Grammaires

MIASHS L2

(d'après le cours de Julie Dugdale)

Jérôme GENSEL – Jerome.Gensel@univ-grenoble-alpes.fr
Prof. Université Grenoble Alpes
UFR SHS – Laboratoire d'Informatique de Grenoble

Sommaire

- Qu'est-ce qu'une grammaire?
  - grammaire générative
  - grammaire analytique
- Définition d’une grammaire
- Exemples
- La hiérarchie de Chomsky
  - Type 0 (grammaires non-restreintes)
  - Type 1 (grammaires sensibles au contexte)
  - Type 2 (grammaires hors contexte)
  - Type 3 (grammaires régulières)
- Dérivations gauches et droites
- Arbre de dérivation
- Ambiguïté syntaxique

Qu'est ce qu'une grammaire?

- Nous avons souvent parlé de chaînes et de langages, mais comment générer ces *mots* dans un *langage* et comment vérifier si un mot est dans un langage ?

- Deux catégories principales de grammaires formelles :
  - *Grammaires génératives*, au sens de « qui permettent d'énumérer », ce sont des ensembles de *règles* pour la génération des chaînes d'un langage
  - *Grammaires analytiques*, ce sont des ensembles de *règles* pour l'analyse de chaînes et déterminer si ces chaînes appartiennent au langage

Qu'est ce qu'une grammaire?

- Brièvement, une grammaire générative décrit comment *écrire* l'ensemble des chaînes d'un langage
- Une grammaire analytique décrit comment *reconnaître* les chaînes qui sont membres de l'ensemble des chaînes d'un langage
- Une grammaire formelle (**ou grammaire**) est une description précise d'un langage formel (c'est-à-dire d'un ensemble de mots)

Grammaires génératives

- Nous nous intéressons ici aux grammaires génératives
- C'est-à-dire aux *ensembles de règles* pour la génération des chaînes d'un langage

Grammaires génératives

- Une grammaire générative consiste en un ensemble de règles pour générer les chaînes

- Pour générer une chaîne dans le langage :
  1. Commencer par une chaîne constituée seulement d'un *symbole de départ*
  2. Appliquer successivement les règles (n'importe quel nombre de fois, dans n'importe quel ordre) pour réécrire cette chaîne

Grammaires génératives

- Le langage est constitué de toutes les chaînes qui peuvent être générées de cette manière
- Toute séquence particulière de choix possibles effectués pendant ce processus de réécriture donne naissance à une chaîne particulière dans le langage
- S'il y a **plusieurs manières différentes** de générer une *seule chaîne*, alors la grammaire est alors dite « **ambiguë** »

Exemple

Supposons l'alphabet contenant les symboles *a* et *b*,

Le symbole de départ est ***S***

Les règles sont :

1. S → aSb
2. S → ba

Procédure

- Commencer avec *S*, et choisir une règle à lui appliquer.
  - Si nous choisissons la règle 1
    - Remplacer S par *aSb*
      ➔ on obtient la chaîne *aSb*
  - Si nous choisissons la règle 1 à nouveau
    - Remplacer *S* par *aSb*
      ➔ on obtient la chaîne *aaSbb*
- Répéter le processus jusqu'à ce qu'il ne reste plus que des symboles de l'alphabet (ici *a* et *b*)
  - Si nous choisissons la règle 2
    - Remplacer *S* par *ba*
      ➔ on obtient la chaîne *aababb*
- Fin

Exemple

- On peut écrire cette série de choix plus brièvement, en utilisant des symboles :

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb$$

- Le langage de la grammaire est l'ensemble de toutes les chaînes qui peuvent être générées en utilisant ce processus :

$$\{ba, abab, aababb, aaababbb, \ldots\}$$

Définition d'une grammaire

- $G = \langle V_N, V_T, S, \mathbf{R} \rangle$

  - V_N : vocabulaire **Non terminal**
  - V_T : vocabulaire **Terminal**
  - $S \in V_N$: **axiome**
  - **R** : **règles de production** (ou de réécriture)

  Avec $V_N \cap V_T = \varnothing$

Sur l'exemple précédent...

Supposons l'alphabet contenant les symboles *a* et *b*,

Le symbole de départ est *S*

Les règles sont :

1. $S \rightarrow aSb$
2. $S \rightarrow ba$

- Que sont V_N, V_T, S, **R**?
  - $G = \langle V_N, V_T, S, \mathbf{R} \rangle$
  - V_N : vocabulaire non terminal
  - V_T : vocabulaire terminal
  - $S \in V_N$: axiome
  - **R** : règles de production (ou de réécriture)

Avec l'exemple précédent

Supposons l'alphabet contenant les symboles *a* et *b*,
Le symbole de départ est *S*
Les règles sont :

1. $S \rightarrow aSb$
2. $S \rightarrow ba$

- Que sont V_N, V_T, S, **R**?
  - $G = \langle V_N, V_T, S, \mathbf{R} \rangle$
  - V_N : vocabulaire non terminal = {S}
  - V_T : vocabulaire terminal = {a, b}
  - $S \in V_N$: axiome
  - **R** : règles de production (ou de réécriture)
    R = {R1 : $S \rightarrow aSb$, R2 : $S \rightarrow ba$}

Mot engendré

- Intuitivement, un mot $w \in V_T^*$ est engendré par une grammaire si on peut l'obtenir au bout d'un certain nombre (donc un nombre fini!) de réécritures à partir du symbole axiome

Règle

- Une règle est un couple (φ, ψ) qu'on note en général :

  $$\varphi \rightarrow \psi$$

  où : $\varphi \in (V_N \cup V_T)^* - \{\varepsilon\}$

  et $\psi \in (V_N \cup V_T)^*$

- On dit que «φ se réécrit ψ »
- C'est-à-dire, chaque règle de production relie une chaîne de symboles à une autre, où la chaîne de gauche (φ) contient au moins un symbole non-terminal
- Si la chaîne de droite (ψ) est la chaîne vide (elle ne contient aucun symbole), elle est notée ε

Exemple 1

- Soit la grammaire…
  - $V_N = \{S, A, B\}$
  - $V_T = \{0, 1\}$
  - $S \in V_N$: axiome
  - **R** :
    1. $S \rightarrow 0A1B$
    2. $1B \rightarrow 1ABB$
    3. $1A \rightarrow A1$
    4. $1B \rightarrow 11$
    5. $0A \rightarrow 00$

Exemple de mot engendré

S => 0A1B	(en utilisant R1)
=> 0A1ABB	(R2)
=> 0AA1BB	(R3)
=> 0AA1ABBB	(R2)
=> 0AAA1BBB	(R3)
=> 0AAA11BB	(R4)
=> 0AAA111B	(R4)
=> 0AAA1111	(R4)
=> 00AA1111	(R5)
=> 000A1111	(R5)
⇒ 00001111	(R5)

R :

1. S → 0A1B
2. 1B → 1ABB
3. 1A → A1
4. 1B → 11
5. 0A → 00

Exemple 2

- Soit la grammaire…
  - $V_N = \{S, B\}$
  - $V_T = \{a,b,c\}$
  - $S \in V_N$: axiome
  - **R** :
    1. $S \rightarrow aBSc$
    2. $S \rightarrow abc$
    3. $Ba \rightarrow aB$
    4. $Bb \rightarrow bb$

Quelques exemples de la dérivation de chaînes dans L(G):

$$S \Rightarrow_2 abc$$

$$S \Rightarrow_1 aBSc \Rightarrow_2 aBabcc \Rightarrow_3 aaBbcc \Rightarrow_4 aaabbcc$$

$$S \Rightarrow_1 aBSc \Rightarrow_1 aBaBScc \Rightarrow_2 aBaBabccc \Rightarrow_3 aaBBabccc \Rightarrow_3 aaBaBbccc \Rightarrow_3 aaaBBbccc \Rightarrow_4 aaaBbbccc \Rightarrow_4 aaabbbccc$$

