en sujet-verbe-complément. Nous pourrions alors définir les catégories syntaxiques de la manière suivante :

$$\begin{aligned} P &\longrightarrow SUJET\ VERBE\ COMPLEMENT \\ VERBE &\longrightarrow lance \mid attrappe \end{aligned}$$

Le sujet comme le complément sont des groupes nominaux composés d'un article et d'un nom :

$$\begin{aligned} SUJET &\longrightarrow GN \\ COMPLEMENT &\longrightarrow GN \\ GN &\longrightarrow ARTICLE\ NOM \\ ARTICLE &\longrightarrow le \mid la \\ NOM &\longrightarrow chien \mid balle \mid maitre \end{aligned}$$

Similitude avec le langage naturel (2/2)

Exemple de dérivation :

$$
\begin{aligned}
P &\longrightarrow \textit{SUJET VERBE COMPLEMENT} \\
&\longrightarrow \textit{SUJET lance COMPLEMENT} \\
&\longrightarrow \textit{GN lance COMPLEMENT} \\
&\longrightarrow \textit{GN lance GN} \\
&\longrightarrow \textit{GN lance ARTICLE NOM} \\
&\longrightarrow \textit{GN lance la NOM} \\
&\longrightarrow \textit{ARTICLE NOM lance la NOM} \\
&\longrightarrow \textit{ARTICLE maitre lance la NOM} \\
&\longrightarrow \textit{le maitre lance la NOM} \\
&\longrightarrow \textit{le maitre lance la balle}
\end{aligned}
$$

Similitude avec le langage naturel (2/3)

Mais ... :

$$\begin{array}{rcl} P & \longrightarrow & \textit{SUJET VERBE COMPLEMENT} \\ & \longrightarrow & \textit{SUJET lance COMPLEMENT} \\ & \longrightarrow & \textit{GN lance COMPLEMENT} \\ & \longrightarrow & \textit{GN lance GN} \\ & \longrightarrow & \textit{GN lance ARTICLE NOM} \\ & \longrightarrow & \textit{GN lance la NOM} \\ & \longrightarrow & \textit{ARTICLE NOM lance la NOM} \\ & \longrightarrow & \textit{ARTICLE chien lance la NOM} \\ & \longrightarrow & \textit{la chien lance la NOM} \\ & \longrightarrow & \textit{la chien lance la maitre} \end{array}$$

➥ seule la syntaxe est vérifiée

Grammaire - Définition

Une grammaire est donc un mécanisme de génération des langages, nous verrons qu'il est plus général que les expressions rationnelles et les automates finis

Les grammaires ont été introduites dans plusieurs domaines en particulier pour formaliser le langage naturel et sa grammaire par Noam Chomsky, mais sont très utilisées en informatique notamment en compilation

Pour compiler un langage, il faut vérifier que les termes utilisés sont corrects par l'<u>analyse lexicale</u> qui utilisera les langages reconnaissables puis vérifier que la structure du programme est correcte, *i.e.* que la syntaxe est correct par l'<u>analyse syntaxique</u>, en vérifiant que la grammaire du langage de programmation est respectée

Grammaire de langage de programmation - Exemple

Exemple tiré du manuel de référence Python*

```
if_stmt     ::=  "if" assignment_expression ":" suite
                 ("elif" assignment_expression ":" suite)*
                 ["else" ":" suite]

while_stmt  ::=  "while" assignment_expression ":" suite
                 ["else" ":" suite]
```

(*) https ://docs.python.org/fr/3/reference/index.html

Grammaire - Définition

Une grammaire est un quadruplet $\mathcal{G} = (\Sigma, V, S, R)$ tel que :

- Σ est l'alphabet sur lequel le langage est défini, appelé alphabet terminal
- V est l'alphabet des symboles utilisés au cours de la génération, appelé aphabet non terminal, ou les « variables » $\Sigma \cap V = \emptyset$
- $S \in V$ est le symbole de départ, appelé également axiome
- R est un ensemble de règles de réécriture de la forme $\alpha \to \beta$ avec $\alpha \in (\Sigma \cup V)^+$ et $\beta \in (\Sigma \cup V)^*$

Réécriture

<u>Idée</u>. Une règle de réécriture permet de réécrire un mot u en un mot v en appliquant une règle $f_1 \to f_2$ si on trouve le facteur f_1 dans le mot u, dans ce cas on « remplace » l'instance de f_1 trouvée dans u par f_2 pour former v

Une **règle de réécriture** $r : f_1 \to f_2$ s'applique pour réécrire un mot u en un mot v, si et seulement si $\exists u_1, u_2$ tels que $u = u_1 f_1 u_2$ et $v = u_1 f_2 u_2$, on note $u \xrightarrow{r} v$

Soit $\mathcal{R}$ un système de règles de réécriture, une **dérivation** d'un mot u, notée $u \xrightarrow[*]{\mathcal{R}} v$, est la clôture par application des règles de réécriture :

$$\exists u = u_1, u_2, \ldots, u_{n-1}, u_n = v \text{ avec } \forall 1 \leq i < n, \exists r_i \in \mathcal{R} \text{ telle que } u_i \xrightarrow{r_i} u_{i+1}$$

Langage engendré

Soient $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire, $\alpha \in (\Sigma \cup V)^+$ et $\beta \in (\Sigma \cup V)^*$

La grammaire $\mathcal{G}$ permet de **dériver** u en v, si $u \xrightarrow[*]{R} v$, et on note $u \xrightarrow[*]{\mathcal{G}} v$

Le **langage engendré par une grammaire** (ou généré) $\mathcal{G}$ est $\mathcal{L}(\mathcal{G}) \in \Sigma^*$ est l'ensemble des mots de l'alphabet terminal qui peuvent être dérivés à partir de S, soit

$$\mathcal{L}(\mathcal{G}) = \{u \in \Sigma^* \mid S \xrightarrow[*]{\mathcal{G}} u\}$$

Un langage $\mathcal{L} \in \Sigma^*$ est dit **recursivement énumérable** s'il existe une grammaire $\mathcal{G}$ telle que $\mathcal{L}(\mathcal{G}) = \mathcal{L}$

Hiérarchie de Chomsky 1/3

Classification introduite par Noam Chomsky, basée sur la forme des règles des grammaires, qui permet d'obtenir des classes ordonnées par inclusion des langages engendrés

