كلـيـــــــــــة العلوم
السملالية - مراكش
FACULTE DES SCIENCES
SEMLALIA-MARRAKECH

MARRAKECH
جامعة القاضي عياض
UNIVERSITÉ CADI AYYAD

Département Informatique

Licence SMI – S6

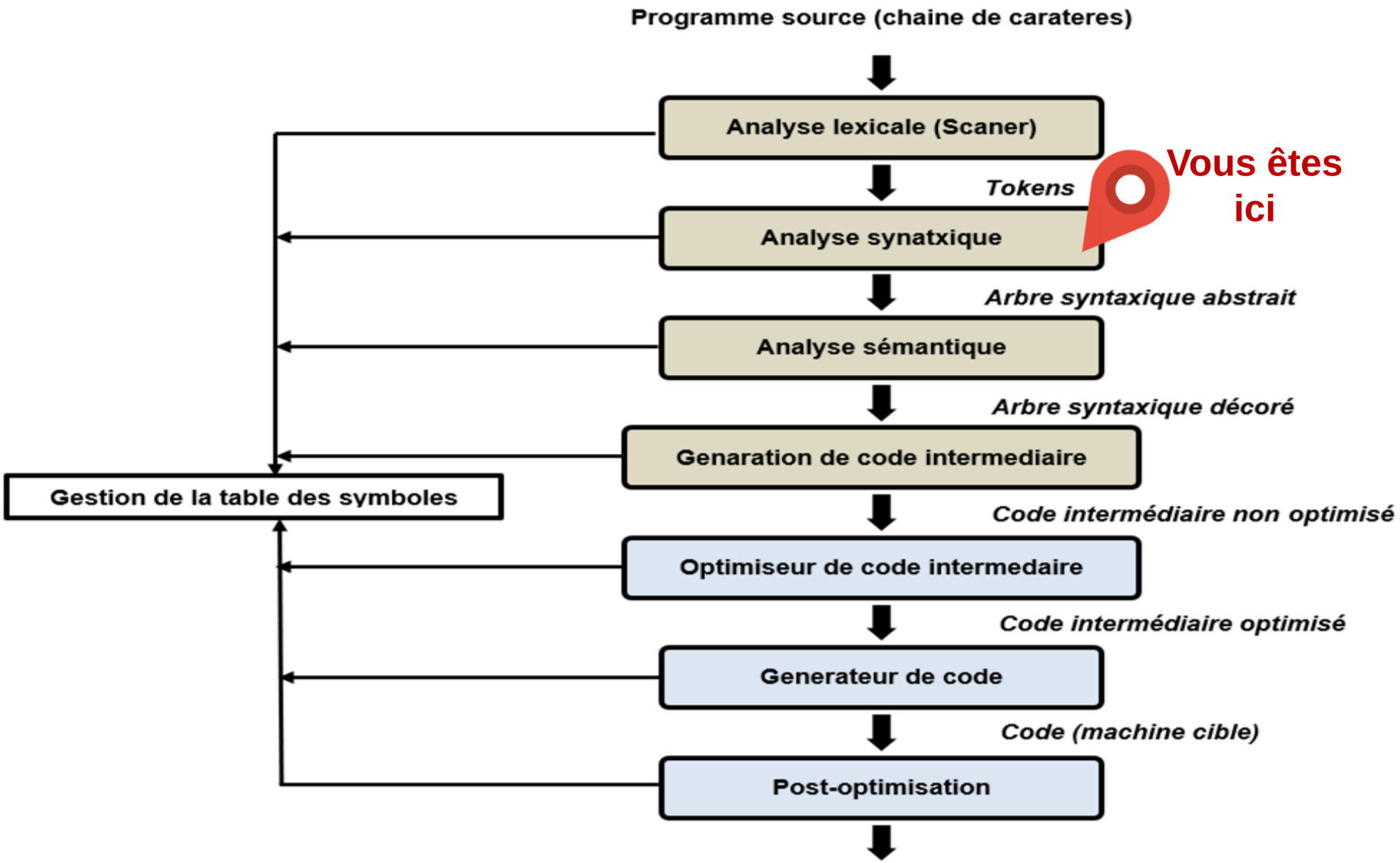
Compilation

Pr. Aimad QAZDAR

aimad.qazdar@uca.ac.ma

14 avril 2021

Rappel : Phases d'un compilateur



Chapitre 3 :

Analyse Syntaxique

Plan

Introduction

- Le résultat de l'analyse lexicale, (pg source) sous forme d'une suite de jetons plus la table des symboles à jour, est transmis à l'analyseur syntaxique.
- L'analyse syntaxique est l'une des opérations majeures d'un compilateur qui consiste à indiquer si un texte est grammaticalement correct et à en tirer une représentation interne, que l'on appelle arbre syntaxique.
- Ce dernier se charge de **vérifier si le programme est syntaxiquement correct** et **génère une représentation syntaxique équivalente du programme source**.
- Le rôle principal de l'analyse syntaxique est de vérifier si l'écriture du programme source conforme avec la syntaxe du langage à compilé.
- Cette dernière est spécifiée à l'aide d'une **grammaire hors contexte**.
- Il existe plusieurs méthodes d'analyse appartenant à l'une des deux catégories qui sont l'**analyse descendante** et l'**analyse ascendante**.
- Dans l'analyse descendante nous essayons de dériver à partir de l'axiome de la grammaire le programme source.
- D'une façon opposée, l'analyse ascendante établit des réductions sur les chaines à analyser pour aboutir à l'axiome de la grammaire

Grammaire

- Tout langage de programmation possède des règles qui indiquent la structure syntaxique d'un programme bien formé.
- Par exemple, en Pascal, un programme bien formé est composé de blocs, un bloc est formé d'instructions, une instruction de ...
- Etant donné un langage, Comment décrire tous les mots acceptables ? Comment décrire un langage ?

  ➔ **Les langages réguliers, qui s'expriment à l'aide d'expressions régulières**
- Mais la plupart du temps, les langages ne sont pas réguliers et ne peuvent pas s'exprimer sous forme d'une ER

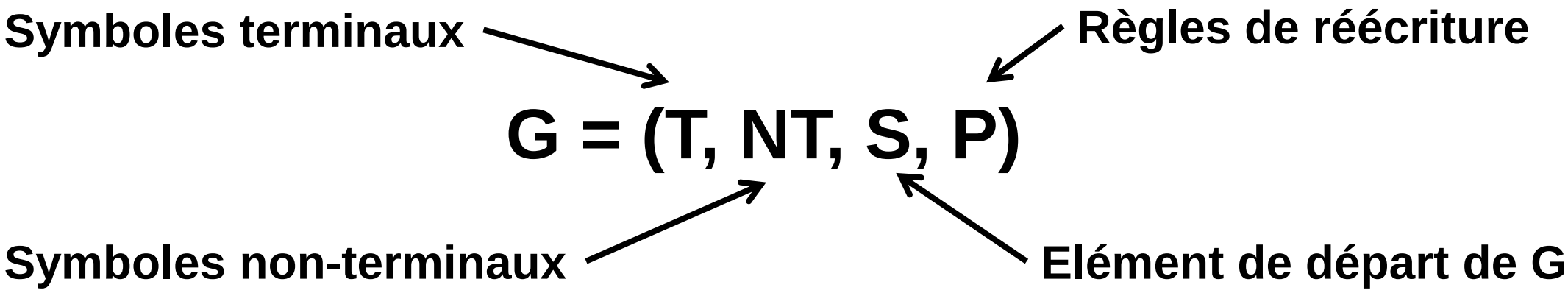
  ➔ **Besoin d'un outil plus puissant : les grammaires.**