- Cette grammaire définit le langage qui est constitué de l'ensemble des chaînes qui commencent par 1 ou plusieurs 'a', suivis par le même nombre de 'b', suivis par le même nombre de 'c' :

$$L = \{a^n b^n c^n | n \geq 1\}$$

Dérivations

- Dérivation immédiate : $\varphi \Rightarrow \psi$
- Dérivation : $\varphi \Rightarrow^* \psi$

Dérivation immédiate

$\varphi \Rightarrow \psi$

si et seulement si :

- $\varphi = xuy$, $u \neq \varepsilon$
- $\psi = xvy$
- $(u \rightarrow v) \in \mathbf{R}_G$

Dérivation

- C'est la fermeture réflexive et transitive de '=>'

$$\varphi \Rightarrow^* \psi$$

si et seulement si :

- Il existe $k \geq 0$ et une suite $\varphi_0, \varphi_1, \ldots \varphi_k$, d'éléments de $(V_N \cup V_T)^*$ tels que
- $\varphi = \varphi_0$
- $\psi = \varphi_k$
- $\forall i, 0 \leq i \leq k-1 \Rightarrow \varphi_i \Rightarrow \varphi_{i+1}$

Exemple 1

- Rappel

S => 0A1B => 0A1ABB => 0AA1BB => 0AA1ABBB => 0AAA1BBB => 0AAA11BB => 0AAA111B => 0AAA1111 => 00AA1111 => 000A1111 => 00001111

Donc, par exemple :

0AAA1BBB => 0AAA11BB (dérivation immédiate)

0A1B => 0A1ABB (dérivation immédiate)

0A1ABB =>* 000A1111 (dérivation)

S =>* 00001111 (dérivation)

Mais aussi: 0A1ABB =>* 0A1ABB (dérivation)

Remarque

- $\varphi \Rightarrow^* \varphi$: autrement dit, la relation $\Rightarrow^*$ est réflexive

Démonstration : dans la définition, si k=0, alors :

$$\varphi = \varphi_0 = \varphi_k = \psi$$

La condition

$$\forall i,\ 0 \leq i \leq k-1 \Rightarrow \varphi_i \Rightarrow \varphi_{i+1}$$

est trivialement vérifiée car aucun i ne satisfait $0 \leq i \leq -1$

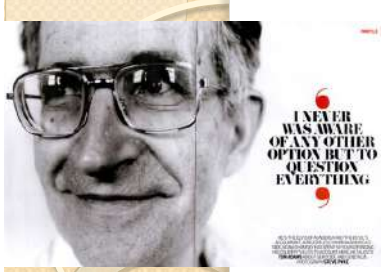
Langage engendré par G

$$L(G) = \{w \in V_T^*;\ S \Rightarrow^* w\}$$

Exemple 1 : $00001111 \in L(G)$

Hiérarchie de Chomsky

- Noam Chomsky a d'abord formalisé les grammaires génératives en 1956
- Il les a classifiées en types (connu maintenant comme la hiérarchie de Chomsky)
- Il y a 4 types (type 0 - type 3)
- La différence : ces 4 types de grammaire ont des règles de production de plus en plus strictes et peuvent exprimer de moins en moins de langages formels



Qui est Noam Chomsky?

- Né en décembre 1928
- Linguiste (Professeur émérite de linguistique de l'Institut MIT)
- Domaine d'expertise : la linguistique théorique
- Contribution majeure : créa une théorie des grammaires génératives
- Chomsky fut cité comme source plus souvent que tout autre chercheur vivant pendant la période 1980–1992, et est le 8ième chercheur le plus cité de tous les temps

Hiérarchie de Chomsky – type 0

- Grammaires de type-0 (ou *grammaires non-restreintes*) incluent toutes les grammaires formelles

- Elles génèrent exactement tous les langages qui peuvent être reconnus par une machine de Turing

- Avec le type-0 : pas de restriction sur les côtés gauche et droit des règles de production de la grammaire

Hiérarchie de Chomsky – type 1

- Grammaires de type-1 (ou *grammaires sensibles au contexte*) génèrent les langages sensibles au contexte
- Règles de la forme $\alpha A \beta \rightarrow \alpha \gamma \beta$
  avec *A* un non-terminal
  et α, β et γ des chaînes de terminaux et non-terminaux
- Les chaînes α et β peuvent être vides, mais γ doit être non-vide
- La règle $S \rightarrow \epsilon$ est permise si *S* n'apparaît sur le côté droit d'aucune règle
- Les langages sont reconnus par un automate borné linéaire (une machine de Turing non-déterministe dont la bande – quantité de mémoire nécessaire pour le calcul – est bornée par le produit de la longueur du flux d'entrée par une constante)

Exemple de grammaire de type 1

V_N = {S, B, C};
V_T = {**a, b, c**}

S → **a**SBC
S → **ab**C
CB → CBC (αAβ→ $\alpha\gamma\beta$ où α=C et A = B et γ=BC)
bB → **bb**
bC → **bc**
cC → **cc**

- *Note* : la grammaire est de type 1 car dans toutes ses règles la partie gauche est de longueur inférieure ou égale à la partie droite

Hiérarchie de Chomsky – type 2

Grammaires de type-2 (grammaires *hors* contexte)
génèrent des langages sans contexte

- Règles de la forme : $A \rightarrow \gamma$
  avec *A* **un** non-terminal et γ une chaîne de terminaux et de non-terminaux

- Ces langages sont reconnus par un automate à pile (*pushdown automaton* en anglais) non-déterministe

- *Note* : les langages hors contexte sont la base théorique pour la syntaxe de la plupart des *langages de programmation*

Exemple de grammaire de type 2

$V_N = \{S\}$;

$V_T = \{\mathbf{a}, \mathbf{b}\}$

$S \rightarrow \mathbf{aSb}$

$S \rightarrow \mathbf{ab}$

Note : une grammaire est dite *de type 2*, (*hors-contexte)* si et seulement si elle est de type 1 et si, pour toute règle, la partie gauche est réduite à un seul symbole non-terminal

Hiérarchie de Chomsky – type 3

Les grammaires de type-3 (grammaires régulières) génèrent des langages réguliers

- Les règles sont ici restreintes à un seul non-terminal sur le côté gauche et un côté droit constitué d'*un seul terminal* potentiellement suivi (ou précédé, mais pas les deux dans la même grammaire) d'*un seul non-terminal*
  - Linéaire gauche
  - Linéaire droite
- La règle $S \longrightarrow \epsilon$ est aussi ici permise si *S* n'apparaît pas à droite d'une règle
- Ces langages peuvent être reconnus par un automate à états finis

Exemple de grammaire linéaire (de type 3)

$S \rightarrow aS$

$S \rightarrow aA$ (linéaire gauche)

$A \rightarrow bA$

$A \rightarrow b$

Types de règles (in a nutshell)

- Type 0 : toutes les règles
- Type 1 : seulement des règles $\varphi \rightarrow \psi$ avec $|\varphi| \leq |\psi|$ ($\alpha A \beta \rightarrow \alpha \gamma \beta$)
- Type 2 : seulement des règles $\varphi \rightarrow \psi$ avec $\varphi = X \in V_N$
- Type 3 :
  - seulement des règles $\varphi \rightarrow \psi$ avec $\varphi = X \in V_N$ et $\psi = xY$ ou $\psi = x$ (avec $Y \in V_N$ et $x \in V_T$)
  - seulement des règles $\varphi \rightarrow \psi$ avec $\varphi = X \in V_N$ et $\psi = Yx$ ou $\psi = x$ (avec $Y \in V_N$ et $x \in V_T$)

