

But de ce cours

	outils pour l'analyse lexicale	outils pour l'analyse syntaxique
type de langage	langage régulier	langage algébrique
description	expressions régulières	grammaires algébriques
formalisme sous-jacent	AF(N)D	automates à pile

Langages et grammaires algébriques

Limites des langages réguliers

Langages et grammaires algébriques

Arbres syntaxiques

Grammaires « pathologiques »

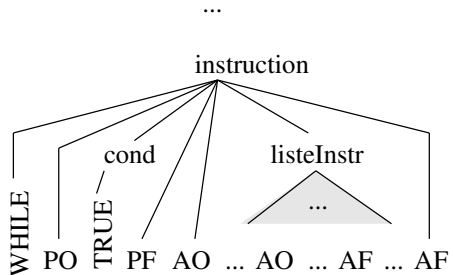
Les algébriques... et les autres

Principes de l'analyse syntaxique

- ▶ reçoit de l'analyseur lexical une suite de symboles ;
- ▶ reconnaît dans cette suite la structure d'un texte.

Ex :

```
...  
while (true) {  
    while (true) {  
        ...  
    }  
}  
...  
...
```



Décrire/reconnaître la structure d'un texte

Les langages réguliers / expressions régulières ne suffisent pas.

exemple archi-typique : $\{a^n b^n | n \geq 0\}$

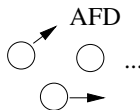
- ▶ parenthésage imbriqué
- ▶ pour le reconnaître :
 - ▶ compter/mémoriser le nombre de a ;
 - ▶ compter /mémoriser le nombre de b .
- ▶ pas régulier.

$a^n b^n$ pas régulier, argument intuitif - 1

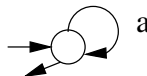
Dans un AF(fin)D : états = mémoire (finie)

Ex langages réguliers finis :

- ▶ $\{a^n b^n \mid n \leq 3\}$;
- ▶ $\{a^n b^n \mid n \leq 9045\}$;
- ▶ $\{a^n b^n \mid n \leq k\}$ pour k fixé ;



Ex langage régulier infini : a^*



$a^n b^n$ pas régulier, argument intuitif - 2

$$\{a^n b^n \mid n \geq 0\}$$

~~AFD mémoire finie~~

⇒ il suffit de prendre n plus grand que la taille de la mémoire.

$a^n b^n$ pas régulier, argument intuitif - 3

Pour reconnaître $\{a^n b^n \mid n \geq 0\}$:

- ▶ mémoriser un nombre non borné de symboles a et b ;
- ▶ \Rightarrow langage non régulier.

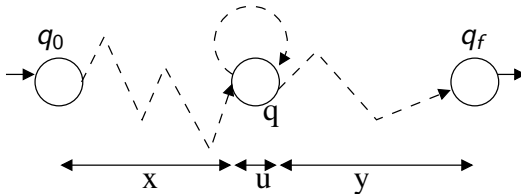
Pour reconnaître $\{a^n b^n \mid n \leq 32\}$:

- ▶ il faut mémoriser au pire 32 a et 32 b ;
- ▶ mémorisation bornée \Rightarrow régulier.

$a^n b^n$ pas régulier, argument formel

Theorem (Lemme de pompage)

Soit L un langage *régulier*. Alors il existe un entier k tq pour tout mot m de L de longueur $\geq k$, il existe des mots $x, u, y \in \Sigma^*$ avec $m = xuv$, $u \neq \epsilon$ et pour tout n , $xu^n y \in L$.



Pour $a^n b^n$: où placer u ?

Limites des langages réguliers

Langages et grammaires algébriques

Les grammaires algébriques

Langage engendré, dérivations

Langages algébriques, comparaison avec les réguliers

Outillage

Arbres syntaxiques

Grammaires « pathologiques »

Les algébriques... et les autres

Grammaire algébrique pour INIT

programme \rightarrow entete listDecl listInst

entete \rightarrow PROG ID PV

listInst $\rightarrow \epsilon$

listInst \rightarrow instr listInst

instr \rightarrow lecture PV

instr \rightarrow affect PV

lecture \rightarrow READ ID

...

TERMINAUX $\in V_T$

non-terminaux $\in V_N$

axiome $\in V_N$

une production

une production vide

Grammaire algébrique pour INIT

programme \rightarrow entete listDecl listInst

entete \rightarrow PROG ID PV

listInst $\rightarrow \epsilon$

listInst \rightarrow instr listInst

instr \rightarrow lecture PV | affect PV

// équivalent !

lecture \rightarrow READ ID

...

TERMINAUX $\in V_T$

non-terminaux $\in V_N$

axiome $\in V_N$

une production

une production vide

On écrit $X \rightarrow \alpha \mid \beta$ pour

$X \rightarrow \alpha$

$X \rightarrow \beta$

Définition formelle

Definition (grammaire algébrique)

Une grammaire **algébrique** ou **hors-contexte** est un quadruplet

$$G = (V_T, V_N, P, S)$$

- ▶ V_T et V_N sont des vocabulaires disjoints : V_T est l'ensemble des **terminaux**, V_N l'ensemble des **non-terminaux** ;
- ▶ $S \in V_N$ est l'**axiome**, ou symbole de départ (**Start**) ;
- ▶ $P \subseteq V_N \times (V_N \cup V_T)^*$ est l'ensemble des (**règles de productions**).

Remarques

« $P \subseteq V_N \times (V_N \cup V_T)^*$ est l'ensemble des productions » :

instr \rightarrow affect PV
est une notation pour
(instr, affect PV) $\in P$

Terminaux de $V_T =$ unités lexicales fixées à l'analyse lexicale.

La grammaire engendre des mots de V_T^* .

Comment engendrer des mots ?

Problème : engendrer des mots de V_T^* à partir :

- ▶ de l'axiome ;
- ▶ et des productions de la grammaire.

Ex : engendrer le mot PROG ID PV à partir de G_I :

- | | |
|---|-------------------------------------|
| (1) programme \rightarrow entete listDecl listInst | (3) listDecl $\rightarrow \epsilon$ |
| (2) entete \rightarrow PROG ID PV | (4) listInst $\rightarrow \epsilon$ |

Moyen : **dérivations directes** successives, à partir de l'axiome.

