Chap 4: Analyse syntaxique

III- L'analyse syntaxique:

1- Le rôle d'un analyseur syntaxique

2- Grammaires non contextuelles

3- Ecriture d'une grammaire

4- Les méthodes d'analyse

5- L'analyse LL(1)

6- Constructeurs d'analyseurs syntaxiques

1- Rôle d'un analyseur syntaxique

1- Définition:

Un analyseur syntaxique reçoit une suite d'unités lexicales de l'analyseur lexical et vérifie que cette chaîne est engendrée par la grammaire du langage.

Le but d'un analyseur syntaxique est de trouver la relation entre les unités lexicales pour constituer des phrases et entre les phrases pour constituer des phrases plus grandes et produire un arbre syntaxique.

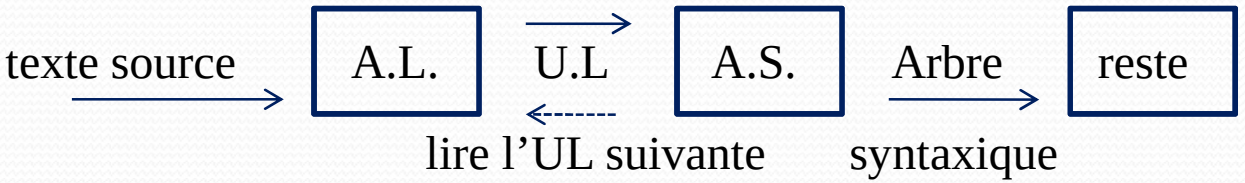
1- Rôle d'un analyseur syntaxique

Il y a 3 types généraux d'analyseurs syntaxiques:

1- méthodes universelles comme celles de *Cocke-Younger-Kasami* qui peuvent analyser une grammaire quelconque

2- méthodes ascendantes qui construisent des arbres d'analyse en partant des feuilles vers la racine.

3- méthodes descendantes qui construisent des arbres d'analyse en partant de la racine vers les feuilles



1- Rôle d'un analyseur syntaxique

2- Traitement des erreurs:

Les programmes peuvent contenir des erreurs à différentes phases de la compilation:

- **lexical**: comme l'écriture est erroné d'un identificateur,
- **syntaxique**: comme une expression mal parenthésée
- **sémantique**: comme un opérateur appliqué à un opérande incompatible

Souvent, la détection et la récupération des erreurs se fait principalement dans la phase d'analyse syntaxique.

1- Rôle d'un analyseur syntaxique

Les stratégies les plus utilisées sont:

1- récupération en mode panique: son implantation est simple, il s'agit d'ignorer les symboles d'entrée les unes après les autres jusqu'à rencontrer un symbole de synchronisation.

2- récupération au niveau du syntagme: correction locale

exemple: remplacer une virgule par un point-virgule

3- productions d'erreur: ajouter des productions à la grammaire qui tiennent compte des erreurs courantes.

4- correction globale: concevoir des algorithmes qui peuvent effectuer le minimum de changement sur une chaîne erronée pour la corriger.

2- Grammaires non contextuelles

Souvent les constructions des langages de programmation ont des *structures récursives*.

Les phrases ou les constructions d'un tel langage peuvent être définies par des grammaires (axiome, terminaux, non terminaux, productions).

Exemple: soit S1 et S2 deux instructions et E une expression.

et soit le texte d'entrée: « *si E alors S1 sinon S2* »

La production associée est:

instr ⟶ *si (exp) alors instr sinon instr*

2- Grammaires non contextuelles

1- Dérivation:

La définition d'un langage par une grammaire peut être vue comme un processus de construction de l'arbre syntaxique.

La dérivation est une description de la construction descendante de cet arbre.

On traite une production en remplaçant le non terminal de la partie gauche par la partie droite de la production.

Exemple: soit la grammaire des expressions arithmétiques

$$E \longrightarrow E + E \mid E * E \mid (E) \mid -E \mid id$$

$-E$ est aussi une expression

E *se dérive en* $-E$ en une étape: $E \Longrightarrow -E$

Une dérivation de E en $-(id)$:

$$E \Longrightarrow -E \Longrightarrow -(E) \Longrightarrow -(id)$$

E *se dérive en* $-(id)$ en 3 étapes.