Remarques

- Les grammaires de types 1, 2, 3 ne permettent pas d'engendrer le mot vide
  - Les langages qu'elles engendrent ne contiennent donc pas le mot vide
- Nous définissons donc les grammaires de types étendus 1, 2, 3 comme étant les grammaires de types respectifs 1, 2, 3 auxquelles on rajoute la possibilité de règles ayant ε en partie droite

Types de règles

type *i* étendu = type *i* + règles avec ε possible en partie droite

Retour sur l'exemple de grammaire linéaire (de type 3)

$S \rightarrow aS$

$S \rightarrow aA$ (linéaire gauche)

$A \rightarrow bA$

$A \rightarrow b$

Exemples de dérivation

- S => aA => ab
- S => aS => aaS => aaaS => aaaaA => aaaab
- S => aS => aaA => aab
- S => aS => aaA => aabA => aabbA => aabbb
- L(G) = $\{a^n b^m;\ n,m \geq 1\}$
  (langage désigné par aa*bb*)

Exemple de grammaire linéaire de type 3 étendu

$S \rightarrow aS$
$S \rightarrow bA$ (linéaire gauche)
$S \rightarrow \varepsilon$
$A \rightarrow bA$
$A \rightarrow \varepsilon$

S => aS => a
S => aS => aaS => aa
S => aS => aaS => aaaS => aaabA => aaab
S => aS => aaS => aaaS => aaabA => aaabbA => aaabb

L(G) = a*b*

Exemple de grammaire hors-contexte (de type 2)

- $S \rightarrow aSb$
- $S \rightarrow ab$

- S => ab
- S => aSb => aaSbb => aaabbb
- S => aSb => aaSbb => aaaSbbb => aaaabbbb

- $L(G) = \{a^n b^n, n \geq 1\}$

Grammaires hors-contexte de type 2 étendu

- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$

- $S \Rightarrow \varepsilon$
- $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$
- $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

- $L(G) = \{a^n b^n;\ n \geq 0\}$

Grammaire et définition récursive

Langage des Structures de Parenthèses Equilibrées (SPE)

1. Base : ε est une SPE
2. Schéma d'induction :
   - Si A est une SPE, alors (A) est une SPE
   - Si A et B sont des SPE, alors AB est une SPE
3. Clause finale : rien n'est une SPE hormis ce qui est obtenu par (1) et/ou (2)

Traduction sous forme de grammaire

$$S \rightarrow \varepsilon \mid (S) \mid SS$$

Ex: S => (S) => ((S)) => ((SS)) =>
(((S)S)) => ((()(S))) => ((()(SS))) =>
((()((S)S))) => ((()(()S))) => ((()(())))

Dérivations gauches et droites

- Etant donné une grammaire hors-contexte (type 2) il peut exister plusieurs manières de réécrire ses non-terminaux pour arriver précisément à la même chaîne
- Par exemple, considérons la grammaire (V_N, V_T,S,R) où V_N = {S,A,B} et V_T = {a, b, c}, R={S→aABb, A→bAb, A→c, B→aB, B→b}

$$
\begin{aligned}
S &\rightarrow aABb \\
A &\rightarrow bAb \\
A &\rightarrow c \\
B &\rightarrow aB \\
B &\rightarrow b
\end{aligned}
$$

Exemple (suite)

- La chaîne abcbabb peut être dérivée :

$$S \Rightarrow aABb \Rightarrow abAbBb \Rightarrow abcbBb \Rightarrow abcbaBb \Rightarrow abcbabb$$

- En étendant toujours le non-terminal le plus à gauche en premier. Une telle dérivation s'appelle une dérivation gauche

$$\begin{aligned} S &\rightarrow aABb \\ A &\rightarrow bAb \\ A &\rightarrow c \\ B &\rightarrow aB \\ B &\rightarrow b \end{aligned}$$

Exemple (suite)

- Mais la chaîne abcbabb pourrait aussi être dérivée :

$$S \Rightarrow aABb \Rightarrow aAaBb \Rightarrow aAabb \Rightarrow abAbabb \Rightarrow abcbabb$$

- En étendant toujours le non-terminal le plus à droite en premier. Une telle dérivation s'appelle une dérivation droite

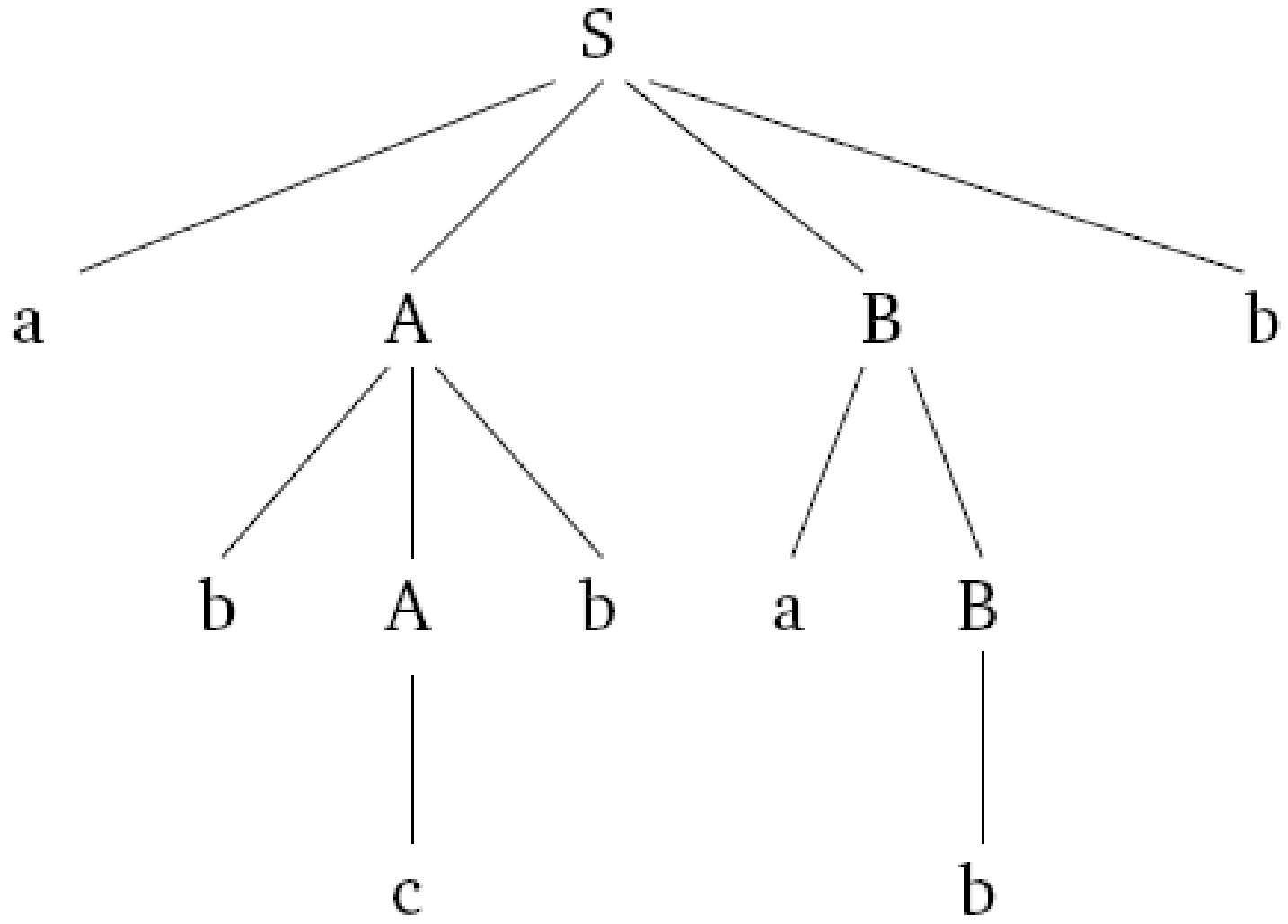
$$\begin{aligned} S &\rightarrow aABb \\ A &\rightarrow bAb \\ A &\rightarrow c \\ B &\rightarrow aB \\ B &\rightarrow b \end{aligned}$$