Comment engendrer des mots - dérivations directes

- (1) **programme** \rightarrow entete listDecl listInst (3) listDecl $\rightarrow \epsilon$
(2) entete \rightarrow PROG ID PV (4) listInst $\rightarrow \epsilon$

programme \Rightarrow_{G_I} **entete listDecl listInst** (dér. directe par (1))

entete listDecl listInst \Rightarrow_{G_I} **PROG ID PV** listDecl listInst par (2)

PROG ID PV listDecl listInst \Rightarrow_{G_I} PROG ID PV listDecl par (4)

PROG ID PV listDecl \Rightarrow_{G_I} PROG ID PV par (3)

Dérivation : définition

Une **dérivation** est une suite de dérivations directes :

programme \Rightarrow_{G_I} entete listDecl listInst
 \Rightarrow_{G_I} PROG ID PV listDecl listInst \Rightarrow_{G_I} PROG ID PV listDecl
 \Rightarrow_{G_I} PROG ID PV

Cette dérivation est de **longueur** 4 : programme $\Rightarrow_{G_I}^4$ PROG ID PV

Notation : programme $\Rightarrow_{G_I}^*$ PROG ID PV

Pour une grammaire G d'axiome S , une **dérivation pour le mot m** désigne une **dérivation de S à m** .

Langage engendré par une grammaire

Definition (langage engendré)

Soit G une grammaire algébrique d'axiome S . Le **langage engendré** par G est défini par :

$$L(G) = \{m \in V_T^* \mid S \Rightarrow^* m\}$$

Déterminer si un mot m appartient au langage engendré, c'est déterminer si :

$$S \Rightarrow_G^* m?$$

Autre exemple - une grammaire des additions - 1

$G_A = (V_T, V_N, P, A)$ où

- ▶ $V_T = \{\text{id}, +\}$
- ▶ $V_N = \{ A \}$
- ▶ $P = \{ A \rightarrow A + A \mid \text{id} \}$

Autre exemple - une grammaire des additions - 2

$$A \rightarrow A + A \mid \text{id}$$

Une dérivation possible pour le mot $\text{id} + \text{id} + \text{id}$:

$$\underline{A} \Rightarrow \underline{A} + \underline{A} \Rightarrow A + \underline{A} + A \Rightarrow \underline{A} + \text{id} + A \Rightarrow \text{id} + \text{id} + \underline{A} \Rightarrow \text{id} + \text{id} + \text{id}$$

Donc $A \Rightarrow^* \text{id} + \text{id} + \text{id}$, ou $A \Rightarrow^5 \text{id} + \text{id} + \text{id}$

Mon voisin a trouvé une dérivation différente !

On peut souvent trouver plusieurs dérivations pour un mot.

Deux sources de choix :

- ▶ choix du non-terminal à dériver ;
- ▶ choix de la production à appliquer, de membre gauche le non-terminal choisi.

Dérivation directe (formellement) - 1

Definition (dérivation directe)

Soient $\beta, \beta' \in (V_N \cup V_T)^*$. β se **dérive directement** en β' selon G , noté $\beta \Rightarrow_G \beta'$, s'il existe des mots $\gamma, \gamma' \in (V_N \cup V_T)^*$ et une production $X \rightarrow \alpha$ tels que :

$$\begin{aligned}\beta &= \gamma X \gamma' \\ \beta' &= \gamma \alpha \gamma'\end{aligned}$$

Ex :
$$\overbrace{A + \underline{A} + A}^{\beta} \Rightarrow \overbrace{A + \text{id} + A}^{\beta'} \text{ par } \overbrace{A}^X \rightarrow \overbrace{\text{id}}^{\alpha}$$

$$\underbrace{A}_{\gamma} + \underbrace{A}_X + \underbrace{A}_{\gamma'} \Rightarrow \underbrace{A}_{\gamma} + \underbrace{\text{id}}_{\alpha} + \underbrace{A}_{\gamma'}$$

Dérivation directe (formellement) - 2

Definition (dérivation directe)

Soient $\beta, \beta' \in (V_N \cup V_T)^*$. β se *dérive directement* en β' selon G , noté $\beta \Rightarrow_G \beta'$, s'il existe des mots $\gamma, \gamma' \in (V_N \cup V_T)^*$ et une production $X \rightarrow \alpha$ tels que :

$$\begin{aligned}\beta &= \gamma X \gamma' \\ \beta' &= \gamma \alpha \gamma'\end{aligned}$$

Ex : $\overbrace{A + \underline{A}}^{\beta} \Rightarrow \overbrace{A + \textcolor{blue}{A} + \textcolor{blue}{A}}^{\beta'}$ par $\overbrace{A}^X \rightarrow \overbrace{A + A}^{\alpha}$

$$\underbrace{A +}_{\gamma} \underbrace{A}_X \underbrace{\quad}_{\gamma'=\epsilon} \Rightarrow \underbrace{A +}_{\gamma} \underbrace{A + A}_{\alpha} \underbrace{\quad}_{\gamma'=\epsilon}$$

Encore un exemple, et un piège

$V_T = \{a, b\}$, $V_N = \{S, B\}$, axiome S ,
 $P = \{S \rightarrow BB, B \rightarrow a \mid b\}$.

Dérivation pour aa ?

De longueur 3, pas 2 !

Dérivation (formellement)

Definition (dérivation)

Soient $\beta, \beta' \in (V_N \cup V_T)^*$. Une suite de mots $\beta_0, \beta_1, \dots, \beta_n$ ($n \geq 0$) est une *dérivation* de β en β' selon G si $\beta_0 = \beta$, $\beta_n = \beta'$, et

$$\beta_0 \Rightarrow \beta_1 \Rightarrow \dots \Rightarrow \beta_n$$

Definition (relation de dérivation)

La **relation de dérivation** \Rightarrow_G^* est la *fermeture réflexive et transitive* de la relation de dérivation directe \Rightarrow_G .

Mon voisin a trouvé une grammaire différente !

On peut trouver 2 grammaires algébriques qui engendrent le même langage.

Definition (grammaires équivalentes)

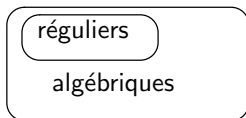
Deux grammaires sont **équivalentes** si elles engendrent le même langage.

Langages algébriques

Definition (langage algébrique)

Les langages engendrés par les grammaires algébriques sont appelés **langages algébriques**.

Comparaison avec les réguliers



régulier \Rightarrow algébrique
algébrique \nRightarrow régulier

Tout langage régulier **est** algébrique :

- ▶ possible d'utiliser une grammaire algébrique au lieu d'expressions régulières ;
- ▶ mais AF plus efficaces.