Type 0. Pas de contrainte sur la forme des règles

Type 1. Grammaires contextuelles (ou sensibles au contexte) : les règles sont de la forme

$$\alpha T \beta \longrightarrow \alpha\gamma\beta$$

où $T \in V$, $\alpha, \beta \in (\Sigma \cup V)^*$ et $\gamma \in (\Sigma \cup V)^+$

- une seule variable est réécrite à chaque règle (ici T)
- les « contextes » gauche et droit ne sont pas modifiés

Hiérarchie de Chomsky 2/3

Type 2. Grammaires hors contexte (ou context-free ou algébrique) : les règles sont de la forme

$$T \longrightarrow \alpha$$

où $T \in V$, $\alpha \in (\Sigma \cup V)^*$

- une seule variable est réécrite à chaque règle (ici T)
- chaque variable peut être réécrite indépendamment du contexte

Type 3. Grammaires régulières, la forme des règles est contrainte selon deux cas :

- régulière à droite $T \longrightarrow aU$ ou $T \longrightarrow a$
- régulière à gauche $T \longrightarrow Ua$ ou $T \longrightarrow a$

où $T, U \in V$, $a \in \Sigma$

Hiérarchie de Chomsky 3/3

Les différents types de grammaire sont associés à différents types de langages, qui sont reconnus par différents types de « machines »

- Type 3, grammaires régulières, langages dits réguliers que nous avons vus sous le nom de rationnels ou reconnaissables, automates finis
- Type 2, grammaires hors contexte, langages hors contexte, automates à pile
- Type 1, grammaires contextuelles, langages contextuels, machines de Turing
- Type 0, langages décidables (reconnus par une machine en un temps fini), machines de Turing

Grammaire - Remarques

Remarques

- Une grammaire engendre un et un seul langage
- Un langage peut être engendré par plusieurs grammaires différentes

Deux grammaires sont **équivalentes** si elles engendrent le même langage

Attention, deux grammaires distinctes pouvant être équivalentes, un langage de type n peut être engendré par une grammaire de type $m \leq n$.
Pour montrer qu'un langage n'est pas de type n il faut montrer que toute grammaire qui l'engendre est de type $< n$

Exemples

Construire une grammaire qui engendre le langage $a^n b^n$.

$$S \longrightarrow aSb \mid \varepsilon$$

Exemples de dérivation :

$$\begin{array}{lcl} S & \longrightarrow & aSb \\ & \longrightarrow & ab \\ S & \longrightarrow & aSb \\ & \longrightarrow & aaSbb \\ & \longrightarrow & aabb \\ S & \longrightarrow & aSb \\ & \longrightarrow & aaSbb \\ & \longrightarrow & aaaSbbb \\ & \longrightarrow & aaabbb \end{array}$$

➥ $a^n b^n$ peut être engendré par une grammaire

Exemples

Quel est le langage engendré par la grammaire :

$$S \longrightarrow aSa \mid bSb \mid cSc \mid a \mid b \mid c \mid \varepsilon$$

Exemple de dérivation :

$$\begin{aligned} S &\longrightarrow aSa \\ &\longrightarrow abSba \\ &\longrightarrow abbSbba \\ &\longrightarrow abbcScbba \\ &\longrightarrow abbccbba \end{aligned}$$

➥ Les palindromes sur $\{a, b, c\}$

Montrer qu'un langage est généré par une grammaire

Pour montrer qu'une grammaire $\mathcal{G}$ engendre un langage $\mathcal{L}$, il faut montrer que :

- $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}$ ➥ tout mot engendré par $\mathcal{G}$ est un mot de $\mathcal{L}$
- $\mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$ ➥ tout mot de $\mathcal{L}$ peut être engendré par $\mathcal{G}$

Exemple

Reprenons l'exemple ci-dessus, considérons la grammaire $\mathcal{G} = (\Sigma, V, S, R)$ avec $\Sigma = \{a, b, c\}$, $V = \{S\}$ et $R = \{S \longrightarrow aSa \mid bSb \mid cSc \mid a \mid b \mid c \mid \varepsilon\}$ et $\mathcal{L}$ le langage des palindromes sur Σ et montrons que $\mathcal{G}$ génère $\mathcal{L}$

Exemple Palindromes - Preuve (1/2)

Montrons $\mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$ par récurrence sur la longueur des mots de $\mathcal{L}$

Soit $u \in \mathcal{L}$

- si $|u| \leq 1$ alors $u \in \{\varepsilon, a, b, c\}$ et $u \in \mathcal{L}(\mathcal{G})$, supposons donc que l'inclusion est vraie pour tout u tel que $|u| \leq n$
- si $|u| > 1$ alors $u = ava$ ou $u = bvb$ ou $u = cvc$ avec v un palindrome et $|v| \leq n$. Par hypothèse de récurrence $v \in \mathcal{L}(\mathcal{G})$ et donc $S \xrightarrow[*]{\mathcal{G}} v$, nous pouvons donc dériver :

$$\begin{array}{rl} & S \to aSa \xrightarrow[*]{\mathcal{G}} ava \\ \text{et} & S \to bSb \xrightarrow[*]{\mathcal{G}} bvb \\ \text{et} & S \to cSc \xrightarrow[*]{\mathcal{G}} cvc \end{array}$$

et donc $u \in \mathcal{L}(\mathcal{G})$

Exemple Palindromes - Preuve (2/2)

Montrons $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}$ par récurrence sur la longueur des mots de $\mathcal{L}(\mathcal{G})$

Soit $u \in \mathcal{L}(\mathcal{G})$

- si $|u| \leq 1$ alors $u \in \{\varepsilon, a, b, c\}$ et $u \in \mathcal{L}$, supposons donc que l'inclusion est vraie pour tout u tel que $|u| \leq n$
- si $|u| > 1$ alors la dérivation qui engendre u commence par l'application d'une règle $S \rightarrow aSa$ ou $S \rightarrow bSb$ ou $S \rightarrow cSc$, les cas étant strictement identiques, supposons qu'il s'agit de la première. Alors la dérivation s'écrit $S \rightarrow aSa \overset{\mathcal{G}}{\underset{*}{\rightarrow}} u = ava$ avec $S \overset{\mathcal{G}}{\underset{*}{\rightarrow}} v$ et $|v| < n$, donc par hypothèse de récurrence $v \in \mathcal{L}$, donc v est un palindrome et donc $ava = u$ également : $u \in \mathcal{L}$