Grammaire

- Exemples : une expression conditionnelle en C est :
  - if (**expression**) then **instruction** ;
  - Par exemple : if (**x<10**) then **a = a + b**;
- Il faut encore définir ce qu'est une **expression** et ce qu'est une **instruction**
- On distingue :
  - Les symboles **terminaux** : les lettres du langage (if ,then)
  - Les symboles **non terminaux** : les symboles qu'il faut encore définir (ceux en **rouge** dans l'exemple précèdent)

Grammaire

- Une grammaire hors contexte (context free grammar, CFG) est un quadruplet



Grammaire

- T est l'ensemble des symboles terminaux du langage.
  - Les symboles terminaux correspondent aux mots découvert par l'analyseur lexical «unité lexicale». if ,else sont des terminaux.
- NT est l'ensemble des symboles non-terminaux du langage.
  - Ces symboles n'apparaissent pas dans le langage mais dans les règles de la grammaire définissant le langage,
  - Ils permettent d'exprimer la structure des règles grammaticales.
- S ϵ NT est appelé l'élément de départ de G (ou axiome de G).
  - Le langage que G décrit (noté L(G)) correspond à l'ensemble des phrases qui peuvent être dérivées à partir de S par les règles de la grammaire.
- P est un ensemble de production (ou règles de réécriture)
  - de la forme N ➔α_1 α_2… α_n avec α_i ϵ (T U NT)*.
  - C.-à-d. que chaque élément de P associe un non terminal à une suite de terminaux et non terminaux.
  - Le fait que les parties gauches des règles ne contiennent qu'un seul non terminal donne la propriété "hors contexte" à la grammaire.

Grammaire

- Exemple G = (T, NT, S, P) avec
  - T = { il, elle, parle, est, devient, court, reste, sympa, vite }
  - NT ={ PHRASE, PRONOM, VERBE, COMPLEMENT, VERBETAT, VERBACTION }
  - S = PHRASE
  - P = { PHRASE PRONOM VERBE COMPLEMENT

    PRONOM ➔il | elle

    VERBE➔ VERBETAT | VERBACTION

    VERBETAT ➔ est | devient | reste

    VERBACTION ➔ parle | court

    COMPLEMENT ➔ sympa | vite }

Grammaire

- Quelques éléments grammaticaux de Pascal :
  - **programme ➔** en-tête de prog. block
  - **en-tête de prog. ➔ program** ident
  - **block ➔** declaration-list instruction-list
  - **declaration-list ➔** declaration | declaration; declaration-list
  - **declaration ➔ var** ident **: typeIdent;**
  - **instruction ➔ begin** instruction-list **end** ...
  - **instruction-list ➔** instruction | instruction ; instruction-list

```
1  program Somme;
2       var A : integer;
3       Var B : integer;
4       Var S : integer;
5
6       begin
7            readln(A, B) ;
8            S:= A + B;
9            writeln(S);
10      end.
```

Arbre de dérivation syntaxique

- On appelle dérivation l'application d'une ou plusieurs règles à partir d'un mot de$(T \cup NT)^+$
- On notera
  - ➔ une dérivation obtenue par application d'une seule règle de production,
  - ➔* une dérivation obtenue par l'application de n règles de production où n >= 0.
- Sur la grammaire de l'exemple :
  - PHRASE ➔ SUJET VERBE COMPLEMENT ➔*elle VERBETAT sympa
  - PHRASE ➔* il parle vite
  - PHRASE ➔*elle court sympa

Remarque :
- Il est possible de générer des phrases **syntaxiquement correctes** mais qui n'ont **pas de sens**.
- C'est l'**analyse sémantique** qui permettra d'éliminer ce problème

Arbre de dérivation syntaxique

- **Qu’est ce qu’un arabe de dérivation syntaxique :**
  - Un arbre de dérivation syntaxique pour la grammaire G de racine S ϵ NT et de feuilles wϵT* est un arbre ordonné dont la racine est S, les feuilles sont étiquetées par des terminaux formant le mot w et les nœuds internes par des non terminaux tels que Y est un nœud interne dont les p fils sont étiquetés par les symboles $a_1 \ldots a_p$ est une production de P
- **Dérivations gauches et droites :**
  - **Dérivation gauche** consiste à réécrire le symbole non-terminal le plus à gauche à chaque étape.
  - **Dérivation droite** consiste à réécrire le symbole non-terminal le plus à droite à chaque étape.

Arbre de dérivation syntaxique

- Soit la grammaire ayant S pour axiome et pour règles de production

  P ={ S➔ aTb | c

  T➔ cSS | S}

- Un arbre de dérivation pour le mot **accacbb** est :

  - ***dérivation gauche***

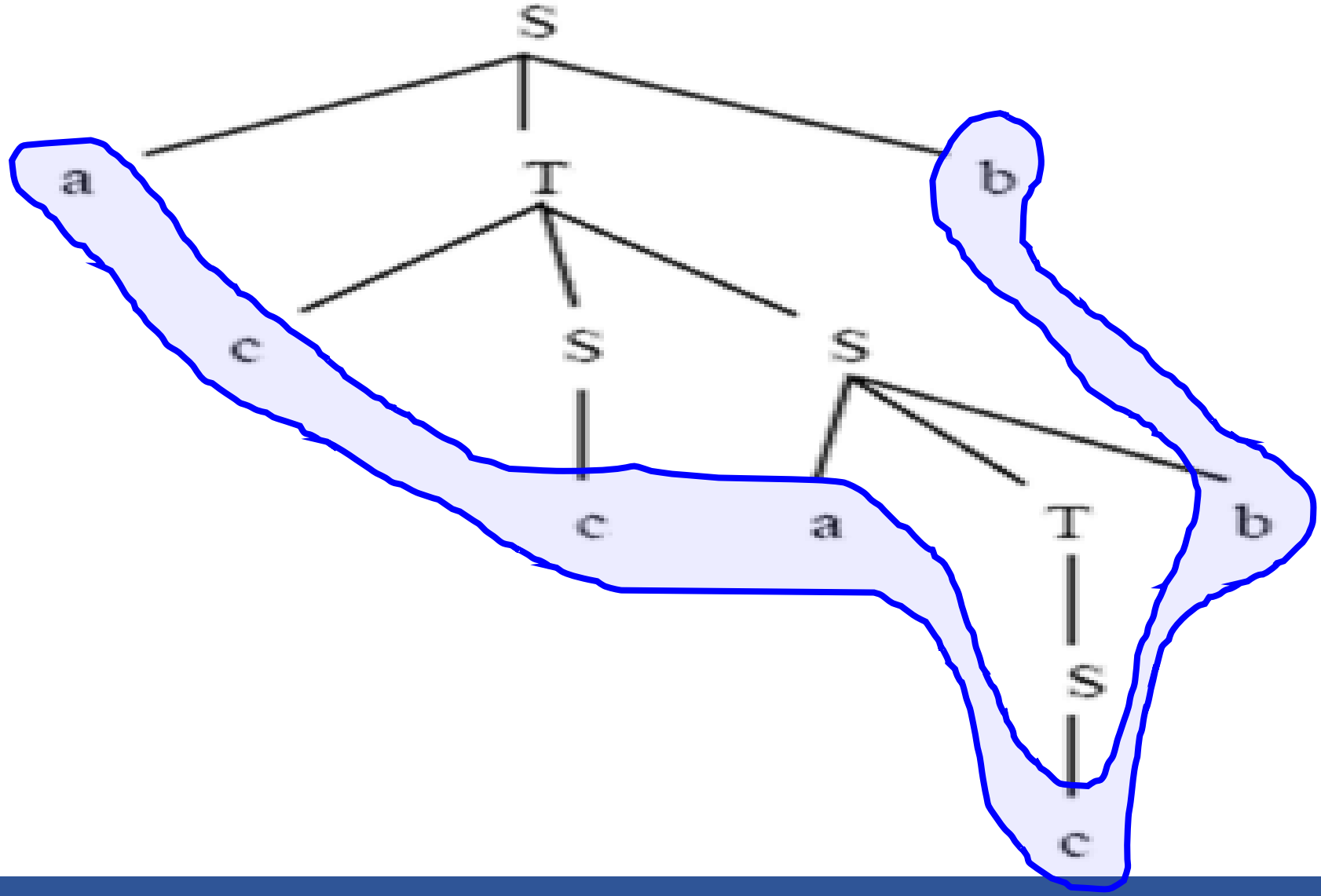
    S➔a**T**b➔ac**S**Sb➔acc**S**b➔acca**T**bb➔acca**S**bb➔accacbb ()