Il existe des algébriques qui ne sont **pas** réguliers : $\{a^n b^n | n \geq 0\}$

Un exemple pour la route

Une première grammaire des expressions arithmétiques :

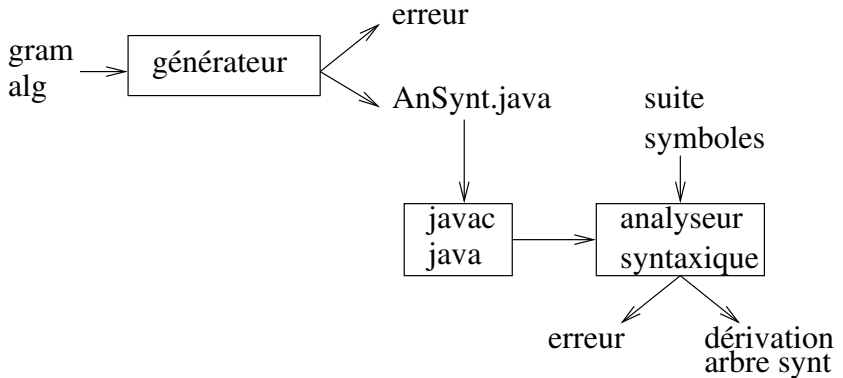
$G_E = (V_T, V_N, P, E)$:

- ▶ $V_T = \{\text{id}, +, *, (,)\}$
- ▶ $V_N = \{ E \}$
- ▶ $P = \{ E \rightarrow E + E \mid E * E \mid (E) \mid \text{id} \}$

Montrer que $\text{id} * (\text{id} + \text{id})$ est un mot de $L(G_E)$.

Langage algébrique ? régulier ?

Générateurs d'analyseurs syntaxiques



Limites des langages réguliers

Langages et grammaires algébriques

Arbres syntaxiques

Grammaires « pathologiques »

Les algébriques... et les autres

Arbre syntaxique

Arbre résultant de l'analyse syntaxique.

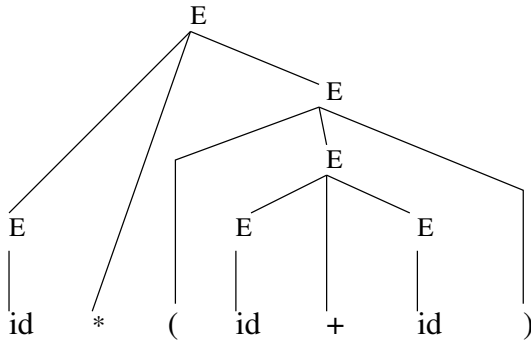
Fait apparaître la **structure syntaxique** d'un mot.

Notion très importante, on le verra plus tard.

On parle aussi d'**arbre de dérivation**.

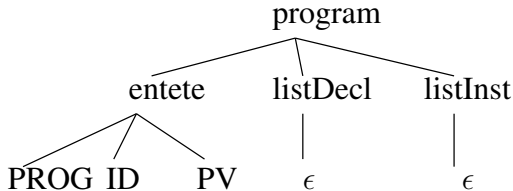
Arbre syntaxique : exemple - 1

Arbre syntaxique pour $\text{id} * (\text{id} + \text{id})$ selon G_E :



Arbre syntaxique : exemple - 2

Arbre syntaxique pour `PR0G ID PV` selon G_I :

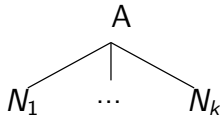


Schématiquement

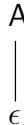
Pour une grammaire G :

- ▶ la racine : l'axiome de G ;
- ▶ le mot des feuilles : le mot qui nous intéresse ;
- ▶ les noeuds internes sont agencés selon les productions de G :

$$A \rightarrow N_1 N_2 \dots N_k$$



$$A \rightarrow \epsilon$$



Arbre syntaxique et appartenance à $L(G)$

Theorem

Soit $m \in V_T^*$. $m \in L(G)$ ss'il existe un arbre syntaxique selon G dont le *mot aux feuilles* est m .

On dit alors que cet arbre est un arbre syntaxique pour m , ou que m admet cet arbre.

Répondre à $m \in L(G) =$ chercher un arbre syntaxique pour m .

En pratique

Les analyseurs syntaxiques ne construisent pas un arbre syntaxique.

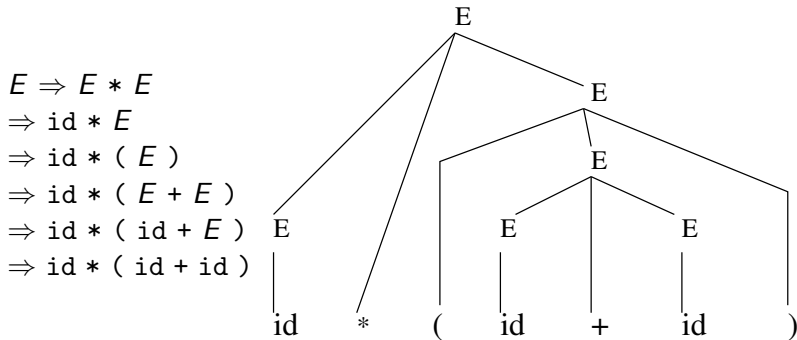
L'arbre est une représentation commode pour comprendre l'analyse syntaxique.

Ils construisent une dérivation particulière, appelée **gauche** ou **droite**.

⇒ lien entre arbre syntaxique et dérivation ?

Arbre syntaxique et dérivations - 1

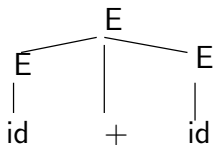
Facile de construire un arbre syntaxique à partir d'une dérivation.



À une dérivation correspond un seul arbre.

Arbre syntaxique et dérivations - 2

À un arbre correspondent potentiellement **plusieurs dérivations**.



$$\begin{aligned}
 E &\Rightarrow_{G_1} \underline{E} + E \Rightarrow_{G_1} \text{id} + E \\
 &\Rightarrow_{G_1} \text{id} + \text{id}
 \end{aligned}$$

$$\begin{aligned}
 E &\Rightarrow_{G_1} E + \underline{E} \Rightarrow_{G_1} E + \text{id} \\
 &\Rightarrow_{G_1} \text{id} + \text{id}
 \end{aligned}$$

Ces 2 dérivations sont « équivalentes » : elles correspondent au **même arbre** :

- ▶ mêmes productions appliquées aux mêmes non- t^{aux} ;
- ▶ seul l'**ordre des dérivations directes** varie.

Arbre syntaxique et dérivations : dérivations gauche/droite

On peut fixer l'ordre des dérivations directes en choisissant systématiquement :

- ▶ soit le non-terminal le plus à gauche ;

$E \Rightarrow_{G_I} \underline{E} + E \Rightarrow_{G_I} \text{id} + \underline{E} \Rightarrow_{G_I} \text{id} + \text{id}$ dérivation gauche

- ▶ soit le non-terminal le plus à droite.