Notation BNF

La syntaxe de nombreux langages de programmation est décrite à l'aide de la notation de Backus-Naur (ou Backus-Naur form) :

- la dérivation $\longrightarrow$ est notée $::=$
- les non terminaux sont notés entre $<$ et $>$
- les alternatives en partie droite sont séparées par des $|$
- les caractères spéciaux sont entourés de $'$

Exemple. Grammaire qui engendre les sommes de nombres

$$\begin{array}{lcl} S & \longrightarrow & S + S \mid NB \\ NB & \longrightarrow & NB\ CH \mid CH \\ CH & \longrightarrow & 0 \mid 1 \mid \cdots \mid 9 \end{array}$$

```
<sum>  ::=  <sum> '+' <sum> | <num>
<num>  ::=  <num> <dig> | <dig>
<dig>  ::=  0 | 1 | ... | 9
```

Grammaires

Grammaires régulières

Langages réguliers

Un langage est dit **régulier** si et seulement si il existe une grammaire régulière qui l'engendre. On note $Reg(\Sigma^*)$ la classe des langages réguliers sur Σ

Théorème

$Reg(\Sigma^*) = Rat(\Sigma^*) = Rec(\Sigma^*)$

Exemple

Pour le langage $(b+c)^*a(b+c)^*a(a+b+c)^*$ nous avions donné la grammaire

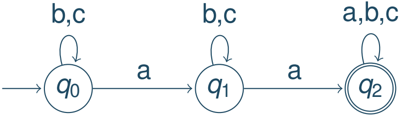
$$\begin{aligned} S &\longrightarrow TaTaT \\ T &\longrightarrow Ta \mid Tb \mid Tc \mid \varepsilon \end{aligned}$$

Cette grammaire est context-free, or le langage est rationnel, nous pouvons proposer une autre grammaire

$$\begin{aligned} S &\longrightarrow bS \mid cS \mid aT \\ T &\longrightarrow bT \mid cT \mid aU \\ U &\longrightarrow aU \mid bU \mid cU \mid \varepsilon \end{aligned}$$

Idée de la construction

$$
\begin{aligned}
S &\longrightarrow bS \mid cS \mid aT \\
T &\longrightarrow bT \mid cT \mid aU \\
U &\longrightarrow aU \mid bU \mid cU \mid \varepsilon
\end{aligned}
$$



Les variables et les états jouent des rôles très similaires en gardant trace de la progression dans la construction/reconnaissance du motif.

Pour le mot *cabbac* par exemple

$$q_0 \xrightarrow{c} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_2 \xrightarrow{c} q_2$$

$$S \xrightarrow{S} cS \xrightarrow{S} caT \xrightarrow{T} cabT \xrightarrow{T} cabbT \xrightarrow{T} cabbaU \xrightarrow{U} cabbacU \xrightarrow{U} cabbac$$

Preuve $Reg \subseteq Rec$ (1/2)

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire régulière à droite. Posons $\mathcal{A} = (\Sigma, V \cup \{F\}, S, \{F\}, \delta)$ un automate fini tel que :

$$\begin{array}{lcl} \delta(T, a) = U & \Leftrightarrow & (T \to aU) \in R \\ \delta(T, a) = F & \Leftrightarrow & (T \to a) \in R \end{array}$$

et montrons que $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{A})$ par récurrence sur la longueur des mots produits par la grammaire :

$$(S \underset{*}{\xrightarrow{\mathcal{G}}} uT) \Leftrightarrow (S \underset{*}{\xrightarrow{u}} T \text{ calcul de } \mathcal{A})$$

Par construction de δ l'équivalence est vraie pour $|u| = 1$. Supposons l'équivalence vraie pour tout u tel que $|u| < n$ et considérons u tel que $|u| = n$ alors $u = va$ avec $a \in \Sigma$ et $|v| < n$.

Dans le sens $\Rightarrow$: soit la dérivation $S \underset{*}{\xrightarrow{\mathcal{G}}} vaT$ comme $\mathcal{G}$ est régulière à droite, il existe une dérivation $S \underset{*}{\xrightarrow{\mathcal{G}}} vU \xrightarrow{\mathcal{G}} vaT$, par HR, $S \underset{*}{\xrightarrow{v}} U$ est donc un calcul de $\mathcal{A}$ par construction de δ comme $U \to aT \in R$, $\delta(U, a) = T$ et donc $S \underset{*}{\xrightarrow{v}} U \xrightarrow{a} T$ est un calcul de $\mathcal{A}$

Preuve *Reg* $\subseteq$ *Rec* (2/2)

Dans le sens $\Leftarrow$: il existe un calcul de $\mathcal{A}$ tel que $S \xrightarrow[*]{u} T$, nous pouvons exhiber la dernière transition $S \xrightarrow[*]{v} U \xrightarrow{a} T$ par HR $S \xrightarrow[*]{\mathcal{G}} vU$ est donc une dérivation. Par construction de δ si $(\delta(U, a) = T) \Leftrightarrow (U \to aT) \in R)$ donc la dérivation se poursuit $S \xrightarrow[*]{\mathcal{G}} vU \xrightarrow{\mathcal{G}} vaT$ soit $S \xrightarrow[*]{\mathcal{G}} uT$

Montrons à présent que $u \in \mathcal{L}(\mathcal{G}) \Leftrightarrow u \in \mathcal{L}(\mathcal{A})$ avec $u = va$. Nous avons montré qu'il existe T tel que

$$S \xrightarrow[*]{v} T \text{ et } S \xrightarrow[*]{\mathcal{G}} vT$$

alors si $u \in \mathcal{L}(\mathcal{G})$, $vT \xrightarrow[*]{\mathcal{G}} va$ et par construction de δ, $\delta(T, a) = F$ et réciproquement

Preuve $Rec \subseteq Reg$

Soit $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ un automate fini déterministe. Posons $\mathcal{G} = (\Sigma, Q, q_0, R)$ telle que :

$$\begin{array}{rcl} R & = & \{T \to aU \mid T, U \in Q, a \in \Sigma, \delta(T, a) = U\} \\ & \cup & \{T \to a \mid T \in Q, a \in \Sigma, \delta(T, a) \in F\} \end{array}$$

Exercice : montrer que $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{A})$.