Arbre de dérivation

- Les dérivations dans les grammaires génératives peuvent être représentées par des **arbres de dérivation**

- La dérivation de la chaîne *abcbabb* selon la grammaire précédente peut être représentée par l'arbre…

$$S \Rightarrow aABb \Rightarrow abAbBb \Rightarrow abcbBb \Rightarrow abcbaBb \Rightarrow abcbabb$$



$$\begin{aligned} S &\rightarrow aABb \\ A &\rightarrow bAb \\ A &\rightarrow c \\ B &\rightarrow aB \\ B &\rightarrow b \end{aligned}$$

Arbre de dérivation

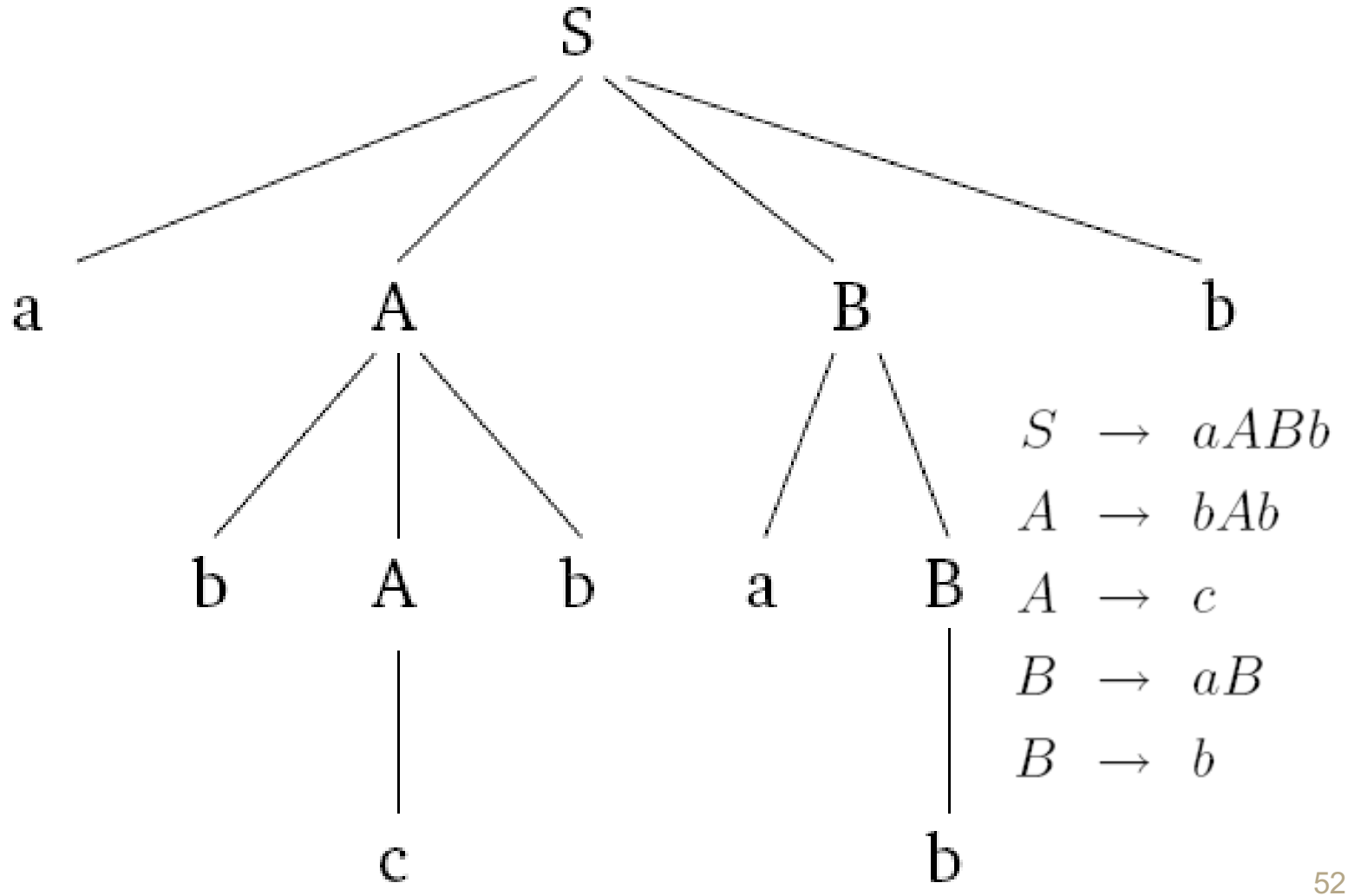
- *Note* :
  - la dérivation gauche correspond à la construction de l'arbre par la branche de gauche en premier
  - la dérivation droite correspond à la construction de l'arbre par la branche de droite en premier

- Cependant, les deux dérivations aboutissent exactement au même arbre…
- … Donc l'ordre d'application des règles de réécriture n'affecte pas l'ensemble des chaînes générées…
- Aussi, si une chaîne peut être générée par une dérivation droite, alors il existe une dérivation gauche de cette chaîne

Arbre de dérivation
dans une grammaire hors-contexte

- Soit G = $<V_N, V_T, S, \mathbf{R}>$ une grammaire hors-contexte
- Un arbre de dérivation pour $w \in V_T^*$ dans G est un arbre tel que :
  - Racine : S
  - Concaténation des feuilles : w
  - Si un nœud N a pour descendants immédiats $N_1, N_2, \ldots, N_k$, alors $(N \rightarrow N_1N_2\ldots N_k) \in R$

Si un nœud N a pour descendants immédiats N_1, N_2, …, N_k, alors $(N \rightarrow N_1N_2\ldots N_k) \in R$