$E \Rightarrow_{G_I} E + \underline{E} \Rightarrow_{G_I} \underline{E} + \text{id} \Rightarrow_{G_I} \text{id} + \text{id}$ dérivation droite

À chaque arbre syntaxique correspond une **unique** dérivation gauche et une unique dérivation droite.

Dérivation gauche/droite, formellement

Definition

Soient $\beta, \beta' \in (V_N \cup V_T)^*$ et $\beta_0 \Rightarrow \beta_1 \Rightarrow \dots \Rightarrow \beta_n$ une dérivation de β en β' . Si à chaque étape on choisit de remplacer dans β_i le non-terminal :

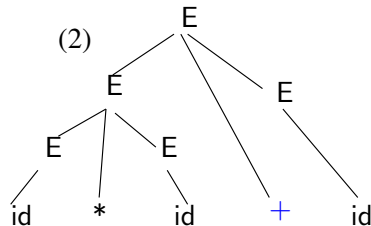
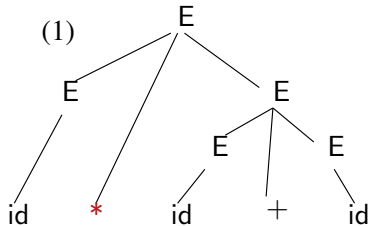
- ▶ le plus à gauche : c'est une **dérivation gauche** (leftmost derivation) notée $\beta \xRightarrow{lm}^* \beta'$;
- ▶ le plus à droite : c'est une **dérivation droite** (rightmost derivation) notée $\beta \xRightarrow{rm}^* \beta'$.

Arbre syntaxique et dérivations - 3

Certaines dérivations **ne correspondent pas** au même arbre.

(1) $E \Rightarrow E * \underline{E} \Rightarrow E * E + E \Rightarrow^* \text{id} * \text{id} + \text{id}$

(2) $E \Rightarrow \underline{E} + E \Rightarrow E * E + E \Rightarrow^* \text{id} * \text{id} + \text{id}$



À **un mot** correspondent potentiellement **plusieurs arbres**.

Récapitulatif

- ▶ À une dérivation correspond un seul arbre.
- ▶ À un arbre correspondent potentiellement plusieurs dérivations.
- ▶ À un arbre correspond une unique dérivation gauche et une unique dérivation droite.
- ▶ À un mot correspondent potentiellement plusieurs arbres/interprétations.

Limites des langages réguliers

Langages et grammaires algébriques

Arbres syntaxiques

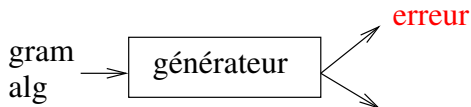
Grammaires « pathologiques »

Ambiguïté

Grammaires réduites

Les algébriques... et les autres

Grammaire refusée par le générateur



2 cas possibles :

- ▶ la grammaire est **pathologique** : le concepteur doit la réviser
- ▶ la grammaire ne convient pas à la méthode d'analyse syntaxique implantée par l'outil : autre problème, cf suite du cours

Pathologies vues dans ce cours

Les **grammaires ambiguës** :

- ▶ elles sont rejetées par le générateur
- ▶ avec un message d'erreur pas forcément explicite
- ▶ il faut trouver une grammaire équivalente

Les grammaires **non réduites** :

- ▶ acceptées par le générateur, éventuellement **avec warning**
- ▶ mais le générateur calcule en fait une **autre grammaire**
- ▶ signale souvent une erreur de conception
- ▶ il faut savoir comprendre et réparer

Limites des langages réguliers

Langages et grammaires algébriques

Les grammaires algébriques

Langage engendré, dérivations

Langages algébriques, comparaison avec les réguliers

Outillage

Arbres syntaxiques

Grammaires « pathologiques »

Ambiguïté

Grammaires réduites

Les algébriques... et les autres

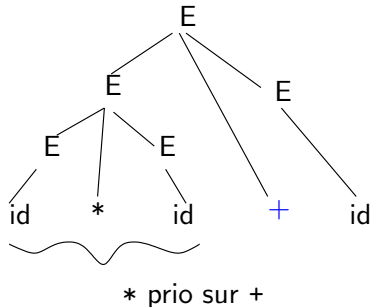
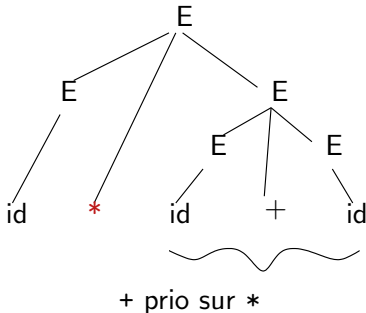
Comparaison avec les réguliers

Limitation des algébriques

Les autres

Interprétation d'un mot et ambiguïté

Deux interprétations (sémantiques) différentes de $\text{id} * \text{id} + \text{id}$:
ce mot est **ambigu**.



Définition

Definition

(**ambiguïté**) Un mot $w \in L(G)$ est **ambigu** s'il admet plusieurs arbres syntaxiques.

Récapitulatif

- ▶ À une dérivation correspond un seul arbre.
- ▶ À un arbre correspondent potentiellement plusieurs dérivations.
- ▶ À un arbre correspond une unique dérivation gauche et une unique dérivation droite.
- ▶ À un **mot** correspondent potentiellement **plusieurs arbres/interprétations**.
- ▶ À un **mot non ambigu** correspond un(e) **unique arbre/interprétation**.

Définitions

Definition

Une grammaire est **ambiguë** si elle permet de dériver au moins un mot ambigu.

Definition

Un langage est **ambigu** si toutes les grammaires qui l'engendrent sont ambiguës.

Le problème des interprétations multiples

Un programme ayant plusieurs interprétations ?

Désastreux pour un programme souhaité déterministe !

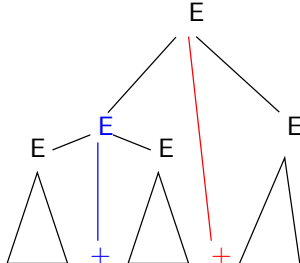
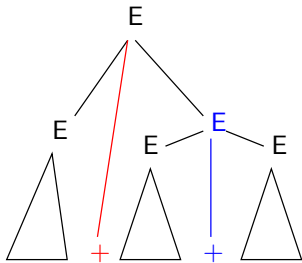
On ne travaille qu'avec des grammaires dites **non ambiguës**, pour lesquelles tout mot admet un unique arbre syntaxique.

⇒ il faut savoir repérer les grammaires ambiguës.