Grammaires

Grammaires context-free

Définition - Rappel

Grammaires hors contexte (ou context-free ou algébrique) : les règles sont de la forme

$$T \longrightarrow \alpha$$

où $T \in V$, $\alpha \in (\Sigma \cup V)^*$

- une seule variable est réécrite à chaque règle (ici T)
- chaque variable peut être réécrite indépendamment du contexte

i.e. À tout moment d'une dérivation un non-terminal peut être réécrit indépendamment du contexte dans lequel il est placé

Un langage est dit **hors contexte** ou context-free ou algébrique si et seulement si il existe une grammaire hors-contexte qui l'engendre

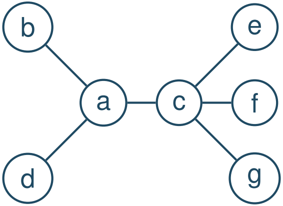
Arbre de dérivation

Les grammaires context-free permettent de dériver un mot de nombreuses manières, nous allons donc utiliser une représentation graphique qui prend en compte le fait que l'ordre des dérivations ne change pas le résultat

Un arbre est un **arbre de dérivation** d'une grammaire context-free $\mathcal{G} = (\Sigma, V, S, R)$ si et seulement si :

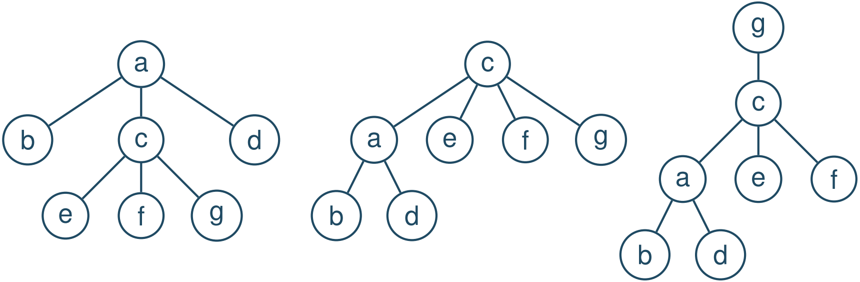
Rappel - Arbre

- Un arbre est un graphe non orienté acyclique et connexe
- Deux sommets sont connectés par un chemin unique
- Un arbre a une arête de moins que son nombre de sommets
- Un arbre est connexe, mais ne l'est plus si on enlève une seule arête, il est acyclique mais ne l'est plus si on ajoute une seule arête



Rappel - Arbre

Un sommet est distingué et appelé **racine**



- Il existe un chemin unique entre la racine R et un nœud N quelconque
- Si on oriente le chemin de R vers N, le prédécesseur de N est appelé père de N, ses successeurs, ses fils
- L'arité d'un nœud est son nombre de fils
- Les nœuds d'arité 0 sont appelés feuilles et les nœuds d'arité > 0 et différents de la racine, nœuds internes

Arbre de dérivation

Les grammaires context-free permettent de dériver un mot de nombreuses manières, nous allons donc utiliser une représentation graphique qui prend en compte le fait que l'ordre des dérivations ne change pas le résultat

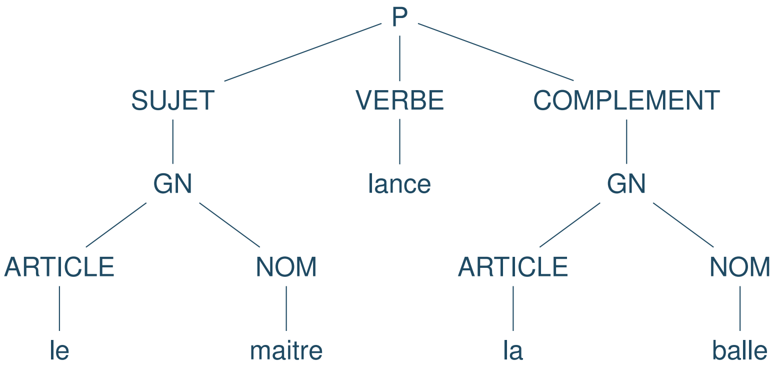
Un arbre est un **arbre de dérivation** d'une grammaire context-free $\mathcal{G} = (\Sigma, V, S, R)$ si et seulement si :

- l'étiquette de la racine est S
- chaque nœud interne est étiqueté par un symbole de V
- chaque feuille est étiquetée par un symbole de Σ
- chaque nœud interne étiqueté par un symbole T a pour fils des symboles $\alpha_1, \ldots, \alpha_n$ (l'ordre est important) si et seulement si $T \to \alpha_1 \ldots \alpha_n \in R$

➥ La concaténation des feuilles de l'arbre de gauche à droite constitue le mot produit

Arbre de dérivation - Exemple Phrase

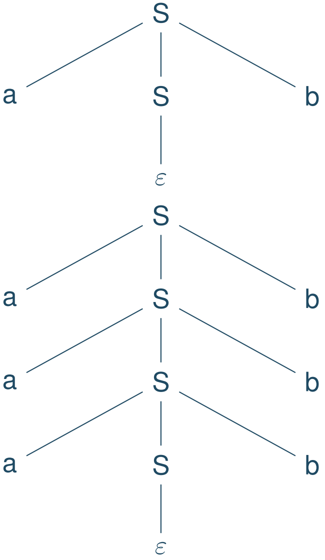
Mot : le maitre lance la balle



Arbre de dérivation - Exemple $a^n b^n$

$$S \longrightarrow aSb \mid \varepsilon$$

$$\begin{array}{lll} S & \longrightarrow & aSb \\ & \longrightarrow & ab \\ S & \longrightarrow & aSb \\ & \longrightarrow & aaSbb \\ & \longrightarrow & aaaSbbb \\ & \longrightarrow & aaabbb \end{array}$$