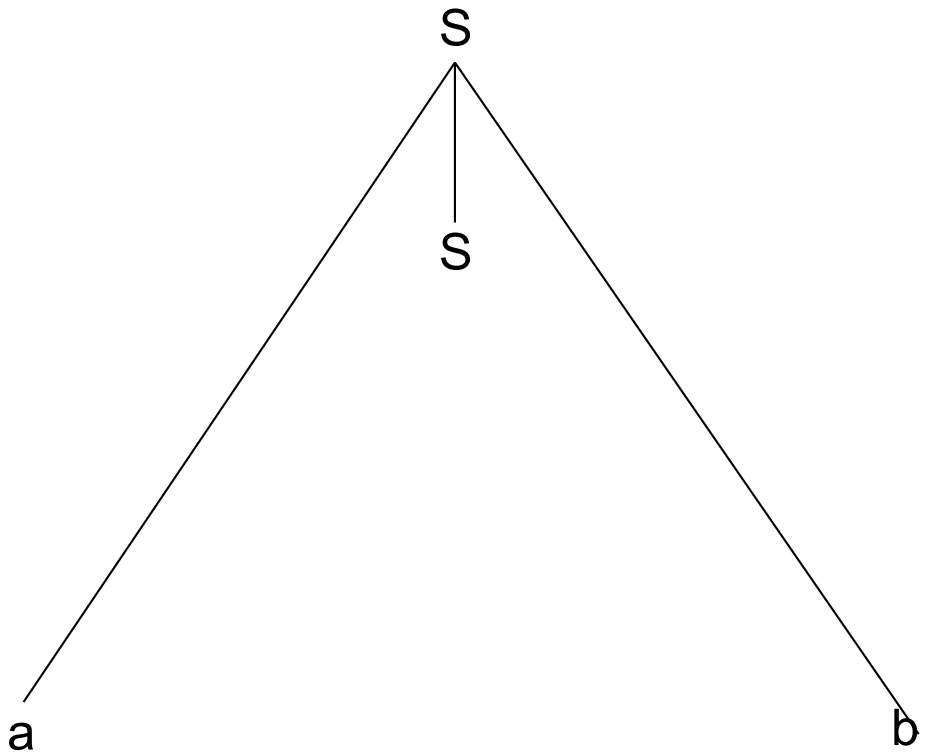


Théorème

- $w \in L(G) \Leftrightarrow$ il existe un arbre de dérivation A pour w dans G

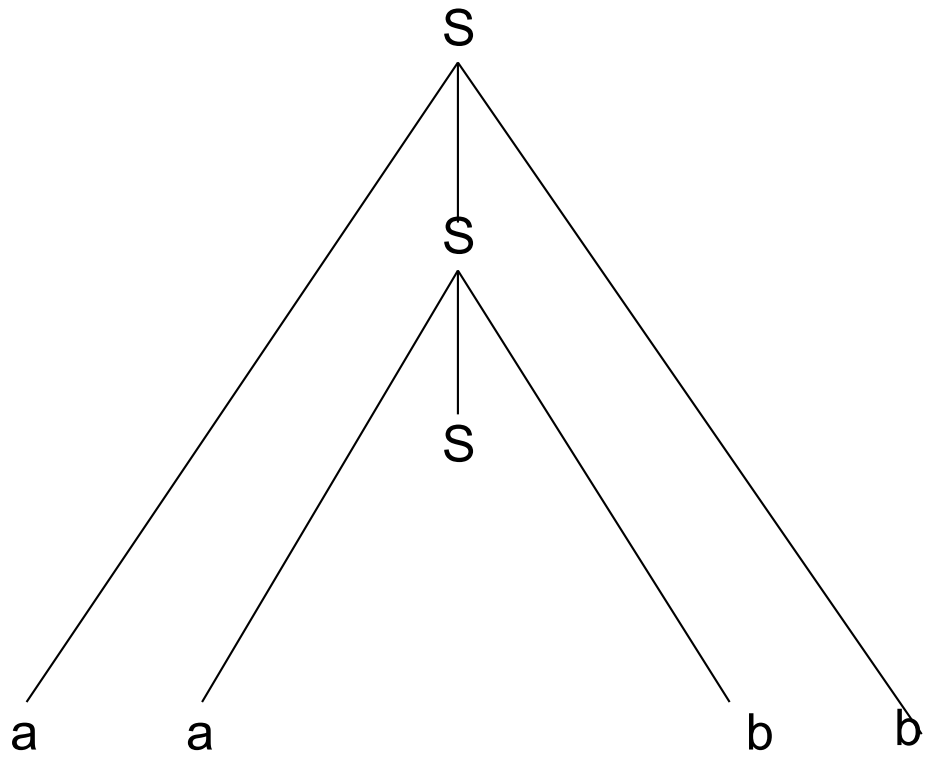
Exemple 1

- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$



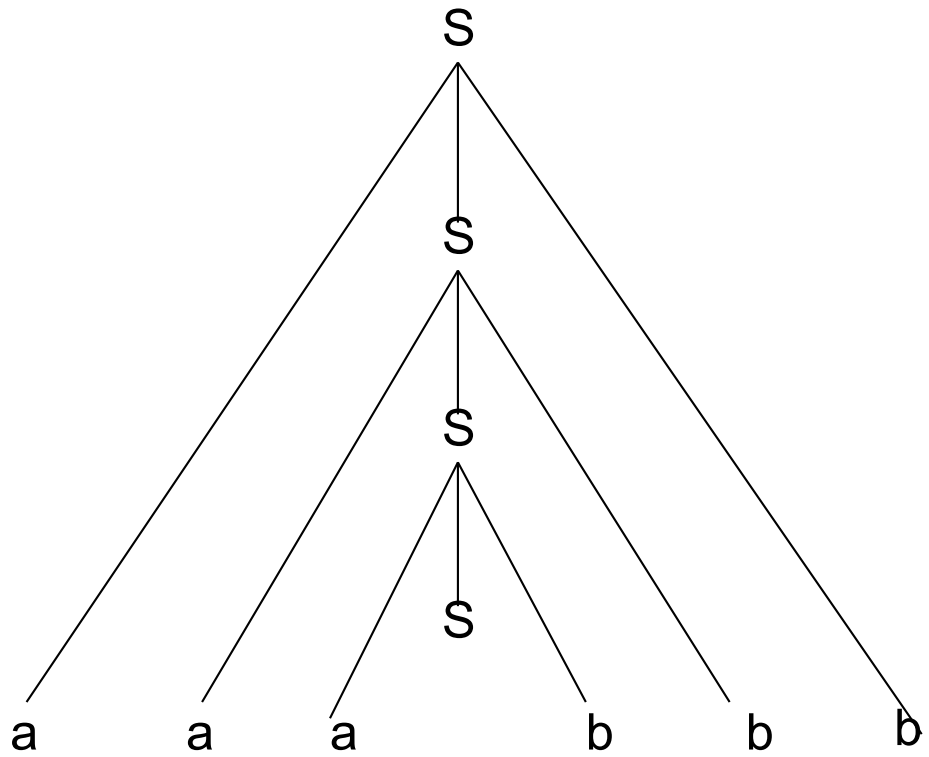
Exemple

- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$



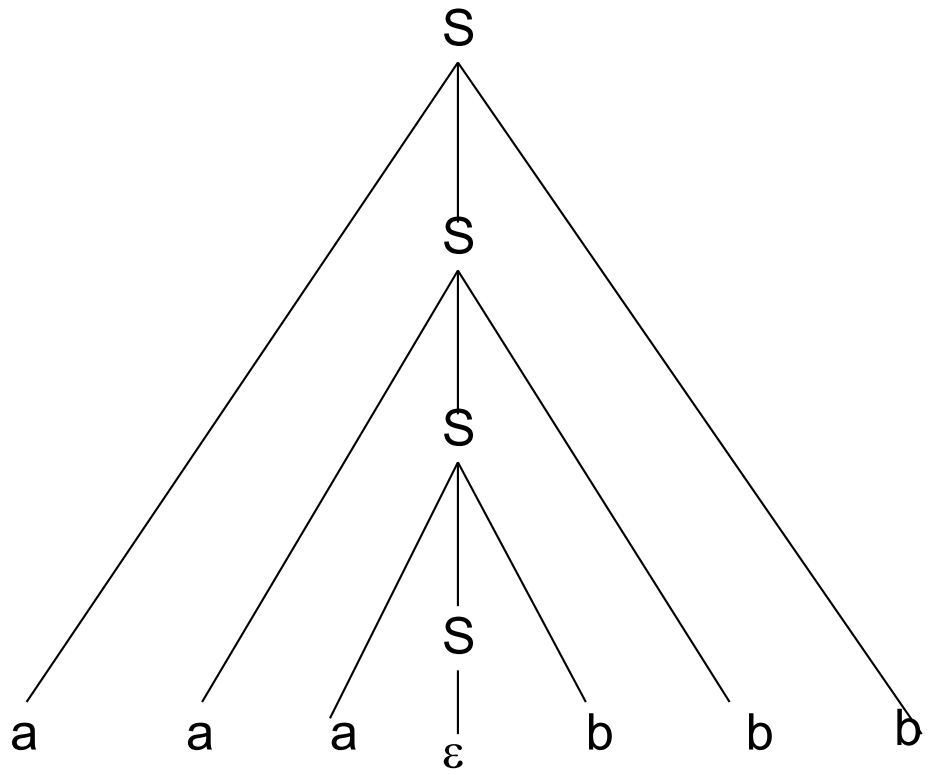
Exemple

- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$



Exemple

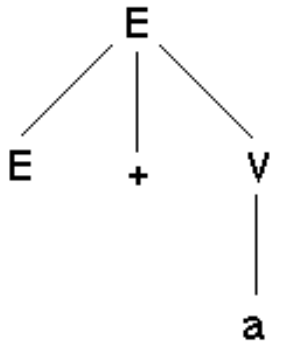
- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$