Détecter l'ambiguïté - 1

Certaines grammaires sont « clairement » ambiguës, repérable avec de l'expérience.

Ex de production récursive symétrique : $E \rightarrow E + E$.



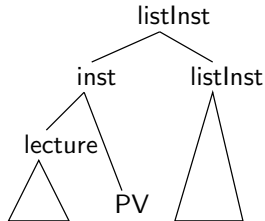
Détecter l'ambiguïté - 2

Certaines grammaires sont mal conçues

Ex grammaire Init G_1 non ambiguë :

$\text{listInst} \rightarrow \epsilon \mid \text{inst listInst}$
 $\text{inst} \rightarrow \text{affect PV} \mid \text{lecture PV}$
 $\text{lecture} \rightarrow \text{READ IDENT}$

READ IDENT PV...



Détecter l'ambiguïté - 3

Déplacement de la production vide.

Ex grammaire Init G_2 ambiguë :

$$\begin{aligned} \text{listInst} &\rightarrow \text{inst listInst} \mid \text{inst} \\ \text{inst} &\rightarrow \epsilon \mid \text{affect PV} \mid \text{lecture PV} \end{aligned}$$

Voyez-vous pourquoi G_2 est ambiguë ?

Prouver une ambiguïté

Prouver qu'une grammaire **est ambiguë** = trouver un mot qui admet au moins 2 arbres syntaxiques.

L'utilisation d'un générateur d'analyseur syntaxique peut aider.

Prouver qu'une grammaire **n'est pas ambiguë** = faire une preuve.

Décider de l'ambiguïté d'une grammaire est **difficile** : c'est un **problème indécidable**.

Grammaires ambiguës, en pratique

Quand un langage L est intuitivement non ambigu...

et qu'une grammaire engendrant L l'est... on peut essayer de la rendre non ambiguë :

- ▶ le plus souvent : on réfléchit et on change peu de choses ;
- ▶ cas particulier : les **grammaires à opérateurs**.

Grammaires à opérateurs

Grammaires faisant intervenir des opérateurs avec :

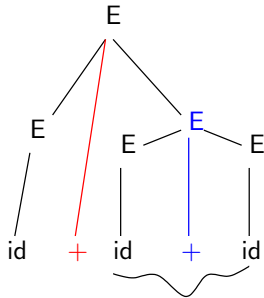
- ▶ associativité ;
- ▶ priorités.

Ex : grammaire (ambiguë) des expressions arithmétiques

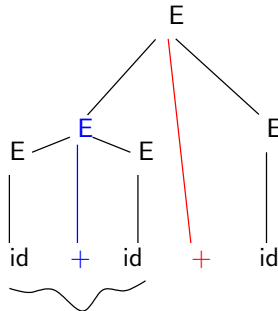
$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

Associativité des opérateurs

Première source d'ambiguïté : l'**associativité** de + et *.



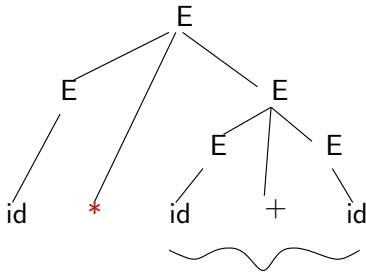
associativité droite



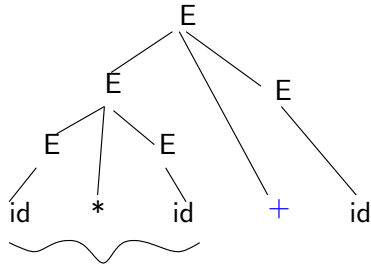
associativité gauche

Priorité des opérateurs

Seconde source d'ambiguïté : **priorité** de + et *.



+ prio sur *



* prio sur +

NB : plus prioritaire = dérivé du E le plus bas dans l'arbre

Principes - 1

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

Dans cette grammaire l'axiome E représente potentiellement :

- ▶ une somme : opérateur + ;
- ▶ un produit : opérateur * ;
- ▶ une expression atomique : id ;
- ▶ une expression parenthésée.

Pour refléter les priorités des opérateurs dans la grammaire :

- ▶ on repère la structure d'une expression ;
- ▶ on structure la grammaire en conséquence.

Principes - 2

Structure d'une expression : **somme** de **produits** de **facteurs**

$$\sum_{i=0 \dots n} \prod_{j=0 \dots m} x_{ij}$$

Rmq : d'abord la somme, la moins prioritaire.

Les facteurs x_{ij} sont soit :

- ▶ un atome `id`;
- ▶ une expression parenthésée.

Pour imiter cette structure dans la grammaire, on lui ajoute :

- ▶ $S \in V_N$ pour une somme ;
- ▶ $P \in V_N$ pour un produit ;
- ▶ $F \in V_N$ pour un facteur.

Traitement de l'associativité, ex des produits - 1

$$F \rightarrow \text{id} \mid (E) \quad P \rightarrow ? * ?$$

Un opérande d'un produit peut être :

- ▶ un autre produit : récursivité sur P ;
- ▶ un facteur F (ne permettant pas de dériver un produit).

Par ailleurs :

- ▶ on ne veut pas d'un produit $P * P$!
- ▶ il faut un « cas d'arrêt » : $P \rightarrow F$

$$P \rightarrow F \mid P * F ?$$

$$P \rightarrow F \mid F * P ?$$

$$F \rightarrow \text{id} \mid (E)$$

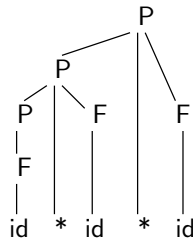
$$F \rightarrow \text{id} \mid (E)$$

Récursivité **gauche** ?

Récursivité **droite** ?

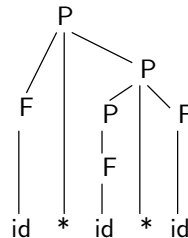
Traitement de l'associativité, ex des produits - 2

$P \rightarrow F \mid P * F$
 $F \rightarrow \dots$ récursivité gauche



associativité gauche

$P \rightarrow F \mid F * P$
 $F \rightarrow \dots$ récursivité droite

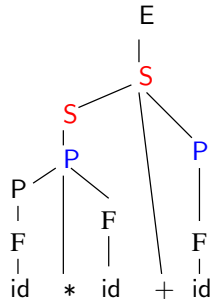


associativité droite

Ici on choisira donc une récursivité gauche.