2- Grammaires non contextuelles

D'une manière générale:

si A $\longrightarrow$ γ alors α A β $\Longrightarrow$ α γ β

avec α, β ,γ des suites de symboles grammaticaux

Notation:

$\overset{*}{\Longrightarrow}$ se dérive en **zéro ou plusieurs étapes**

$\overset{+}{\Longrightarrow}$ se dérive en **une ou plusieurs étapes**

*Soit **G** une grammaire, **S** son axiome et **L(G)** le langage engendré par **G**.*

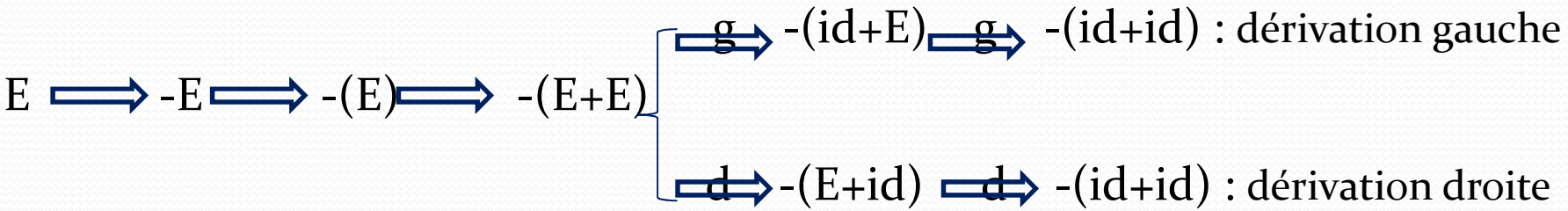
*Une chaîne de terminaux **ω ∈ L(G)** si et seulement si* $S \overset{+}{\Longrightarrow} \omega$

La chaîne **ω** est appelée **phrase** de **L(G)**.

Si $S \overset{*}{\Longrightarrow} \alpha$, avec **α** pouvant contenir certains non terminaux, α est appelée **proto-phrase**.

2- Grammaires non contextuelles

Question : Montrer que la chaîne « -(id+id) » est une phrase engendrée par la grammaire précédente ?



Notation: $\alpha \overset{g}{\Rightarrow} \beta$;

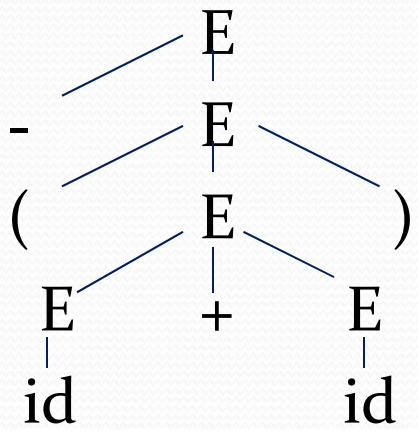
α se dérive en β en remplaçant le non terminal de α ***le plus à gauche***

2- Grammaires non contextuelles

2- Arbre syntaxique et dérivation :

Un arbre syntaxique est une représentation graphique d'une dérivation dans laquelle l'ordre de remplacement a disparu.

Exemple: soit la phrase -(id+id)

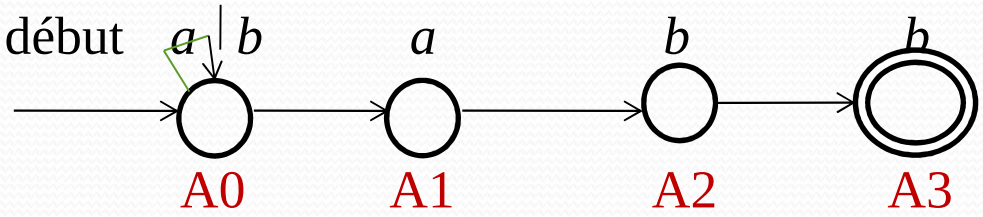


3- Ecriture d'une grammaire

Chaque construction qui peut être écrite par une expression régulière peut également être décrite par une grammaire dite régulière.

exemple: soit l'expression régulière (a | b)* abb

l'AFN correspondant est:



La grammaire associée est:

A0 ⟶ a A0 | b A0 | a A1

A1 ⟶ b A2

A2 ⟶ b A3

A3 ⟶ ε

3- Ecriture d'une grammaire

Construction d'une grammaire régulière:

- On nomme chaque état de l'automate,

- A chaque état de l'automate correspond un non terminal de la grammaire.