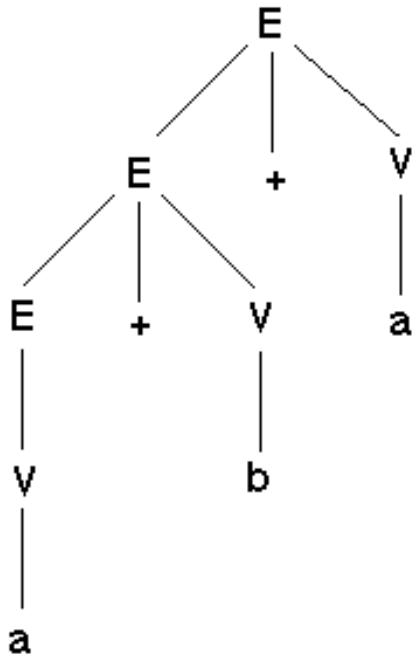


Exemple 2

- Considérons une grammaire pour les expressions arithmétiques additives :
  - E → V
  - E → E '+' V
  - V → a | b | c
- Ici *E* est le symbole de départ, et *a*, *b*, et *c* sont terminaux

Exemple 2





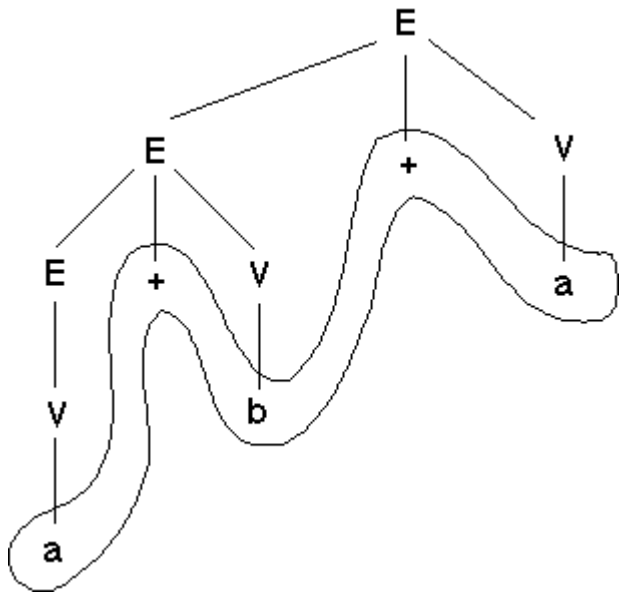
E → V
E → E '+' V
V → a | b | c

à gauche, un arbre de dérivation pour la grammaire ci-dessus au milieu, un arbre de dérivation complet

Un **arbre de dérivation complet** est un arbre dont la racine est étiquetée avec le symbole de départ de la grammaire et les feuilles sont étiquetées par les symboles terminaux.

Exemple 2

- Notez que les feuilles d'un arbre de dérivation complet, lorsque lues de gauche à droite, donne une chaîne qui est dans le langage généré par la grammaire.
- Par exemple, a+b+a est la séquence de feuilles pour le diagramme ci-dessous.

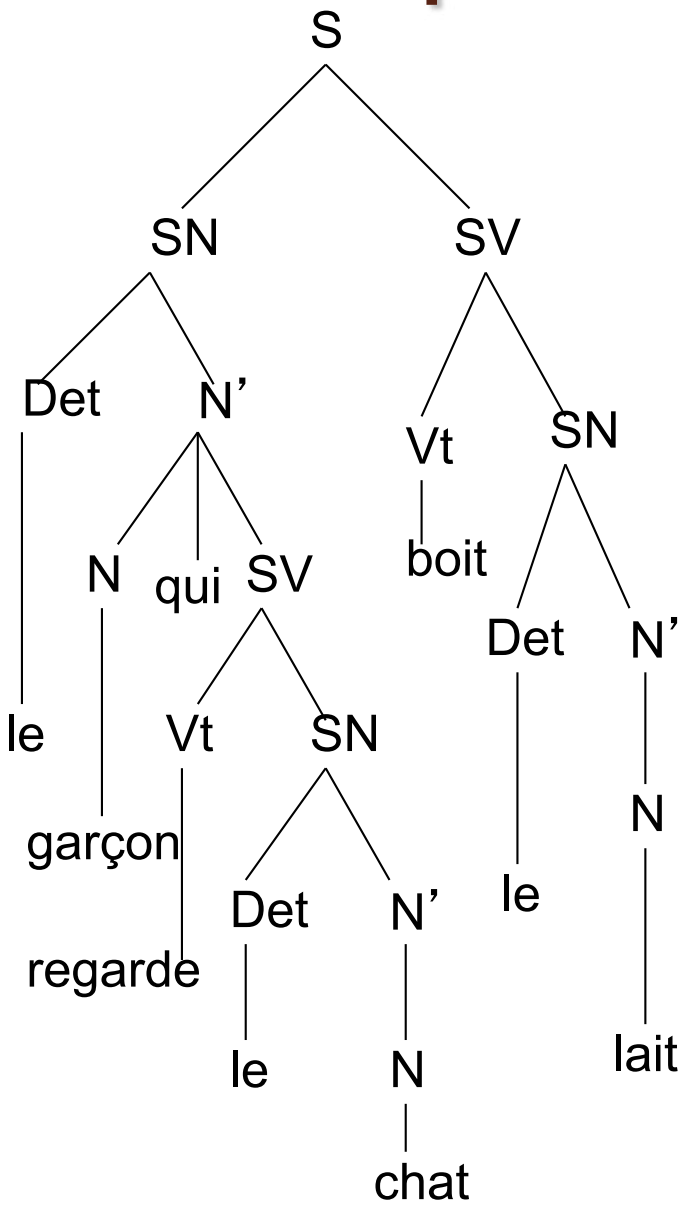


Dérivation de la chaîne a + b + a dans un arbre de dérivation complet

Exemple 3

- S → SN SV
- SN → Det N’
- N’ → N | N qui SV
- SV → Vt SN | Vi
- Det → un | le
- N → chat | garçon | lait | ballon
- Vt → regarde | boit | frappe
- Vi → boude

Exemple 3 : dérivations



- S → SN SV
- SN → Det N'
- N' → N | N qui SV
- SV → Vt SN | Vi
- Det → un | le
- N → chat | garçon | lait | ballon
- Vt → regarde | boit | frappe
- Vi → boude

Le garçon qui regarde le chat boit le lait

Symbole récursif

- SV => Vt SN => Vt Det N' => Vt Det N qui SV
- donc SV =>* Vt Det N qui SV

On dit que SV est un symbole **récursif**

- d'où : SV =>* Vt Det N qui SV =>* Vt Det N qui Vt Det N qui SV =>* … =>* Vt Det N qui Vt Det N qui … Vt Det N qui SV

➔ Langage infini

Structure en constituants

- Une sous-expression φ d'une phrase P est un **constituant** de type X **si et seulement si** φ est la concaténation des feuilles d'un sous-arbre dans l'arbre de dérivation de φ de racine X

- *Exemples* :
  - *regarde le chat* est un constituant de type SV,
  - *garçon qui regarde le chat* de type N' ,
  - *garçon qui regarde* **n'**est **pas** un constituant