Traitement des priorités - 1

En traitant la somme (de produits) de la même manière : OK

$$\begin{aligned} E &\rightarrow S \\ S &\rightarrow P \mid S + P \\ P &\rightarrow F \mid P * F \\ F &\rightarrow \text{id} \mid (E) \end{aligned}$$


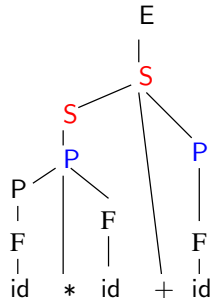
Traitement des priorités - 2

Rmq : en inversant somme et produit : KO

$E \rightarrow S$ S en haut

$S \rightarrow P \mid S + P$

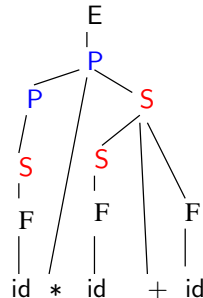
$P \rightarrow F \mid P * F \dots$



$E \rightarrow P$ P en haut

$P \rightarrow S \mid P * S$

$S \rightarrow F \mid S + F \dots$



Systématisation

Pour rendre une grammaire à opérateur non ambiguë :

- ▶ on ajoute un non terminal par niveau de priorité (S pour $+$, P pour $*$, etc) ;
- ▶ les moins prioritaires en haut de l'arbre, proches de l'axiome ;
- ▶ les plus prioritaires en bas de l'arbre, proches des feuilles ;
- ▶ les « atomes » tout en bas ;
- ▶ associativité gauche/droite \Rightarrow récursivité gauche/droite.

Exemple plus gros

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid - E \mid (E) \mid \text{id}$$

$$\underbrace{\{+, -\}}_S \leq_{\text{prio}} \underbrace{\{*, /\}}_P \leq_{\text{prio}} \underbrace{\{-\}}_U \leq_{\text{prio}} \underbrace{\{()\}}_A$$

$$E \rightarrow S$$

$$S \rightarrow P \mid S + P \mid S - P$$

$$P \rightarrow U \mid P * U \mid P / U$$

$$U \rightarrow - U \mid A$$

$$A \rightarrow (E) \mid \text{id}$$

Grammaires à opérateurs, bilan

Version ambiguë : très intuitive... mais ambiguë !

Version non ambiguë : opérationnalisable...

- ▶ mais complètement illisible !
- ▶ taille arbre beaucoup plus grande.

Grammaires à opérateurs, automatisations

On a un **algorithme** qui calcule une grammaire non ambiguë équivalente.

Il prend en entrée :

- ▶ une grammaire à opérateurs ambiguë ;
- ▶ et les niveaux de priorité + associativité des opérateurs.

Grammaires à opérateurs, outillage

Cup :

- ▶ on spécifie (et décore) uniquement la grammaire ambiguë ;
- ▶ on spécifie les niveaux de priorité + associativité des opérateurs.
- ▶ la version non-ambiguë est automatiquement calculée, mais reste implicite.

antLR :

- ▶ il faut écrire la grammaire non-ambiguë à la main
- ▶ donc maîtriser l'algorithme de calcul...

Limites des langages réguliers

Langages et grammaires algébriques

Les grammaires algébriques

Langage engendré, dérivations

Langages algébriques, comparaison avec les réguliers

Outillage

Arbres syntaxiques

Grammaires « pathologiques »

Ambiguïté

Grammaires réduites

Les algébriques... et les autres

Comparaison avec les réguliers

Limitation des algébriques

Les autres

Aparté : parallèle code/grammaire - 1

Quand on **code**... :

- ▶ on écrit des **bêtises** (bugs) ;
- ▶ on écrit du code mort ;
- ▶ on écrit des choses simplifiables, genre `if (x == true)` ;
- ▶ etc.

Pour **détecter ces bêtises** :

- ▶ **analyses statiques** possibles (à la **compilation**) :
 - ▶ détecte des erreurs de typages, de syntaxe, code mort, etc.
- ▶ **test** (à l'**exécution**) :
 - ▶ on essaie pour voir si ça marche ;
 - ▶ détecte les erreurs « sémantiques » (le programme est syntaxiquement correct mais ne répond pas à sa spécification).

Aparté : parallèle code/grammaire - 2

Quand on écrit **une grammaire** : **c'est pareil** ! on peut se tromper :

- ▶ on a oublié une production ;
- ▶ on s'est trompé dans l'écriture d'une production ;
- ▶ etc.

Certaines erreurs sont détectables **statiquement** :

- ▶ détection de productions et non- t^{aux} inutiles (\sim code mort) ;
- ▶ \Rightarrow obtention d'une grammaire dite **réduite** (cf cours).

Erreurs sémantiques (la grammaire ne dit pas ce qu'on pense) :

- ▶ on **teste** en **exécutant** l'analyseur syntaxique.

Aparté : parallèle code/grammaire - 3

On souhaite parfois réusiner (**refactoring**) un programme...

... de même on peut souhaiter **transformer une grammaire** pour :

- ▶ obtenir des **arbres syntaxiques** plus **compacts** ;
 - ▶ suppression des règles $X \rightarrow Y$ et $X \rightarrow \epsilon$;
 - ▶ \Rightarrow **grammaires propres** ;
- ▶ obtenir une forme de grammaire particulière propice à certaines analyses ;
 - ▶ **forme normale de Chomsky** (FNC) ;
 - ▶ $X \rightarrow a$ ou $X \rightarrow YZ$;
- ▶ mettre à part ϵ .

Pas dans ce cours, cf la littérature.

Grammaires réduites

Certaines grammaires sont « pathologiques » car certains **non-terminaux ne servent à rien**.

Souvent dû à une **erreur de conception** : cf code mort.

En pratique :

- ▶ les générateurs signalent au mieux le problème par un warning à la génération
- ▶ on décèle souvent le problème en testant l'analyseur syntaxique.

Definition

Une grammaire est dite **réduite** si elle ne contient pas de non-terminal **improductif** ou **inaccessible**.

Non-terminaux improductifs : exemple

$$liste \rightarrow elt ; liste$$
$$elt \rightarrow a \mid b$$

liste est **improductif** : il ne permet pas de dériver un mot de $\{a, b, ;\}^*$.

En fait on voulait dire : $liste \rightarrow elt ; liste \mid elt$
 $elt \rightarrow a \mid b$

Ce qu'en disent les outils

liste \rightarrow *elt* ; *liste*

elt \rightarrow *a* | *b*

Rien ! Pas de warning à la génération.

Mais l'analyseur syntaxique généré signale une erreur syntaxique dès qu'on termine l'entrée par EOF.

\Rightarrow on se doute qu'il y a une erreur qqpart !

Définition d'un improductif

Définition

Un non-terminal $X \in V_N$ est **improductif** s'il n'existe pas de mot $u \in V_T^*$ tel que $X \Rightarrow^* u$ (le langage engendré par X est vide). Il est **productif** sinon.