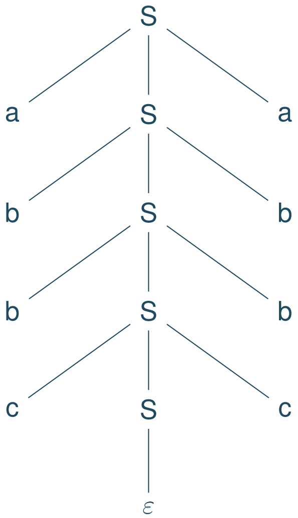


Arbre de dérivation - Exemple palindromes

$$S \longrightarrow aSa \mid bSb \mid cSc \mid a \mid b \mid c \mid \varepsilon$$

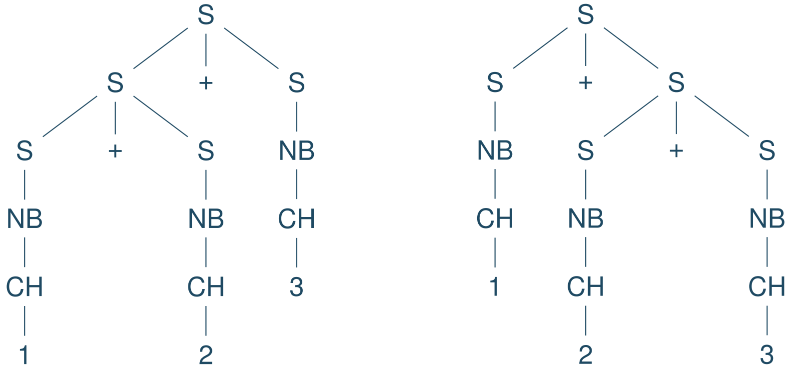
Dérivation

$$\begin{aligned} S &\longrightarrow aSa \\ &\longrightarrow abSba \\ &\longrightarrow abbSbba \\ &\longrightarrow abbcScbba \\ &\longrightarrow abbccbba \end{aligned}$$



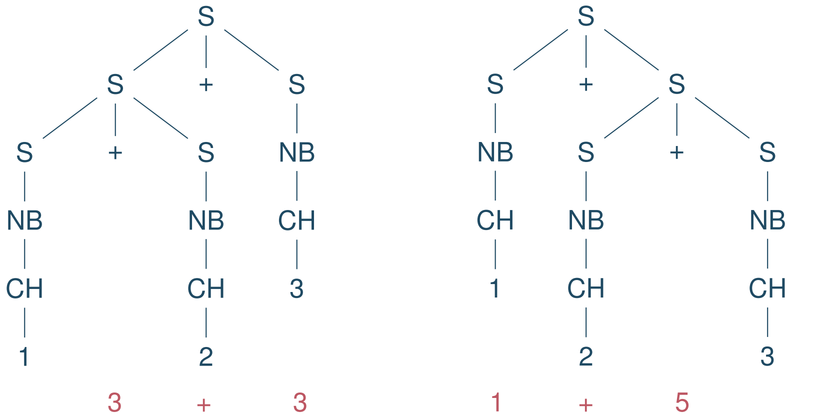
Ambiguïté - Exemple

$$S \longrightarrow S + S \mid NB$$
$$NB \longrightarrow NB\ CH \mid CH$$
$$CH \longrightarrow 0 \mid 1 \mid \cdots \mid 9$$

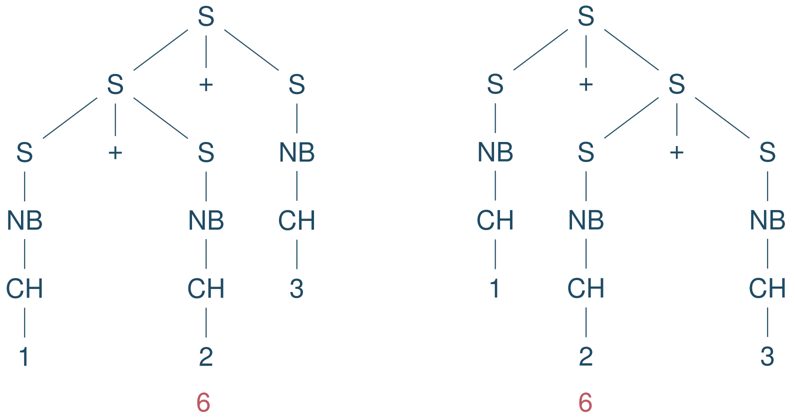


Ambiguïté - Exemple

$$S \longrightarrow S + S \mid NB$$
$$NB \longrightarrow NB\ CH \mid CH$$
$$CH \longrightarrow 0 \mid 1 \mid \cdots \mid 9$$

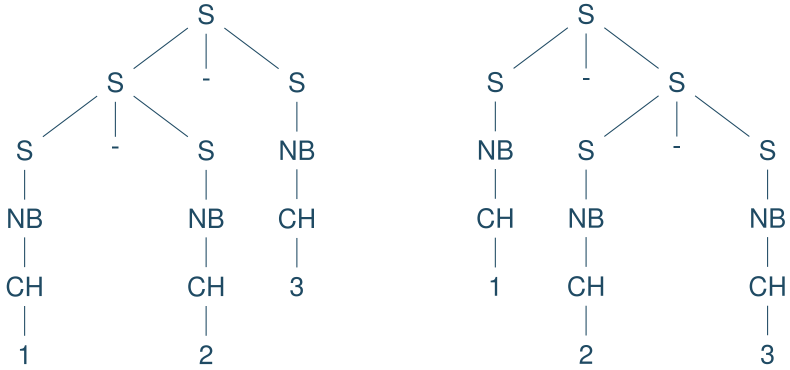


$$S \longrightarrow S + S \mid NB$$
$$NB \longrightarrow NB\ CH \mid CH$$
$$CH \longrightarrow 0 \mid 1 \mid \cdots \mid 9$$



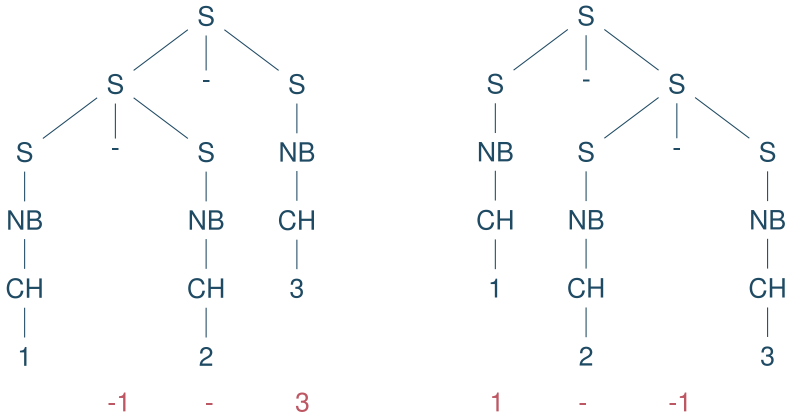
Ambiguïté - Gênant ?