  - ***dérivation droite***

    S➔a**T**b ➔ acS**S**b ➔acSa**T**bb➔acSa**S**bb➔ ac**S**acbb➔ accacbb ()

Arbre de dérivation syntaxique

- Ces deux suites différentes de dérivations donnent le même arbre de dérivation pour le mot **accacbb**



Arbre de dérivation syntaxique

- Exemple la phrase :

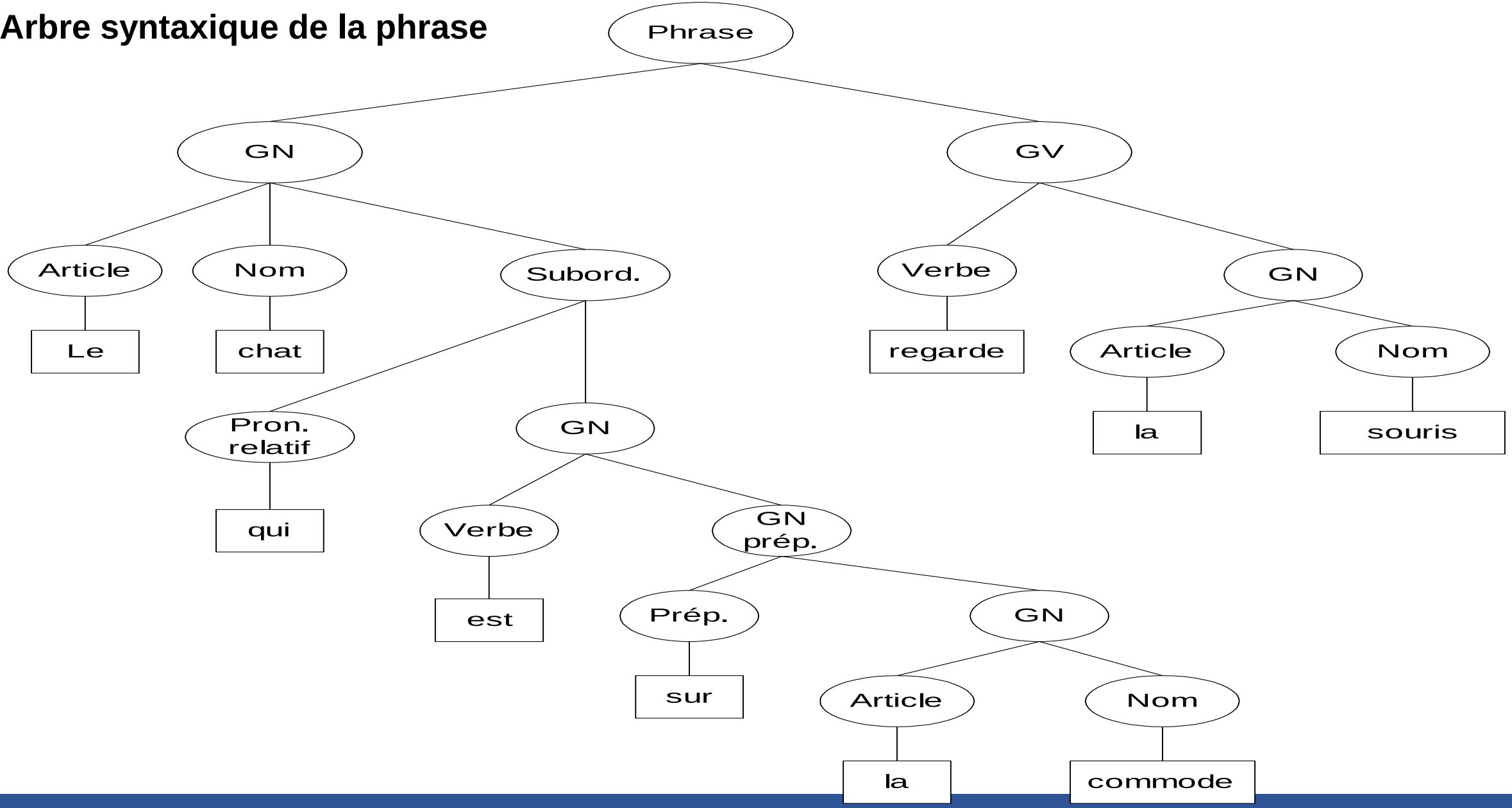
  « le chat qui est sur la commode regarde la souris »

<u>Règles de grammaire</u> :

<phrase> → <GN> <GV>
<GN> → <Article> <Nom> | <Article> <Nom> <Subord>
<Subord> → <Pron. relatif> <GV>
<GV> → <verbe> <GN> | <Verbe> <GN prep>
<GN Prep> → <Prep> <GN>
<Article> → **le** | **la** | **les** ...
<Nom> → **chat** | **commode** | **souris** | ...
<Verbe> → **est** | **regarde**
<Prep> → **sur** | **sous** | ...
<Pron. relatif> → **qui** | **que** | **dont** | ...

Arbre de dérivation syntaxique

Arbre syntaxique de la phrase



- Les expressions arithmétiques ont été particulièrement étudiées et posent de nombreux problèmes que nous verrons plus loin dans le cours.
- Exemple , pour exprimer une expression arithmétique de la forme : (a + b) * (a * 2+3) on peut utiliser deux grammaires :

GEA1 :

E → E+T | T

T → T*F | F

F → G ^ F | G

G → (E) | Id | Cste

GEA 1:

E ⇒ T ⇒ T * F ⇒ T * G ⇒ T * Id ⇒ F * Id ⇒ (E) * Id ⇒ (E + T) * Id ⇒* (G + T)*Id ⇒* (G + G) * Id ⇒ (Id + G) * Id

⇒ (Id + Id) * Id

GEA2 :

E → E + E | E * E | E ^ E | (E) | Id |Cste

GEA 2:

E ⇒(E)*E ⇒ (E + E) * Id ⇒ (E + Id) * Id

⇒ (Id + Id) * Id

Obtenir l'expression (Id + Id) * Id

GEA 1:

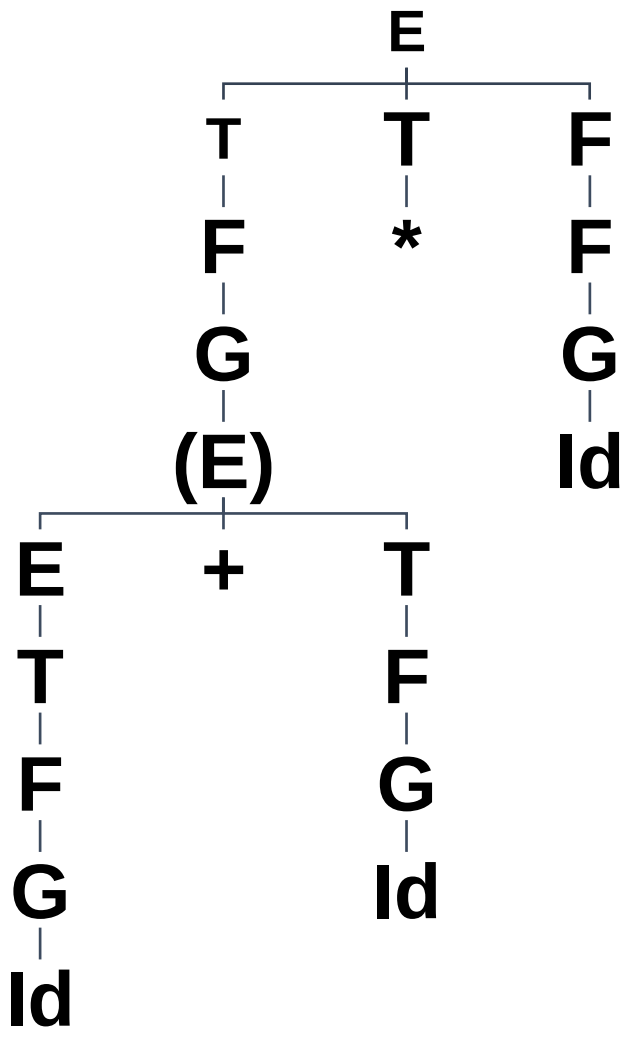
E ⇒ T ⇒ T * F ⇒ T * G ⇒ T * Id ⇒ F * Id ⇒
(E) * Id ⇒ (E + T) * Id ⇒* (G + T)*Id ⇒*
(G + G) * Id ⇒ (Id + G) * Id

⇒ (Id + Id) * Id

GEA 2:

E ⇒(E)*E ⇒ (E + E) * Id ⇒ (E + Id) * Id

⇒ (Id + Id) * Id



➔ **Ces dérivations doivent être considérées comme équivalentes.**

Exercice

- **Exo 1:**
  - Décrivez la grammaire qui engendre les instructions conditionnelles IF ...THEN….ELSE…
  - Donner la suite de dérivation pour générer le mot **ω = if a then if b then c else d**.
  - Donner l'arbre syntaxique correspondant au mot ω.

G=({S}, {if, then, else, condition, instruction}, P, A)

Avec P:

1) A→ if condition then S

2) S→if condition then S else S

3) S → instruction

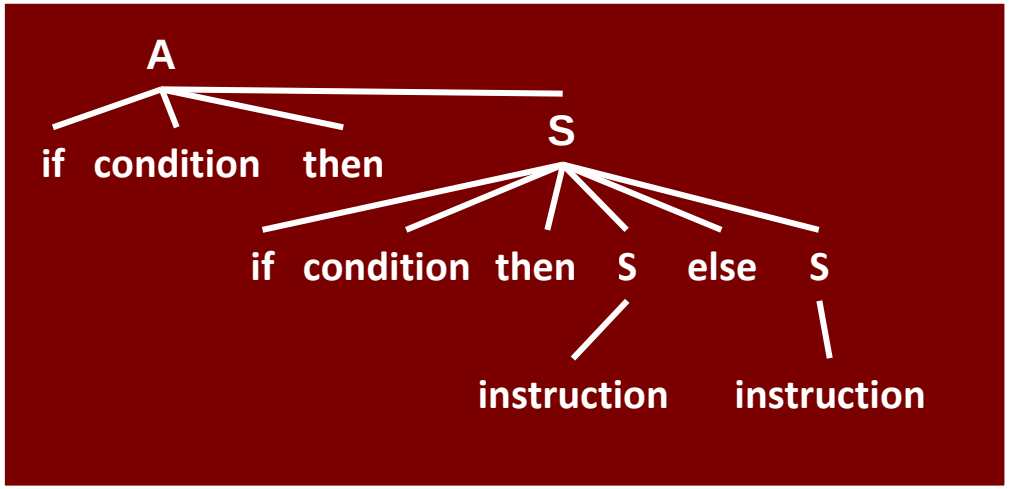
La suite de dérivation pour le mot ω est :

A ⇒ if condition then S

⇒ if condition then if condition then S else S

⇒* if condition then if condition then instruction else instruction

Pour retrouver le mot ω il suffit de parcourir les feuilles de l'arbre de gauche à droite.



Exercice

Exo 2: soit la grammaire

<instruction> → <instruc. conditionnelle> | ...

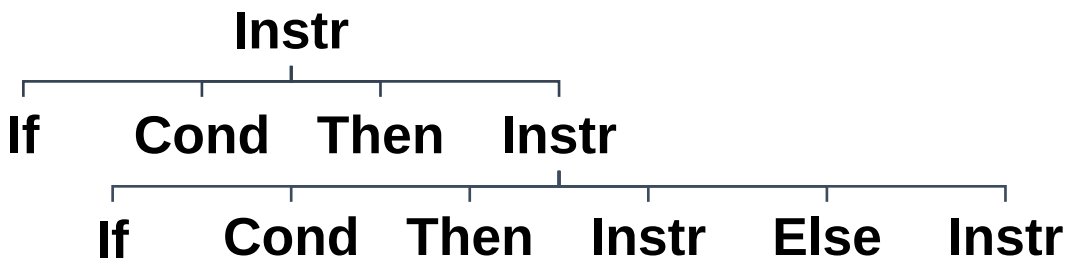
<instruc. conditionnelle> → **if** <condit> **then** <instruction> |**if** <condit> **then** <instruction> **else** <instruction>

Proposer un arbre de derivation de la phrase :

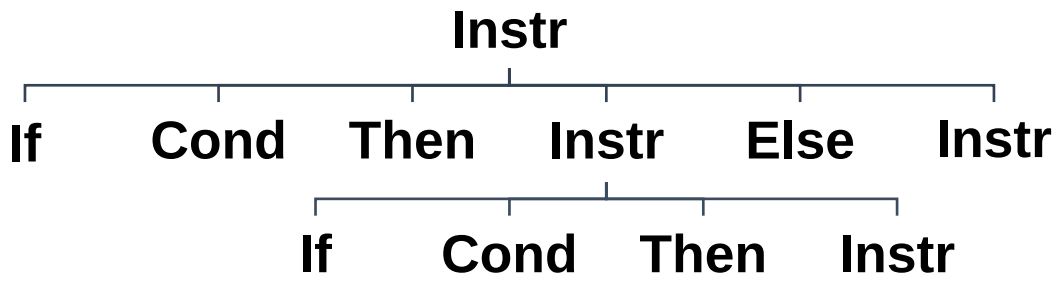
if C1 **then** if C2 **then** S1 **else** S2

- La phrase : **if** C1 **then** if C2 **then** S1 **else** S2 possède les deux arbres de dérivations suivants :

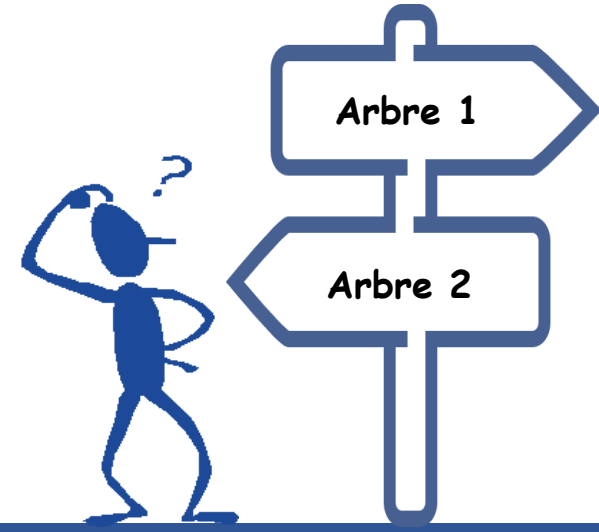
Arbre 1



Arbre 2



Une grammaire est dite ambiguë s'il existe une phrase de L(G) ayant plusieurs arbres syntaxiques