Calcul des improductifs

1. on calcule les **productifs** ;
2. par complémentaire on a les improductifs.

Ensuite on répare la grammaire pour que les improductifs ne le soient plus.

NB : on obtiendrait une grammaire équivalente à celle de départ en supprimant toutes les productions contenant un improductif en partie gauche ou droite.

Calcul des productifs : idée

X est productif :

- ▶ s'il existe une production $X \rightarrow u$ avec $u \in V_T^*$;
- ▶ ou s'il existe une production $X \rightarrow \alpha$ avec $\alpha \in (V_N \cup V_T)^*$ tel que tous les non-terminaux apparaissant dans α sont productifs.

Calcul des productifs : plus petit point fixe

Algorithme naïf pour calculer un **plus petit point fixe** par **itérations** successives :

- ▶ on part d'un ensemble initial ;
- ▶ on le fait grossir itérativement ;
- ▶ jusqu'à stabilisation.

Exemple

G_1 telle que $V_N = \{S, X, Y, Z, W\}$ (axiome S), $V_T = \{a, b, d\}$ et

$$(1) S \rightarrow a X$$

$$(2) S \rightarrow d W$$

$$(3) X \rightarrow b S$$

$$(4) X \rightarrow a Y b Y$$

$$(5) Y \rightarrow b a$$

$$(6) Y \rightarrow a Z$$

$$(7) Z \rightarrow a Z X$$

$$(8) W \rightarrow a S$$

Exemple, résolution

n° itération	<i>Prod</i>
0 (init)	
1	
2	
3	
...	

Algorithme de calcul des productifs

Entrée : une grammaire algébrique G

Sortie : l'ensemble $Prod$

de ses non-terminaux productifs

// Init

$Prod = \emptyset$

pour toute production $X \rightarrow u, u \in V_T^*$

faire $Prod = Prod \cup \{X\}$ // noté $Prod \cup = \{X\}$

fait

Algorithme de calcul des productifs

```
// Itérations
faire jqa stabilisation de Prod
    New =  $\emptyset$  // les productifs découverts
                // pendant l'itération
    pour toute production  $X \rightarrow u_1 X_1 u_2 \dots X_n u_n$ 
        tq  $X \notin Prod$  // X pas encore traité
            et  $\{X_1, \dots, X_n\} \subseteq Prod$  // X productif
        faire New  $\cup = \{X\}$ 
    fait
    Prod  $\cup = New$ 
fait
```

Exemple, solution

On trouve $Prod = \{Y, X, S, W\}$, donc Z est improductif.

En supprimant Z partout on obtient la grammaire équivalente à G_1 :

G'_1 telle que $V_N = \{S, X, Y, W\}$ et

$$(1) S \rightarrow a X$$

$$(2) S \rightarrow d W$$

$$(3) X \rightarrow b S$$

$$(4) X \rightarrow a Y b Y$$

$$(5) Y \rightarrow b a$$

$$(8) W \rightarrow a S$$

Non-terminaux inaccessibles : exemple

Ex : $instr \rightarrow if \mid while \mid do$
 $affect \rightarrow IDAFFID$

L'axiome ne permet pas « d'attraper » *affect*, qui est inutile.

En fait on voulait dire : $instr \rightarrow if \mid while \mid do \mid affect$
 $affect \rightarrow \dots$

Ce qu'en disent les outils

instr → *if* | *while* | *do*

affect → IDAFFID

Cup affiche un warning :

**** Production "affect : := ID AFF ID" never reduced*

antLR ne signale rien.

Dans les 2 cas l'analyseur syntaxique généré n'accepte aucune affectation.

⇒ on se doute qu'il y a une erreur qqpart !

Définition d'un inaccessible

Definition

Soit G une grammaire algébrique d'axiome S . Un non-terminal $X \in V_N$ est **inaccessible** s'il n'existe pas de mots $\alpha, \beta \in (V_N \cup V_T)^*$ tels que $S \Rightarrow^* \alpha X \beta$. Il est **accessible** sinon.

Calcul des inaccessibles

1. on calcule les **accessibles** ;
2. on a par complémentaire les inaccessibles ;

Ensuite on répare la grammaire pour que les inaccessibles ne le soient plus.

NB : on obtiendrait une grammaire équivalente à celle de départ en supprimant toutes les productions contenant un inaccessible en partie gauche (suffisant, supprime automatiquement les inaccessibles en partie droite).

Calcul des accessibles : idée

X est accessible si :

- ▶ c'est l'axiome ;
- ▶ ou il existe une production $Y \rightarrow \alpha X \beta$ telle que Y est accessible.

Même principe d'itération de point fixe que pour les accessibles
mais on cherche les candidats en partie **droite** de production.

Exemple

G_2 telle que $V_N = \{S, Y, U, V, X, Z\}$ (axiome S),
 $V_T = \{a, b, d, e\}$ et

- (1) $S \rightarrow Y$
- (2) $Y \rightarrow Y Z$
- (3) $Y \rightarrow Y a$
- (4) $Y \rightarrow b$
- (5) $U \rightarrow V$
- (6) $X \rightarrow e$
- (7) $V \rightarrow V d$
- (8) $V \rightarrow d$
- (9) $Z \rightarrow Z X$

Exemple, solution

$$Acc = \{S, Y, Z, X\}$$

En supprimant U et V partout on obtient la grammaire équivalente à G_2 :

G'_2 telle que $V_N = \{S, X, Y, Z\}$ et

- (1) $S \rightarrow Y$
- (2) $Y \rightarrow Y Z$
- (3) $Y \rightarrow Y a$
- (4) $Y \rightarrow b$
- (6) $X \rightarrow e$
- (9) $Z \rightarrow Z X$

Algorithme de calcul des accessibles

Entrée : une grammaire algébrique G d'axiome S
Sortie : l'ensemble Acc
de ses non-terminaux accessibles
// Init
 $Acc = \{S\}$

Algorithme de calcul des accessibles

```
// Itérations
faire jqa stabilisation de  $Acc$ 
     $New = \emptyset$  // les accessibles découverts
    // pendant l'itération
    pour toute production  $Y \rightarrow \alpha X \beta$ ,  $\{X, Y\} \subseteq V_N$ 
        tq  $X \notin Acc$  // X pas encore traité
            et  $Y \in Acc$  // X accessible
        faire  $New \cup = \{X\}$ 
    fait
     $Acc \cup = New$ 
fait
```

Grammaire réduite : attention

Si on souhaite calculer une grammaire réduite :

Il faut supprimer **d'abord** les improductifs **puis** les inaccessibles.