$$S \longrightarrow S + S \mid NB$$
$$NB \longrightarrow NB\ CH \mid CH$$
$$CH \longrightarrow 0 \mid 1 \mid \cdots \mid 9$$



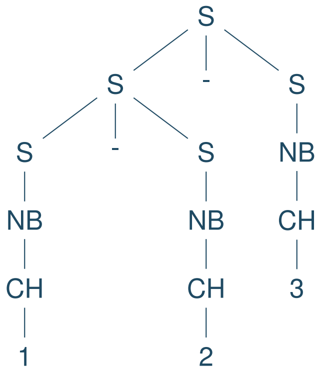
Ambiguïté - Gênant ?

$$S \longrightarrow S + S \mid NB$$
$$NB \longrightarrow NB\ CH \mid CH$$
$$CH \longrightarrow 0 \mid 1 \mid \cdots \mid 9$$

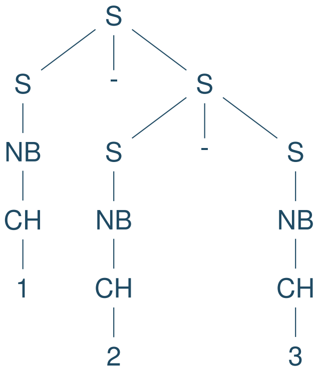


Ambiguïté - Gênant ?

$$S \longrightarrow S + S \mid NB$$
$$NB \longrightarrow NB\ CH \mid CH$$
$$CH \longrightarrow 0 \mid 1 \mid \cdots \mid 9$$



-4



2

Dérivations gauche et droite

Une dérivation **gauche** d'une grammaire hors contexte est une dérivation dans laquelle chaque pas de dérivation réécrit le non-terminal le plus à gauche

Une dérivation **droite** d'une grammaire hors contexte est une dérivation dans laquelle chaque pas de dérivation réécrit le non-terminal le plus à droite

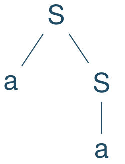
Dans l'exemple précédent :

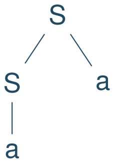
- dérivation gauche :
  $S \to S + S \to S + S + S \to NB + S + S \to CH + S + S \to 1 + S + S \to 1{+}NB{+}S \to 1{+}CH{+}S \to 1{+}2{+}S \to 1{+}2{+}NB \to 1{+}2{+}CH \to 1{+}2{+}3$
- dérivation droite :
  $S \to S + S \to S + S + S \to S + S + NB \to S + S + CH \to S + S + 3 \to S{+}NB{+}3 \to S{+}CH{+}3 \to S{+}2{+}3 \to NB{+}2{+}3 \to CH{+}2{+}3 \to 1{+}2{+}3$

Ambiguïté

Une grammaire est **ambigüe** s'il existe un mot qui admet plusieurs arbres de dérivation

$S \to aS + Sa + a$





L'ambiguïté est relative à la grammaire : une grammaire ambigüe peut engendrer un langage qui peut également être engendré par une autre grammaire non ambigüe : $S \to aS + a$

Langage ambigu

➥ L'ambiguïté est fort embarrassante lors de l'analyse d'un langage de programmation par exemple

Un **langage ambigu** est un langage qui n'est engendré par aucune grammaire non ambigüe

Exemple

$\{a^n b^n c^m \mid m, n \geq 0\} \cup \{a^m b^n c^n \mid m, n \geq 0\}$ est un langage ambigu

⚠ Il n'existe pas d'algorithme permettant de décider si une grammaire est ambigüe

Exemple de langage - Langage de Dyck

Le langage de Dyck est le langage des mots « bien parenthésés » avec n type de parenthèses, $\Sigma = \{a_1, a'_1, \ldots, a_n, a'_n\}$, le langage de $Dyck_n$ sur Σ est engendré par :

$$\begin{array}{rcl} S & \longrightarrow & ST \mid \varepsilon \\ T & \longrightarrow & a_1 S a'_1 \mid \cdots \mid a_n S a'_n \end{array}$$

$$\begin{array}{rcl} S & \longrightarrow & ST \longrightarrow Sa_1Sa'_1 \longrightarrow STa_1Sa'_1 \longrightarrow STa_1STa'_1 \longrightarrow Sa_1Sa'_1a_1STa'_1 \\ & \longrightarrow & Sa_1Sa'_1a_1Sa_2Sa'_2a'_1 \longrightarrow Sa_1STa'_1a_1Sa_2Sa'_2a'_1 \\ & \longrightarrow & Sa_1Sa_3Sa'_3a'_1a_1Sa_2Sa'_2a'_1 \longrightarrow Sa_1Sa_3Sa'_3a'_1a_1Sa_2Sa'_2a'_1 \\ & \longrightarrow^* & a_1a_3a'_3a'_1a_1a_2a'_2a'_1 \end{array}$$

Ce langage est algébrique, est-il rationnel ?