Suppression des ambiguïtés

- Quelque fois, une grammaire peut être réécrite pour éliminer les ambiguïtés.
- Dans l'exemple « **If Then Else** » il suffit d'ajouter des règles de réécriture permettant d'appliquer le principe : « **chaque *else* doit être associé avec le *then* le plus près** ».
- En incorporant ce principe dans la grammaire on obtient :
  - <instruction> ➔ <instr. assoc.> | <instr. non-assoc>
  - <instr. assoc.> ➔ if <condit> then <instr. assoc.> else <instr. assoc> | <autres instr.>
  - <instr. non-assoc> ➔ if <condit> then <instruction> | if <condit> then <instr. assoc.> else <instr. non-assoc>

Problème

- L'analyseur syntaxique reçoit une suite d'unités lexicales de la part de l'analyseur lexical.
- Il doit dire si cette suite (phrase) est syntaxiquement correct, c'est à dire si elle est générée par la grammaire du langage.

Solution

- Essayer de reconstruire l'arbre syntaxique de la phrase à partir d'une grammaire (non-ambiguë).
- Si on y arrive, alors la phrase est syntaxiquement correct, sinon elle est incorrect.

Types de Processus d'analyse

- Il existe deux approches pour construire l'arbre de dérivation : une **méthode descendante** et une **méthode ascendante**:
- Les **méthodes d'analyse descendante** :
  - Elles construisent l'arbre syntaxique en **préordre**, c'est-à-dire du haut en bas, en partant de la racine (l'axiome) vers les feuilles (les lexemes).
- Les **méthodes d'analyse ascendante :**
  - Elles construisent l'arbre syntaxique en **postordre**, c.-à-d. du bas en haut, en partant des feuilles vers la racine.

Analyse Ascendante

- Connu aussi sous le nom d'**analyse par décalage-réduction**.
- **But** : construire un arbre d'analyse pour une chaîne source en commençant par les feuilles (le bas) et en remontant vers la racine (le haut).
- C.-à-d. la réduction d'une chaîne m vers l'axiome de la grammaire.
- Dans ce type d'analyse, on essaye d'appliquer le membre droit d'une règle et à le remplacer par le non-terminal gauche correspondant.
- L'opération est donc inverse de celle la dérivation.
- La principale difficulté consiste à trouver le bon membre droit à appliquer.
- Elle est à la fois plus puissante mais aussi plus complexe à mettre en œuvre que l'analyse descendante (sans retour arrière).
- Cette analyse effectue un parcours **postfixe de l'arbre**.

Analyse Ascendante

- Principe :
  - A chaque étape de « réduction » une sous chaîne particulière correspondant à la partie droite d'une production est remplacée par le symbole de la partie gauche de cette production.
  - Si la sous chaîne est choisie correctement à chaque étape ➔ dérivation droite est ainsi élaborée en sens inverse.
- Exemple : soit la grammaire :
  - S➔aABe
  - A ➔Abc| b
  - B➔ d

➔ La phrase **abbcde** peut être réduite vers S par les étapes suivantes :

abbcde
aAbcde
aAde
aABe
S

➔ On cherche une sous-chaîne qui correspond à la partie droite d'une production.

Analyse Ascendante

- Parmi les techniques d'analyse ascendante pour analyser les grammaires non contextuelles, on a **les analyseurs LR** (L: parcours du mot de Gauche vers droite, R: dérivation Droite inverse).
- **La principale difficulté consiste à trouver le bon membre droit à appliquer.**
- **Cette analyse est à la fois plus puissante mais aussi plus complexe à mettre en œuvre que l'analyse descendante.**

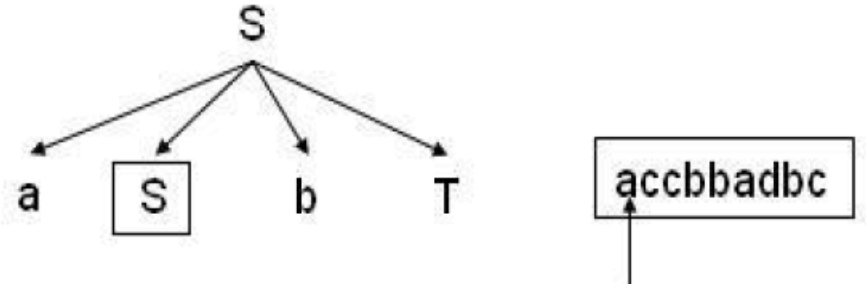
Analyse Descendante

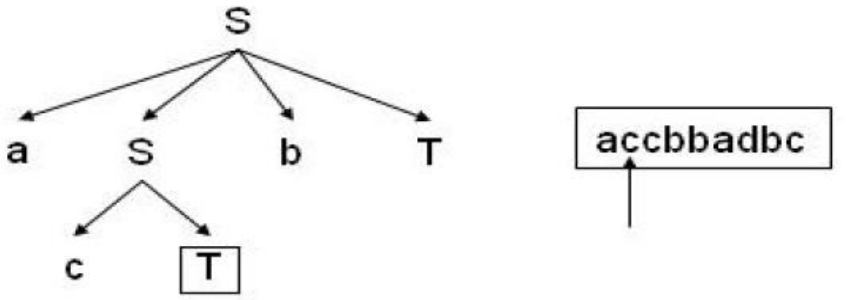
- **Principe :**
  - On construit l'arbre de dérivation en descendant de la racine (c.-à-d. l'axiome de départ) vers les feuilles (c.-à-d. les unités lexicales).
  - Cette analyse correspond à un parcours descendant gauche (parce qu'on lit de gauche à droite).
  - Il s'agit de deviner à chaque étape quelle est la règle qui sert à engendrer le mot que l'on lit.
  - Cette analyse effectue un **parcours préfixe de l'arbre.**

- Exemple :

Soit
- S➔ aSbT|cT|d
- T➔ aT|bS|c

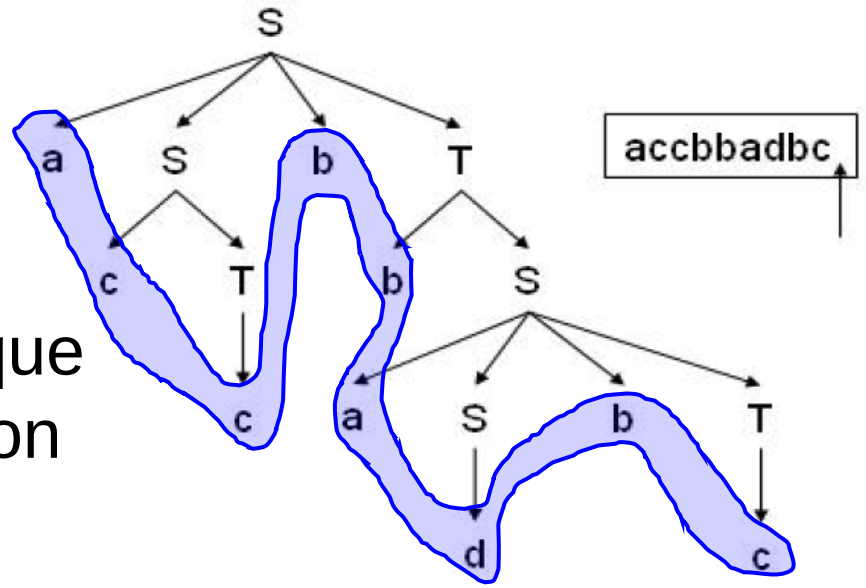
avec le mot **w=accbbadbc**
On part avec l'arbre contenant le seul sommet *S*
La lecture de la première lettre du mot (*a*) nous permet d'avancer la construction





On a trouvé un arbre de dérivation, donc le mot appartient au langage.