- $[[[le]_{Det}\ [[garçon]_{N}$ qui $[[regarde]_{Vt}\ [[le]_{Det}\ [[chat]_{N}]_{N'}]_{SN}]_{SV}]_{N'}]_{SN}\ [[boit]_{Vt}\ [[le]_{Det}\ [[lait]_{N}]_{N'}]_{SN}]_{SV}]_{S}$

Ambiguïté syntaxique

- Certaines grammaires permettent que plusieurs arbres soient construits pour la même chaîne

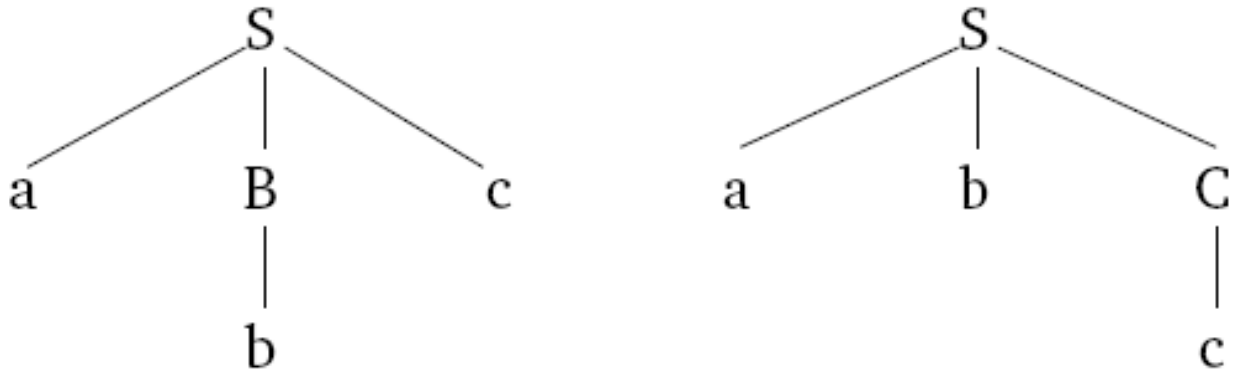
- De telles chaînes sont dites **ambiguës** par rapport à la grammaire

Ambiguïté syntaxique

- Par exemple, considérons la grammaire :

$$
\begin{aligned}
S &\rightarrow aBc \\
S &\rightarrow abC \\
B &\rightarrow b \\
C &\rightarrow c
\end{aligned}
$$

- Cette grammaire permet que deux arbres soient construits pour la chaîne *abc* :



Ambiguïté syntaxique

- Dans de tels cas, des règles différentes sont utilisées dans les deux dérivations

- Comparez ceci avec l'exemple précédent des dérivations gauche et droite où dans les deux cas les règles utilisées sont les mêmes et où simplement l'ordre d'application est différent

Note…

- Certains langages de programmation ont des grammaires ambigües
- Dans ce cas, de l'information sémantique est nécessaire à la sélection de la signification voulue d'une construction ambiguë.
- Par exemple, en C, l'expression suivante:

  x * y ;

  peut être interprétée :
  - Soit comme la déclaration d'un identifiant y de type pointeur sur x,
  - Soit comme une expression dans laquelle x est multipliée par y et dont le résultat n'est pas utilisé.

- Pour choisir entre les deux interprétations possibles, un compilateur doit consulter sa table de symbole pour trouver si `x` a été déclaré comme un nom de `typedef` qui est visible à cet endroit

Ambiguïté syntaxique

- $w \in L(G)$ est dit ambigu si et seulement si w admet plus d'un arbre de dérivation
- G est ambiguë si et seulement si elle engendre des mots ambigus
- L est ambigu si et seulement si L n'admet que des grammaires ambiguës

Exemple

- S → SN SV
- SN → Det N | Det N A | Det A N
- SV → V SN | Pro V
- Det → le | la
- Pro → le | la
- N → pilote | porte
- A → ferme
- V → ferme | porte

SN représente un syntagme nominal
SV représente un syntagme verbal
Pro sont les pronoms
Det sont les déterminants
N sont les noms
V sont les verbes
A sont les adjectifs