Ex : Improductifs de $G'_2 : \{Z\}$

Si on supprime $Z \dots X$ devient inaccessible !

La grammaire réduite équivalente est donc :

(1) $S \rightarrow Y$

(3) $Y \rightarrow Y a$

(4) $Y \rightarrow b$

Limites des langages réguliers

Langages et grammaires algébriques

Arbres syntaxiques

Grammaires « pathologiques »

Les algébriques... et les autres

Comparaison avec les réguliers

Limitation des algébriques

Les autres

Propriétés de clôture

réguliers \subset algébriques

Les algébriques sont **plus expressifs** que les réguliers. Ils ont donc des **propriétés moins fortes** :

clôture par	langages réguliers	langages algébriques
Union	Oui	Oui
Concaténation	Oui	Oui
Etoile	Oui	Oui
Intersection	Oui	Non
Complémentaire	Oui	Non

Régulier ? Algébrique ?

Description de déplacements sur h et b .

$$L_1 = \{w \in \{h, b\}^*\}$$

$$L_2 = \{h^n b^p \mid n \geq 0, p \geq 0\}$$

$$L_3 = \{h^n b^p \mid n \geq p \geq 0\}$$

$$L_4 = \{w \in \{h, b\}^* \mid |w|_h = |w|_b\}$$

$$L_5 = \{w \in \{h, b\}^* \mid w \text{ est un palindrome} \}$$

$$L_6 = \{ww \mid w \in \{h, b\}^*\}$$

Limitation des algébriques

$L_6 = \{ww \mid w \in \{h, b\}^*\}$ n'est **pas** algébrique.

Les grammaires algébriques expriment :

- ▶ les propriétés **structurelles** des langages ;
- ▶ mais pas leurs propriétés **contextuelles** ;
- ▶ on les appelle aussi « **grammaires hors contexte** ».

Par ex, on ne peut pas exprimer par une grammaire algébrique :

- ▶ que toute variable utilisée a été déclarée ;
- ▶ les vérifications de typage, etc.

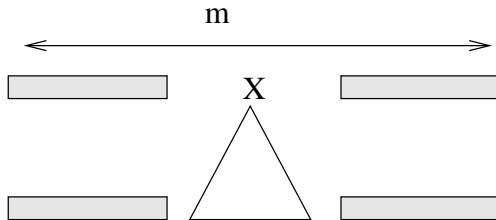
Ces propriétés relèvent de **l'analyse sémantique**.

Limitation des algébriques

Étant donné :

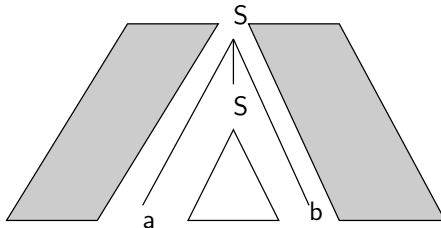
- ▶ un mot m de $(V_T \cup V_N)^*$ contenant $X \in V_N$
- ▶ une production $X \rightarrow \alpha$

la dérivation remplace X par α **indépendamment de m** , le **contexte** de X .



Ex de propriété structurelle

Toute accolade ouverte est fermée : $S \rightarrow \epsilon \mid aSb$



a et b sont dans le même arbre issu de S

\Rightarrow propriété **indépendante** du contexte.

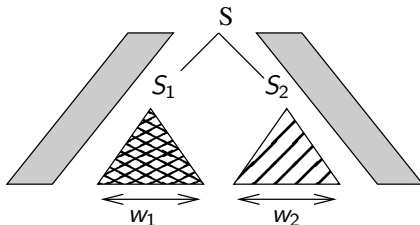
Ex de propriété contextuelle

$\{ww \mid w \in \{h, b\}^*\}$: essai...

$$S \rightarrow S_1 S_2$$

$$S_1 \Rightarrow^* w_1$$

$$S_2 \Rightarrow^* w_2$$



w_1 et w_2 sont dans deux arbres indépendants :

- ▶ on ne peut pas forcer $w_1 = w_2$;
- ▶ propriété **dépendante** du contexte.

Autre exemple

$$\{a^n b^n c^n \mid n \geq 0\}$$

Comment forcer autant de a que de b que de c ?

Impossible avec des dérivations indépendantes les unes des autres.

$\Rightarrow \{a^n b^n c^n \mid n \geq 0\}$ n'est pas algébrique.

Justification théorique : lemme fondamental des algébriques

Soient $u_1, u_2, v \in (V_T \cup V_N)^*$.

Si $u_1 u_2 \rightarrow^k v$ alors il existe $v_1, v_2 \in (V_T \cup V_N)^*$ tels que :

- ▶ $v = v_1 v_2$
- ▶ $u_1 \rightarrow^{k_1} v_1$ et $u_2 \rightarrow^{k_2} v_2$
- ▶ $k_1 + k_2 = k$.

Ce lemme ou sa généralisation sert dans les preuves, par exemple pour montrer que le langage engendré par $S \rightarrow aSb \mid \epsilon$ est $\{a^n b^n \mid n \geq 0\}$.

Comment être sensible au contexte ?

En utilisant une **grammaire contextuelle**.

Productions de la forme : $\alpha \rightarrow \beta$ avec $|\alpha| \leq |\beta|$
avec $\alpha, \beta \in (V_N \cup V_T)^*$

Ex : $AB \rightarrow BA$

Les grammaires contextuelles :

- ▶ engendrent les **langages contextuels** ;
- ▶ ne sont pas utilisées lors de l'analyse syntaxique ;
- ▶ pas d'algorithme polynomial connu qui, pour tout mot, détermine si ce mot est engendré par une grammaire contextuelle donnée.

Si on continue à généraliser...

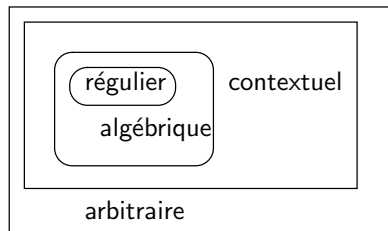
Il existe des **grammaires arbitraires**.

Productions de la forme : $\alpha \rightarrow \beta$
avec $\alpha, \beta \in (V_N \cup V_T)^*$

Ex : $AB \rightarrow \epsilon$

Ces grammaires engendrent l'ensemble de tous les langages.

Hiérarchie des langages et des grammaires



Chaque type de langage est généré par un type de grammaire particulier... y compris les réguliers.

Classification de Chomsky pour les grammaires :

$\text{régulière} \subset \text{algébrique} \subset \text{contextuelle} \subset \text{quelconque}$