Sur cet exemple, c'est très facile parce que chaque règle commence par un terminal différent, donc on sait immédiatement laquelle prendre.



Analyse Descendante

- Exemple 2 : $S \rightarrow aAb$, $A \rightarrow cd \mid c$

  avec le mot $w=acb$

  On se retrouve avec ??? Déduire le pb.

- Faisons l'analyse du mot w = acb. On part de la racine S.
  - On lit le premier lexeme du mot : a.
  - Il y a une seule production a appliquer : **S** → **aAb** et le prochain NT à développer est **A**, ce qui donne : **S**→ **aAb**
  - En lisant le prochain lexeme **c**, on ne sait pas s'il faut prendre la production **A** →**cd** ou la production **A** → **c**.
  - Pour le savoir, il faut lire aussi le lexeme suivant **b**, ou alors, il faut donner la possibilité de faire des retours en **arrière** (**backtracking**) :
  - On choisit la production **A** → **cd**, on aboutit à un échec, ensuite on retourne en arrière et on choisit la production **A** → **c** qui convient cette fois-ci.

Analyse Descendante

- Malheureusement, pour que l'on puisse appliquer une analyse descendante récursive, il faut que:
  - Toutes les règles de la forme: $\mathbf{A \rightarrow a_1 \mid a_2 \mid ... \mid a_n}$ puissent être telles que l'on sache toujours quelle règle choisir.
  - Pour un non-terminal donné, toutes les règles associées commencent par un symbole terminal.

Analyse Descendante

- En pratique, les langages de programmation « bien écrits » les mots clés (ou des classes lexicales) permettent de distinguer parmi les règles sans problèmes.
- Par exemple, dans :
  - <instruction> ➔ for <var> := <expr> do <instr> | if <cond> then <expr> else <expr> | while <cond> do <instruction> | begin <instr-list> end |

    <procedure>(<arg-list>) | <ident> := <expr>
  - Les mots clés permettent de reconnaître sans ambiguïté le type d'instruction. Il en est de même des mots clés procedure, function, var, type, const qui sont placés devant ce qu'ils introduisent.
  - Les classes lexicales <procedure> et <ident> agissent comme des mots clés pour lever l'ambiguïté sur les règles.
  - Mais cela ne suffit pas toujours.

Analyse Descendante

- Il existe un ensemble de transformations que l'on doit appliquer aux règles pour qu'elles aient le bon format :
  1. **Récursivité à gauche**,
  2. **Traitement des règles vides**,
  3. **Factorisation**,
  4. **Expansion partielle**

Analyse prédictive

- **Problématique**: Soit : S→aSb|aSc|d avec le mot w=aaaaaaadbbcbbbc .
- Pour savoir quelle règle utiliser, il faut cette fois-ci connaître aussi **la dernière lettre du mot**.
- L'analyse prédictif permet la construction descendante d'un arbre syntaxique en partant de l'axiome pour arriver au mot à analyser.
- Le principe de l'analyse prédictif c'est d'obtenir suffisamment d'informations pour choisir correctement la règle de dérivation à appliquer à tout moment afin de fabriquer un arbre.

Analyse prédictive

- Ce qui serait pratique, ça serait d'avoir une table qui nous dit : quand je lis tel caractère et que j'en suis à dériver tel symbole non-terminal, alors j'applique telle règle et je ne me pose pas de questions.
- Ça existe, et ça s'appelle une **table d'analyse**.

	nb	**+**	*****	**(**	**)**	**$**
E	E→TE’			E→TE’		
E'		E’→+TE’			E’→ε	E’→ε
T	T→FT’			T→FT’		
T'		T’→ε	T’→*FT’		T’→ε	T’→ε
F	F→nb			F→(E)		

Table d'analyse

- **Déf.:** Une table d'analyse est un tableau M à deux dimensions qui indique pour chaque symbole non-terminal A et chaque symbole **terminal a** ou **symbole $**, quelle **la règle de production appliquée**.

Construction de la table d'analyse

- Pour chaque production **A→α** faire :
  1. pour tout a ∈ PREMIER(α) (et a≠ε), rajouter la production A→α dans la case M[A,a]
  2. si ε∈ PREMIER(α), alors pour chaque b ∈ SUIVANT(A) ajouter A→α dans M[A,b]
- Chaque case M[A,a] vide est une erreur de syntaxe

➔ Pour construire une table d'analyse, on a besoin des ensembles **PREMIER** et **SUIVANT**

Table d'analyse : construction des Premiers

- **Calcul de PREMIER**
  - Pour toute chaîne α composée de symboles terminaux et non-terminaux, on cherche PREMIER(α) : l'ensemble de tous les terminaux qui peuvent commencer une chaîne qui se dérive de α.
  - C'est à dire que l'on cherche toutes les lettres a telles qu'il existe :

    une dérivation $\alpha \rightarrow^* a\beta$ (β étant une chaîne quelconque composée de symboles terminaux et non-terminaux).

Exemple :

S ➔ Ba

B ➔ cP|bP|P|ε

P➔ dS

$S \rightarrow^* a$ donc $a \in$ PREMIER(S)

$S \rightarrow^* cPa$ donc $c \in$ PREMIER(S)

$S \rightarrow^* bPa$ donc $b \in$ PREMIER(S)

$S \rightarrow^* dSa$, donc $d \in$ PREMIER(S)

Il n'y a pas de dérivation $S \rightarrow^* \varepsilon$

Donc PREMIER(S) ={a,b,c,d}

Table d'analyse : construction des Suivants

- **Calcul de SUIVANT**
  - Pour tout non-terminal A, on cherche SUIVANT(A) : l'ensemble de tous les symboles terminaux a qui peuvent apparaître immédiatement à droite de A dans une dérivation : $S \rightarrow^* \alpha A \beta$

Exemple :

S ➔ Ba
B ➔ cP|bP|P|ε
P➔ dS

a ∈ SUIVANT(B) car il y a la dérivation :

$S \rightarrow B\mathbf{a}$

a ∈ SUIVANT(P) car il y a la dérivation :

$S \rightarrow^* P\mathbf{a}$

a ∈ SUIVANT(S) car il y a la dérivation :

$S \rightarrow^* dS\mathbf{a}...$

Exercices

- Calculer les premiers et les suivants des non-terminaux des grammaires suivantes :

G1 =({ S,A,B},{a,b, c, d,e},P,S}

Avec P:

S ➔ aSb | cd | SAe

A ➔aAdB |ε

B ➔ bb

	PREMIER	SUIVANT
S	***a c***	***$ b a e, d***
A	***a ε***	***e d***
B	***b***	***e d***

G2=({ E,T,F},{a,+,*,(,,)},P,E}

Avec P :

E➔E+T |T

T➔T*F | F

F ➔ (E) | a

	Premier	Suivant
E	**(, a**	**$, +,)**
T	**(, a**	***, +, $,)**
F	**(, a**	***, +, $,)**