- L'état initial de l'automate correspond à l'axiome de la grammaire.

- A chaque transition correspond une production de la grammaire.

- A l'état d'acceptation correspond une production d'une chaîne vide.

3- Ecriture d'une grammaire

1- Suppression de l'ambiguïté:

Une grammaire est dite *ambigüe* si elle produit plus d'un arbre syntaxique pour la même phrase.

exemple:

Soit la grammaire des instructions conditionnelles à la PASCAL:

instr → si exp alors instr
| si exp alors instr sinon instr
| autre

exp → Ei

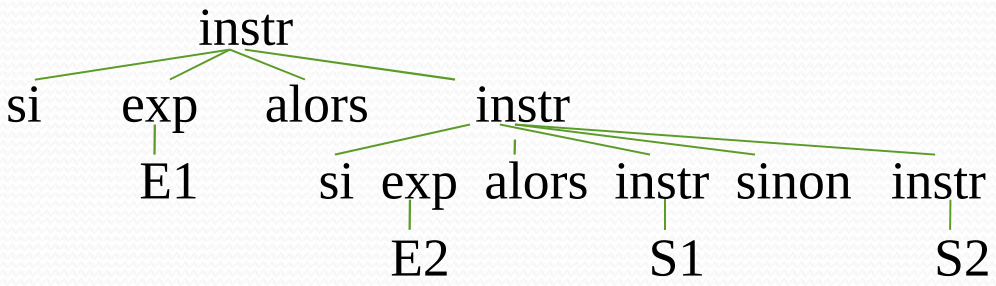
Et soit l'instruction: « ***si E1 alors si E2 alors S1 sinon S2*** »

3- Ecriture d'une grammaire

Nous avons, d'après cette grammaire, deux interprétations possibles pour la même phrase :

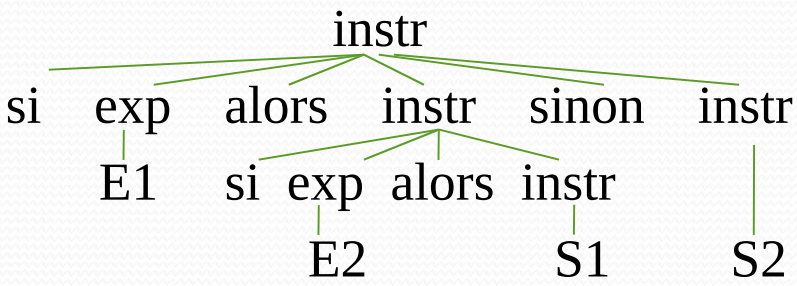
1- si E1 alors si E2 alors S1 sinon S2

l'arbre d'analyse correspondant:



2- si E1 alors si E2 alors S1 sinon S2

l'arbre d'analyse correspondant:



Cette grammaire est ***ambigüe***.

3- Ecriture d'une grammaire

Dans les langages de programmation ayant des instructions conditionnelles, on préfère le 1er arbre.

*Par convention on fait correspondre chaque **sinon** avec le **si** qui le précède, le plus proche et sans correspondant.*

L'idée est qu'une instruction apparaissant entre un ***alors*** et un ***sinon*** doit être *'close'*, c-à-d qu'elle ne doit pas se terminer par un ***alors*** sans correspondant.

Une instruction close est soit une instruction si-alors-sinon ne contenant pas d'instruction non close, soit une instruction non conditionnelle quelconque.

La réécriture de cette grammaire donne :

3- Ecriture d'une grammaire

instr ⟶ instr-close
| instr-non-close

instr-close ⟶ si exp alors instr-close sinon instr-close
| autre

instr-non-close ⟶ si exp alors instr
| si exp alors instr-close sinon instr-non-close

A tester pour l'instruction:

« si E1 alors si E2 alors S1 sinon si E3 alors S2 sinon S3 »

Remarque:

Pour éviter ce type d'ambigüité, certains langages introduisent des ***fin-de-si*** :

- ***fi*** pour Algol 68, Eiffel
- ***end*** pour Modula
- ***endif*** pour ADA
- ***done*** pour Newton

3- Ecriture d'une grammaire

2- Suppression de la récursivité gauche:

Une grammaire est dite récursive à gauche si elle contient un non terminal A tel qu'il existe une dérivation:

$A \Longrightarrow^{+} A\,\alpha$ avec α une chaîne quelconque de symboles grammaticaux (une suite de terminaux et de non terminaux).