$Dyck_1 \cap a_1^* a_1'^* = \{a_1^n a_1'^n \mid n \in \mathbb{N}\}$

Propriétés de clôture

Proposition

Les langages algébriques sont clos par les opérations rationnelles (union, concaténation, étoile)

Construction

Soient $\mathcal{G} = (\Sigma, V, S, R)$ et $\mathcal{G}' = (\Sigma, V', S', R')$ deux grammaires algébriques (avec $V \cap V' = \emptyset$)

- $\mathcal{L}(\mathcal{G}) \cup \mathcal{L}(\mathcal{G}')$ est engendré par la grammaire $(\Sigma, V \cup V' \cup \{S''\}, S'', R \cup R' \cup \{S'' \to S + S'\})$
- $\mathcal{L}(\mathcal{G})\mathcal{L}(\mathcal{G}')$ est engendré par la grammaire $(\Sigma, V \cup V' \cup \{S''\}, S'', R \cup R' \cup \{S'' \to SS'\})$
- $\mathcal{L}(\mathcal{G})^*$ est engendré par la grammaire $(\Sigma, V \cup \{S''\}, S'', R \cup \{S'' \to S''S + \varepsilon\})$

Grammaires

Transformation de Grammaires context-free

Grammaire réduite

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire, pour tout $T \in V$ on note $\mathcal{L}_{\mathcal{G}}(T) = \mathcal{L}(\mathcal{G}')$ avec $\mathcal{G}' = (\Sigma, V, T, R)$

Une grammaire est dite **réduite** si

- $\forall T \in V, \mathcal{L}_{\mathcal{G}}(T) \neq \emptyset$ ➥ toute variable peut engendrer un mot
- $\forall T \in V, \exists \alpha, \beta \in (\Sigma \cup V)^*$ tels que $S \xrightarrow{*} \alpha T \beta$
  ➥ toute variable est atteignable à partir de S

➥ Il n'y a pas de variable trivialement inutile

« Réduire » une grammaire

Proposition. Pour toute grammaire $\mathcal{G} = (\Sigma, V, S, R)$, il existe une grammaire $\mathcal{G}' = (\Sigma, V', S', R')$ réduite telle que $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$

Algorithme.

1. Calculer l'ensemble U des symboles de V qui peuvent engendrer un mot ➥ Si $S \notin U$, la grammaire ne génère aucun mot
2. Supprimer les symboles de $V \setminus U$ ainsi que toutes les règles dans lesquelles ils apparaissent
3. Calculer l'ensemble W des symboles de U atteignables
4. Supprimer les symboles de $U \setminus W$ ainsi que toutes les règles dans lesquelles ils apparaissent

« Réduire » une grammaire - Symboles productifs

i.e. les symboles qui peuvent engendrer un mot

Construisons l'ensemble par itérations :

- $U_0 = \Sigma$
- $\forall i, U_{i+1} = U_i \cup \{T \in V \mid T \to \alpha \text{ avec } \alpha \in U_i^*\}$
- Arrêter quand $U_{i+1} = U_i$. Noter que comme les U_i forment une suite croissante et que $\Sigma \cup V$ est fini, la suite est forcément constante à partir d'un certain rang
- Alors $U = U_i$

« Réduire » une grammaire - Symboles accessibles

Construisons l'ensemble par itérations :

- $W_0 = \{S\}$
- $\forall i, W_{i+1} = W_i \cup \{T \in V \mid \exists T' \in W_i, T' \to \alpha T \beta$ avec $\alpha, \beta \in (\Sigma \cup V)^*\}$
- Arrêter quand $W_{i+1} = W_i$. Noter que comme les W_i forment une suite croissante et que V est fini, la suite est forcément constante à partir d'un certain rang
- Alors $W = W_i$

« Réduire » une grammaire - Exemple

$$\begin{array}{lcl} S & \longrightarrow & ST \mid a \mid \varepsilon \\ T & \longrightarrow & aTa \mid TX \\ X & \longrightarrow & b \end{array}$$

- Symboles productifs : $U_0 = \{a, b\}$, $U_1 = \{a, b, S, X\}$, $U_2 = \{a, b, S, X\} = U_1$
- On obtient

$$\begin{array}{lcl} S & \longrightarrow & a \mid \varepsilon \\ X & \longrightarrow & b \end{array}$$

- Symboles accessibles : $W_0 = \{S\}$, $W_1 = \{S\} = W_0$
- On obtient la grammaire réduite

$$S \longrightarrow a \mid \varepsilon$$

Grammaire propre

Une grammaire algébrique est **propre** si elle ne contient pas :

- de règle $T \to U$ avec $T, U \in V$
- de règle $T \to \varepsilon$

Proposition

Tout langage algébrique ne contenant pas le mot vide peut être engendré par une grammaire propre

Remarque. Si on veut considérer un langage algébrique qui contient le mot vide on peut autoriser $S \to \varepsilon$, si S n'apparaît jamais en partie droite d'une règle

Rendre une grammaire propre - mot vide

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique, il existe une grammaire $\mathcal{G}' = (\Sigma, V', S', R')$ propre telle que $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$

Algorithme

1. Calculer l'ensemble U des symboles de V qui peuvent se dériver en ε
   1.1 $U_0 = \{\varepsilon\}$
   1.2 $\forall i, U_{i+1} = U_i \cup \{T \in V \mid T \to \alpha \text{ avec } \alpha \in U_i^*\}$
   1.3 Arrêter quand $U_{i+1} = U_i = U$
2. Pour toute règle $T \to \alpha X \beta$ avec $X \in U$ ajouter $T \to \alpha\beta$
3. Supprimer toutes les règles $T \to \varepsilon$

Rendre une grammaire propre - variables équivalentes

Il faut à présent supprimer les règles de type $T \to U$ avec $T, U \in V$. Comme aucune variable n'engendre le mot vide, $T \underset{*}{\to} U$ si et seulement si il existe $T = T_0 \to T_1 \to \ldots \to T_n = U$

Idée. Remplacer toute déviration de type $T \underset{*}{\to} U \to \alpha$ avec $\alpha \notin V$ par une dérivation $T \to \alpha$.

1. Supprimer les règles $T \to U$ avec $T, U \in V$
2. Ajouter $T \to \alpha$, si $T \underset{*}{\to} U$ et $U \to \alpha$ avec $T, U \in V$ et $\alpha \notin V$

Algorithme

- Calculer les paires de symboles de V telles que $T \underset{*}{\to} U$ et $U \underset{*}{\to} T$, ce qui produit des classes d'équivalence de symboles de V. Remplacer tous les symboles d'une classe par un représentant choisi parmi eux
- Supprimer les règles de type $T \to T$
- Pour chaque règle de type $T \to U$, et chaque règle $U \to \alpha$, on ajoute $T \to \alpha$, puis on supprime $T \to U$

Rendre une grammaire propre - Exemple (1/3)

$$\begin{array}{lcl} S & \longrightarrow & aTU \mid UT \mid b \\ T & \longrightarrow & UU \mid a \\ U & \longrightarrow & TU \mid b \mid bX \mid \varepsilon \\ X & \longrightarrow & Y \\ Y & \longrightarrow & aa \end{array}$$