Construction de la table d'analyse

Dans la table d'analyse M, chaque M[A, a] indique quelle production utiliser

- **Pour chaque production A → α faire:**
  1. **pour tout a ∈ PREMIER(α) (et a≠ε), rajouter la production A → α dans la case *M*[*A*,*a*]**
  2. **si ε∈ PREMIER(α), alors pour chaque b ∈ SUIVANT(*A*) ajouter A → α dans *M*[*A*,*b*]**
- **Chaque case *M*[*A*,*a*] vide est une erreur de syntaxe**

Exemple sur la grammaire:

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | nb

On obtient la table suivante :

PREMIER(E) = PREMIER(T) = { (, nb}
PREMIER(E') = {+, ε}
PREMIER(E) = PREMIER(F) = { (, nb}
PREMIER(T') = {*, ε}
PREMIER(F) = { (, nb}

SUIVANT(E) ={ $,) }
SUIVANT(E') ={ $,) }
SUIVANT(T) = SUIVANT(T') = { +,) , $ }
SUIVANT(F) = {*,) , +, $}

	nb	+	*	(	)	$
E	**E→TE'**			**E→TE'**		
E'		**E' →+TE'**			**E' →ε**	**E' →ε**
T	**T→FT'**			**T→FT'**		
T'		**T' →ε**	**T' →*FT'**		**T' →ε**	**T' →ε**
F	**F→nb**			**F→(E)**		

Construction de la table d'analyse

- Construire les tables d’analyse des grammaires suivantes :

G1 =({ S,A,B},{a,b, c, d,e},P,S}

Avec P:

S ➔ aSb | cd | SAe

A ➔aAdB |ε

B ➔ bb

G2=({ E,T,F},{a,+,*,(,,)},P,E}

Avec P :

E➔E+T |T

T➔T*F | F

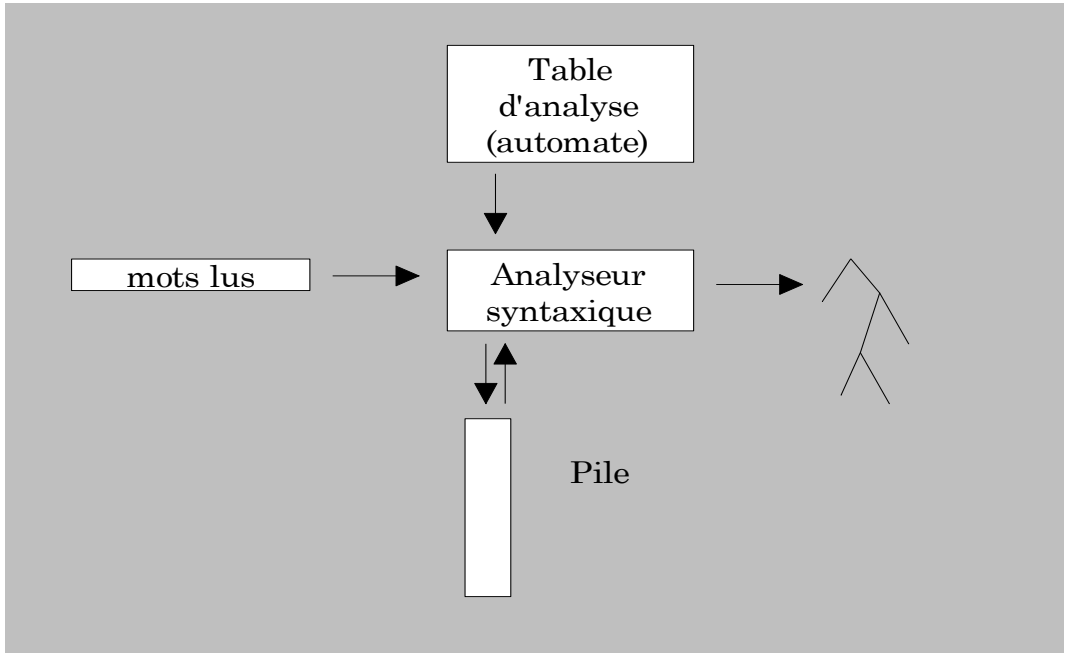
F ➔ (E) | a

Grammaire LL(1)

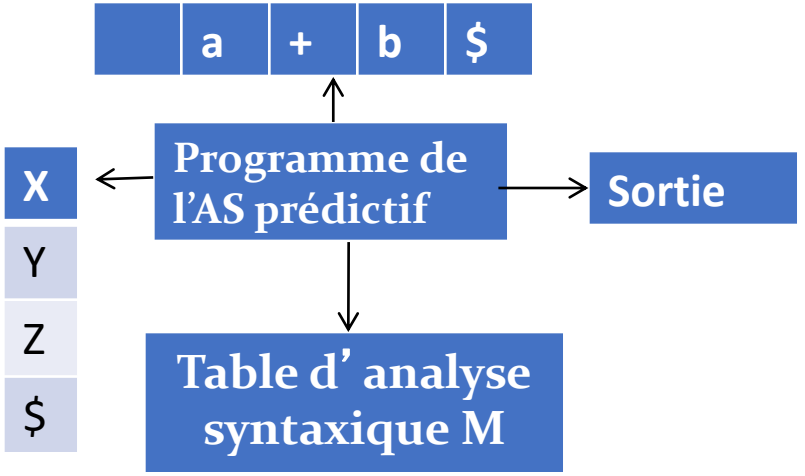
- Déf. : On appelle **grammaire LL(1)** une grammaire pour laquelle la table d'analyse décrite précédemment n'a aucune case définie de façon multiple.

L’analyse prédictive non récursive

- Maintenant qu'on a la table, comment l'utiliser pour répondre aux questions :
  - analyser un mot ω?
  - ω appartient-il à L(G) ?
  - si oui donner une dérivation.



- Un analyseur prédictif se compose :
  - d'une pile (pour contenir les symboles en cours)
  - et d'une table qui contient les relations entre terminaux courants, symboles terminaux en entrée et règles à appliquer.



Analyseur syntaxique LL(1)

Algorithme :

On utilise une **pile**.

- données : mot *m*, table d'analyse *M*
- initialisation de la pile : *S*, $

et un pointeur *ps* sur la 1ère lettre de *m*

	nb	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→nb			F→(E)		

répéter
  Soit *X* le symbole en sommet de pile
  Soit *a* la lettre pointée par *ps*
  Si *X* est un non terminal **alors**
    Si *M[X,a]* = *X →Y1…Yn* **alors**
      enlever *X* de la pile
      mettre dans la pile : *Yn* puis *Yn*-1 puis..*Y*1
      émettre en sortie la production *X →Y1…Yn*
      sinon
        ERREUR
      finsi
  Sinon
    Si *X*=$ **alors**
      Si *a*=$ **alors** ACCEPTER
      Sinon ERREUR
      Finsi
    Sinon
      Si *X*=*a* **alors**
        enlever *X* de la pile
        avancer *ps*
      Sinon
        ERREUR
      finsi
    finsi
  finsi
jusqu'à ERREUR ou ACCEPTER