S → SN SV

SN → Det N | Det N A | Det A N

SV → V SN | Pro V

Det → le | la

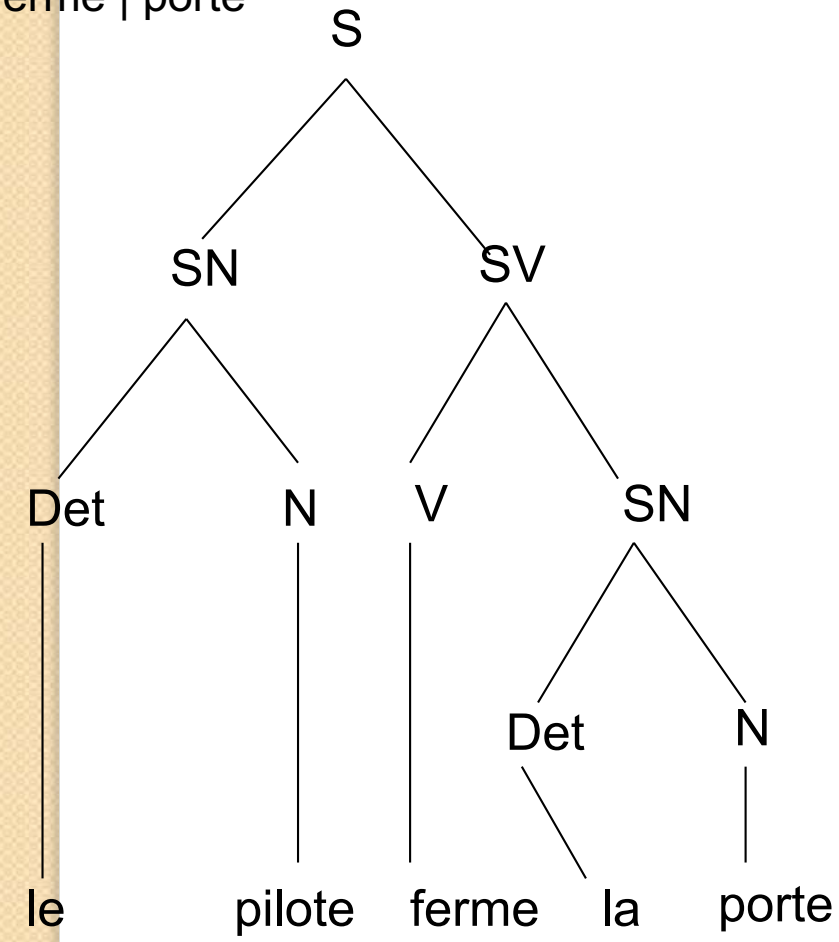
Pro → le | la

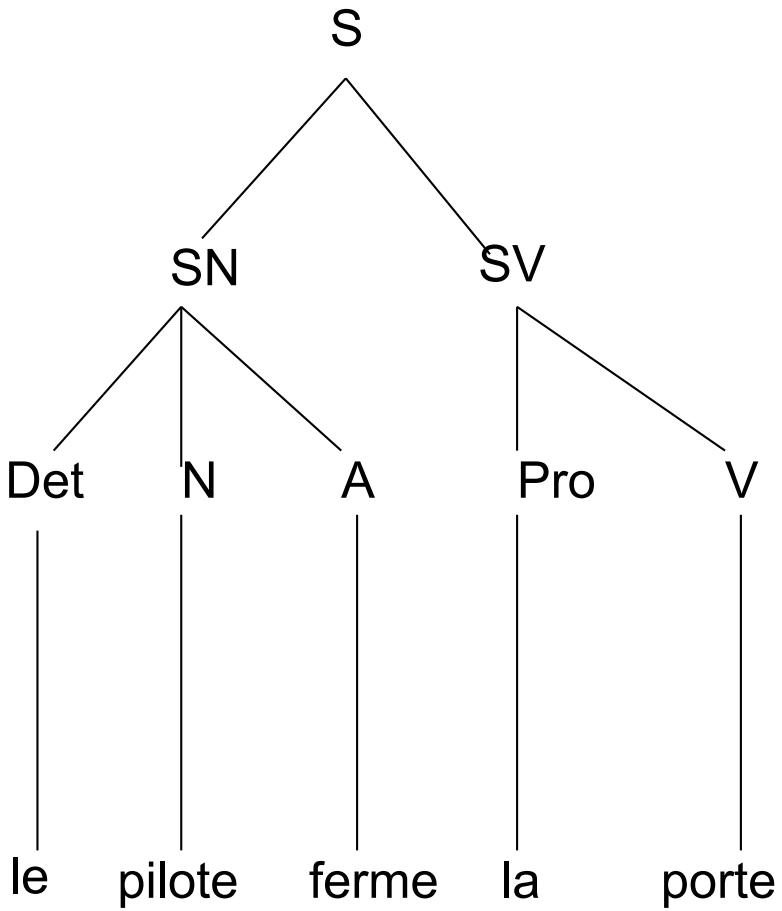
N → pilote | porte

A -→ ferme

V -→ ferme | porte

le pilote ferme la porte





Exemple 2

- La grammaire suivante va générer les expressions arithmétiques de la forme a, a+b, a+b+c, a*b, a+b*c, etc.

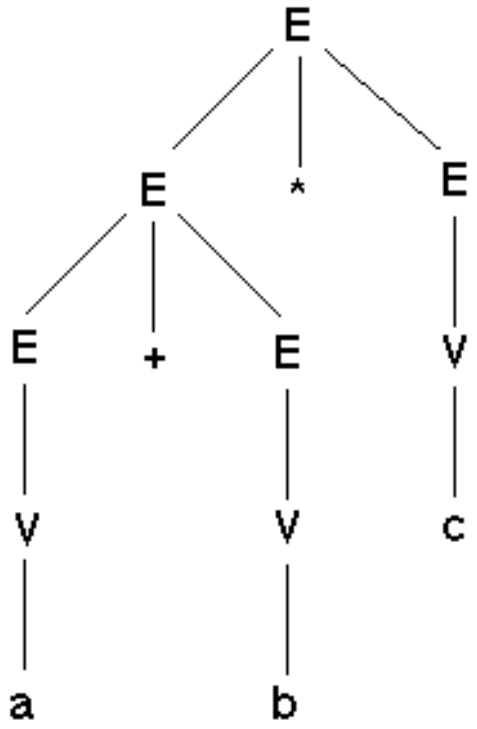
L'alphabet terminal est { a, b, c, + }
L'alphabet non-terminal est {E, V}
Le symbole de départ est E
Les productions sont :

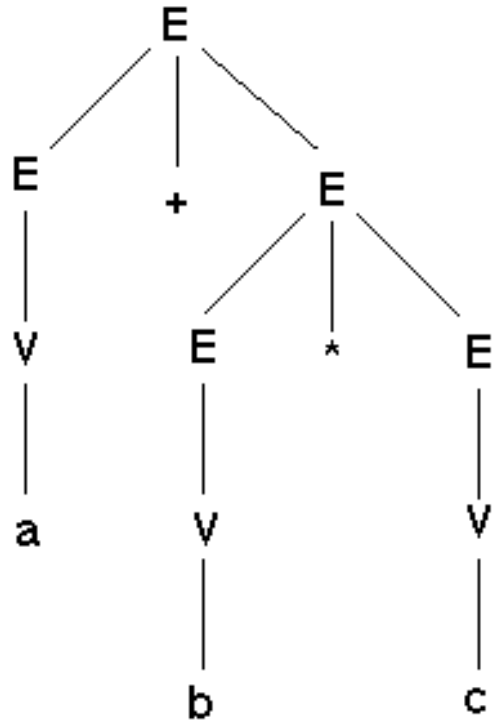
E → V
E → E ‘+’ E
E → E ‘* ’ E
V → a | b | c

Exemple 2

- Pour voir que la grammaire est ambiguë, nous pouvons construire deux arbres différents pour la chaîne a+b*c

E → V
E → E ‘+’ E
E → E ‘* ’ E
V → a | b | c





Exemple 2

- Par exemple, si *a*, *b*, et *c* avaient les significations 3, 5, et 17, l'arbre de gauche donnerait une signification de 136 pour l'expression arithmétique…
- Alors que l'arbre de droite donnerait une signification de 88
- Quelle signification est correcte? Avec une grammaire ambiguë, on ne peut pas vraiment dire...

Ambiguïté syntaxique

- Le problème d'ambigüité est parfois résolu par l'utilisation de priorités entre les règles à utiliser dans une dérivation quand il y a un choix
- Cependant, on résout le problème de la manière peut-être la plus propre en trouvant une autre grammaire qui génère le même langage, mais qui n'est pas ambiguë
- Ceci est habituellement obtenu en trouvant des productions différentes qui ne causent pas d'ambigüité

- Une grammaire non-ambiguë pour des expressions arithmétiques simples :

E → T
E → E '+' T
T → V
T → T '*' V
V → 'a' | 'b' | 'c'

E → V
E → E ‘+’ E
E → E ‘* ’ E
V → a | b | c

- Ceci implémente la priorité de * sur +.

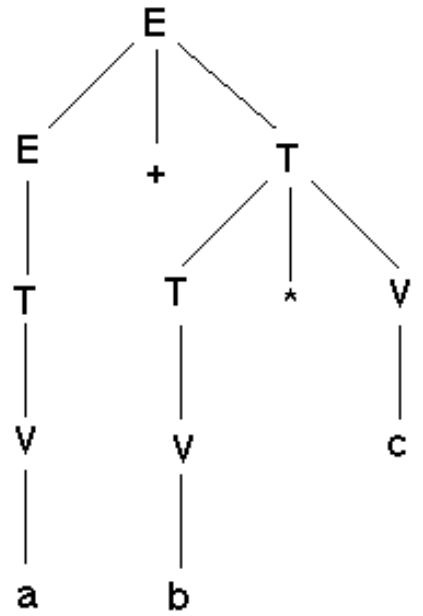
Exemple

- *Dérivation de* a+b*c *dans une grammaire non-ambiguë*

E → T
E → E '+' T
T → V
T → T '*' V

V → 'a' | 'b' | 'c'

- *Un seul arbre est généré*



Exemple

- La nouvelle grammaire fonctionne en imposant une priorité :
  - Elle nous interdit de développer une expression (c'est-à-dire une chaîne E-dérivée) en utilisant l'opérateur *
  - On peut seulement développer un terme (c'est-à-dire une chaîne T-dérivée) en utilisant l'opérateur *

E → T
E → E '+' T
T → V
T → T '*' V
V → 'a' | 'b' | 'c'

~~E → V
E → E '+' E
E → E '* ' E
V → a | b | c~~

Exercice

- Montrer que la grammaire hors-contexte :

  $A \rightarrow A + A \mid A - A \mid a$

  est ambiguë

Exercice

$A \rightarrow A + A \mid A - A \mid a$

- Elle est ambiguë car il y a 2 dérivations gauche pour la chaine a + a – a:

$A \rightarrow A + A$

$\rightarrow a + A$

$\rightarrow a + A - A$

$\rightarrow a + a - A$

$\rightarrow a + a - a$

$A \rightarrow A - A$

$\rightarrow A + A - A$

$\rightarrow a + A - A$

$\rightarrow a + a - A$

$\rightarrow a + a - a$

Exercice

- Il existe deux arbres pour la chaîne
  a + a − a