Calculons l'ensemble U des symboles de V qui peuvent se dériver en ε :
$U_0 = \{\varepsilon\}$, $U_1 = \{U\}$, $U_2 = \{U, T\}$, $U_3 = \{U, T, S\}$ Donc

$$\begin{array}{lcl} S & \longrightarrow & aTU \mid UT \mid b \mid aU \mid aT \mid U \mid T \\ T & \longrightarrow & UU \mid a \mid U \\ U & \longrightarrow & TU \mid b \mid bX \mid U \mid T \\ X & \longrightarrow & Y \\ Y & \longrightarrow & aa \end{array}$$

Calculons les classes d'équivalences de symboles : $\{S\}$, $\{T, U\}$, $\{X\}$, $\{Y\}$

Rendre une grammaire propre - Exemple (2/3)

Choisissons T :

$$
\begin{array}{lcl}
S & \longrightarrow & aTT \mid TT \mid b \mid aT \mid aT \mid T \mid T \\
T & \longrightarrow & TT \mid a \mid T \\
T & \longrightarrow & TT \mid b \mid bX \mid T \mid T \\
X & \longrightarrow & Y \\
Y & \longrightarrow & aa
\end{array}
$$

Supprimons les redondances

$$
\begin{array}{lcl}
S & \longrightarrow & aTT \mid TT \mid b \mid aT \mid T \\
T & \longrightarrow & TT \mid a \mid T \mid b \mid bX \\
X & \longrightarrow & Y \\
Y & \longrightarrow & aa
\end{array}
$$

Rendre une grammaire propre - Exemple (3/3)

Supprimons les règles de type $T \to T$

$$\begin{array}{lcl} S & \longrightarrow & aTT \mid TT \mid b \mid aT \mid T \\ T & \longrightarrow & TT \mid a \mid b \mid bX \\ X & \longrightarrow & Y \\ Y & \longrightarrow & aa \end{array}$$

Supprimons les règles de type $T \to U$

1. Supprimons $S \to T$

$$\begin{array}{lcl} S & \longrightarrow & aTT \mid TT \mid b \mid aT \mid a \mid bX \\ T & \longrightarrow & TT \mid a \mid b \mid bX \\ X & \longrightarrow & Y \\ Y & \longrightarrow & aa \end{array}$$

2. Supprimons $X \to Y$

$$\begin{array}{lcl} S & \longrightarrow & aTT \mid TT \mid b \mid aT \mid a \mid bX \\ T & \longrightarrow & TT \mid a \mid b \mid bX \\ X & \longrightarrow & aa \end{array}$$

Forme normale de Chomsky

Une grammaire $\mathcal{G} = (\Sigma, V, S, R)$ est en **forme normale de Chomsky** si toute règle est de la forme :

$$\begin{array}{lcll} T & \to & UW & \text{avec } T, U, W \in V \\ T & \to & a & \text{avec } T \in V, a \in \Sigma \end{array}$$

Remarques

- la grammaire est donc propre
- la grammaire ne peut engendrer le mot vide
- une variante autorise la règle $S \to \varepsilon$ si S n'apparait jamais en partie droite
- l'arbre de dérivation est donc un arbre binaire

Mettre en forme normale de Chomsky

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique réduite et propre

La transformation s'effectue en 2 étapes

- $\forall x \in \Sigma$, ajouter une variable V_x et une règle $V_x \to x$, puis remplacer x par V_x dans toutes les règles de R

Toutes les règles sont désormais de la forme $T \to T_1 \ldots T_n$ ou $T \to x$

- Pour chaque règle $T \to T_1 \ldots T_n$ avec $n \geq 2$ ajouter de nouvelles variables $U_1 \ldots U_{n-2}$ et remplacer la règle par

$$\begin{array}{lcl} T & \to & T_1 U_1 \\ U_1 & \to & T_2 U_2 \\ U_2 & \to & T_3 U_3 \\ \ldots & & \\ U_{n-2} & \to & T_{n-1} T_n \end{array}$$

Décider si un mot appartient à un langage algébrique

Soient $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique en forme normale de Chomsky et $u \in \Sigma^*$. Il est possible de **décider en temps fini** si u appartient à $\mathcal{L}(\mathcal{G})$

Comment ? La partie droite α de toute règle de R est telle que soit $\alpha \in \Sigma$ soit $\alpha \in V^2$ donc une dérivation qui engendre le mot u a une longueur d'au plus $2|u|$, il suffit donc d'énumérer toutes les dérivations possibles pour tester si u est engendré par la grammaire

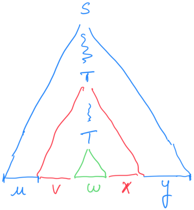
Grammaires

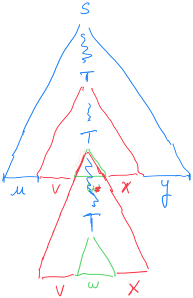
Lemme d’itération

Lemme d'itération

Soit $\mathcal{L}$ un langage context-free, il existe un entier k tel que tout mot $u \in \mathcal{L}$ avec $|u| > k$ se factorise en $u = vwxyz$ avec $wy \neq \varepsilon$, $|wxy| < k$ et pour tout n, $vw^nxy^nz \in \mathcal{L}$

Lemme d’itération - Idée





Montrer qu'un langage n'est pas context-free

Soit $\mathcal{L} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$, supposons qu'il est context-free

Soient k l'entier donné par le lemme d'itération et u tel que $|u| > k$, alors u se factorise en $u = vwxyz$ avec $u_n = vw^n xy^n z \in \mathcal{L}$ pour tout n $wy \neq \varepsilon$, $|wxy| < k$

Si w ou y sont composés d'au moins 2 lettres différentes de $\{a, b, c\}$, alors clairement les u_n ne peuvent appartenir à $\mathcal{L}$ puisque la répétition des w et des y introduit une alternance de lettres impossible dans le langage

Donc w et y appartiennent à $a^* + b^* + c^*$, or par hypothèse, $vwxyz$ contient autant d'occurrences de a, de b et de c, il est donc impossible que $vw^n xy^n z$ dans lequel on a modifié le nombre d'occurences de une ou deux des trois lettres contienne toujours le même nombre d'occurrences des trois lettres

Contradiction