Analyseur syntaxique LL(1): exemple

Sur notre exemple (grammaire *E*,*E'*,*T*,*T'*,*F*), soit le mot : *m*=3+4*5

E → TE’
E’ → +TE’ | ε
T → FT’
T’ → *FT’ | ε
F → (E) | nb

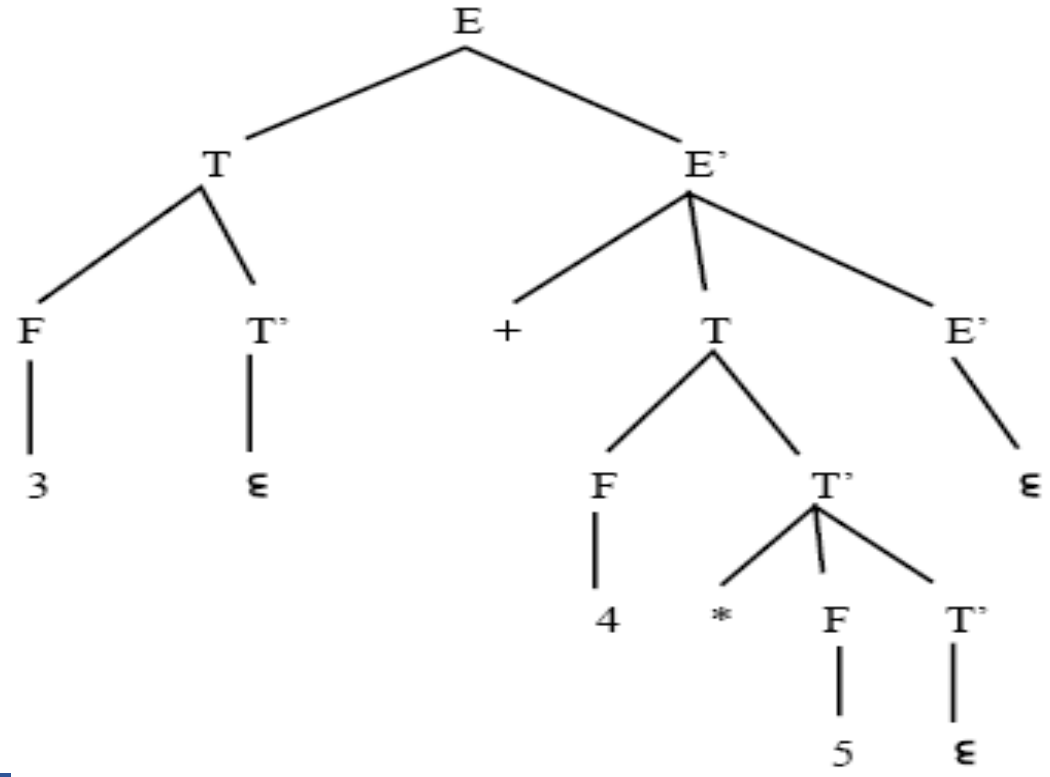
	nb	+	*	(	)	$
E	E→TE’			E→TE’		
E'		E’ →+TE’			E’ →ε	E’ →ε
T	T→FT’			T→FT’		
T'		T’ →ε	T’ →*FT’		T’ →ε	T’ →ε
F	F→nb			F→(E)		

PILE	Entrée	Sortie
$ *E*	3+4*5$	E→TE’
$ *E'T*	3+4*5$	T→FT’
$ *E'T ’ F*	3+4*5$	F→nb
$ *E'T* '3	3+4*5$	
$ *E'T* '	+4*5$	T’ →ε
$ *E'*	+4*5$	E’ →+TE’
$ *E'T* +	+4*5$	
$ *E'T*	4*5$	T→FT’
$ *E'T'F*	4*5$	F→nb
$ *E'T'*4	4*5$	
$ *E'T'*	*5$	T’ →*FT’
$ *E'T'F**	*5$	
$ *E'T'F*	5$	F→nb
$ *E'T'*5	5$	
$ *E'T'*	$	T’ →ε
$ *E'*	$	E’ →ε
$	$	Analyse syntaxique réussie

Analyseur syntaxique LL(1): exemple

On obtient donc l'arbre syntaxique : pour 3+4*5

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | nb



PILE	Entrée	Sortie
$ *E*	3+4*5$	E→TE'
$ *E'T*	3+4*5$	T→FT'
$ *E'T ' F*	3+4*5$	F→nb
$ *E'T* '3	3+4*5$	
$ *E'T* '	+4*5$	T' →ε
$ *E'*	+4*5$	E' →+TE'
$ *E'T* +	+4*5$	
$ *E'T*	4*5$	T→FT'
$ *E'T'F*	4*5$	F→nb
$ *E'T*'4	4*5$	
$ *E'T'*	*5$	T' →*FT'
$ *E'T'F**	*5$	
$ *E'T'F*	5$	F→nb
$ *E'T*'5	5$	
$ *E'T'*	$	T' →ε
$ *E'*	$	E' →ε
$	$	Analyse syntaxique réussie

Analyseur syntaxique LL(1): exemple

- Si l'on essaye d'analyser maintenant le mot :

  m=(7+3)5

	nb	+	*	(	)	$
E	E→TE'			E→TE'		
E'	Err	E' →+TE'			E' →ε	E' →ε
T	T→FT'			T→FT'		
T'	Err	T' →ε	T' →*FT'		T' →ε	T' →ε
F	F→nb			F→(E)		

PILE	Entrée	Sortie
$ *E*	(7+3)5$	E→TE'
$ *E'T*	(7+3)5$	T→FT'
$ *E'T'F*	(7+3)5$	F→(E)
$ *E'T'*) *E* (	(7+3)5$	
$ *E'T'*) *E*	7+3)5$	E→TE'
$ *E'T'*)*E'T*	7+3)5$	T→FT'
$ *E'T'*)*E'T'F*	7+3)5$	F→7
$ *E'T'*)*E'T'*7	7+3)5$	
$ *E'T'*)*E'T'*	+3)5$	T' →ε
$ *E'T'*)*E'*	+3)5$	E' → + T E'
$ *E'T'*)*E'T*+	+3)5$	
$ *E'T'*)*E'T*	3)5$	T→ FT'
$ *E'T'*)*E'T'F*	3)5$	F→ 3
$ *E'T'*)*E'T'*3	3)5$	
$ *E'T'*)*E'T'*	)5$	T' →ε
$ *E'T'*)*E'*	)5$	E' →ε
$ *E'T'*)	)5$	
$ *E'T'*	5$	ERREUR !!

RESUME : Analyse prédictive sans récursion

- L'analyseur syntaxique prédictif:
  - comporte un tampon d'entrée, initialisé avec la chaîne d'entrée suivie du symbole $;
  - comporte une pile, initialisée avec le symbole de départ par-dessus le symbole $;
  - utilise une table d'analyse M, où M[A, a] indique quelle production utiliser si le symbole sur le dessus de la pile est A et que le prochain symbole en entrée est a.
- Dans une configuration où X est sur le dessus de la pile et a est le prochain symbole dans l'entrée,
- L'analyseur effectue une des actions parmi les suivantes:
  - Si X = a = $, l'analyseur arrête ses opérations et annonce une analyse réussie.
  - Si X = a <> $, l'analyseur dépile X et fait avancer le pointeur de l'entrée.
  - Si X est un non-terminal, le programme de l'analyseur consulte la table d'analyse en position M[X, a]. La case consultée fournit soit une production à utiliser, soit une indication d'erreur.
  - Si, par exemple, M[X, a] = {X → UVW}, alors l'analyseur dépile X et empile W, V et U, dans l'ordre. En cas d'erreur, l'analyseur s'arrête et signale l'erreur.
- La sortie de l'analyseur consiste en la suite de productions utilisées.

Conclusion

- Si notre grammaire est LL(l) , l'analyse syntaxique peut se faire par l'analyse descendante vue précédemment . Mais comment savoir que notre grammaire est LL(1) ?
- Etant donnée une grammaire
  1. la rendre non ambigüe. Il n'y a pas de méthodes. Une grammaire ambigüe est une grammaire qui a été mal conçue.
  2. éliminer la récursivité à gauche si nécessaire
  3. la factoriser à gauche si nécessaire
  4. construire la table d'analyse. Il ne reste plus qu'a espérer que ça soit LL(1), Sinon il faut concevoir une autre méthode pour l'analyse syntaxique.