Soit la grammaire: $A \longrightarrow A\,\alpha_1 \mid A\,\alpha_2 \mid \ldots \mid A\,\alpha_m \mid \beta_1 \mid \ldots\ldots \mid \beta_n$

Pour supprimer la récursivité gauche, nous réécrivons cette grammaire sous la forme:

$$A \longrightarrow \beta_1\,A' \mid \beta_2\,A' \mid \ldots\ldots \mid \beta_n\,A'$$
$$A' \longrightarrow \alpha_1\,A' \mid \alpha_2\,A' \mid \ldots\ldots \mid \alpha_m\,A' \mid \varepsilon$$

avec les βi ne commençant pas par A,

et les α_i ne commençant pas par A' et ne produisent pas la chaîne vide

3- Ecriture d'une grammaire

Exemple : soit la grammaire récursive à gauche:

$E \longrightarrow E + T \mid T$ $\quad\quad \alpha 1 = + T,\ \beta = T$

$T \longrightarrow T * F \mid F$ $\quad\quad \alpha 1 = * F,\ \beta = F$

$F \longrightarrow id \mid (E)$

La grammaire, non récursive à gauche, équivalente est:

$E \longrightarrow T\ E'$

$E' \longrightarrow + T\ E' \mid \varepsilon$

$T \longrightarrow F\ T'$

$T' \longrightarrow * F\ T' \mid \varepsilon$

$F \longrightarrow id \mid (E)$

3- Ecriture d'une grammaire

Remarque: cas d'une récursivité cachée

$$A \Longrightarrow \alpha \overset{+}{\Longrightarrow} A\,\beta$$

Exemple:

$S \longrightarrow A\,a \mid b$

$A \longrightarrow A\,c \mid S\,d \mid \varepsilon$

le S est récursif à gauche.

3- Ecriture d'une grammaire

3- La factorisation à gauche:

Si les parties droites de plusieurs productions commencent par le même préfixe, lors de l'analyse il n'est pas évident de choisir la 'bonne' production.

($\Longleftrightarrow$ AFN dans l'analyse lexicale)

Soit les S-productions

$S \longrightarrow A\,\beta1 \mid A\,\beta2 \mid \ldots\ldots \mid A\,\beta n$

avec A le préfixe commun.

En factorisant à gauche, on a:

$S \longrightarrow A\,A'$

$A' \longrightarrow \beta1 \mid \beta2 \mid \ldots\ldots \mid \beta n$

3- Ecriture d'une grammaire

Exemple: soit les S-règles

S ⟶ A B C D

S ⟶ A B a D

S ⟶ A b a

factorisées, elles deviennent:

S ⟶ A E

E ⟶ B F | b a

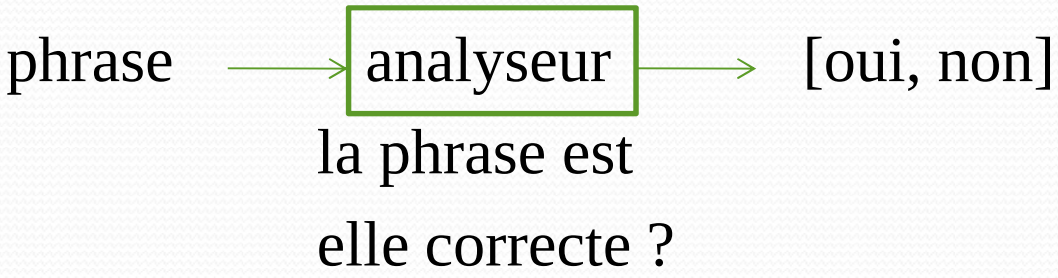
F ⟶ C D | a D

4- Les méthodes d'analyse

Les différentes questions que l'on pose par rapport aux phrases engendrées par une grammaire sont:

Supposons qu'on a une phrase ω :

- comment analyser ω ?
- ω appartient-elle au langage L(G) ?
- si ω ϵ L(G), donner une dérivation.



4- Les méthodes d'analyse

Il y'a 2 critères de classification pour distinguer les différentes méthodes d'analyse syntaxique:

- le sens de parcours de la chaîne analysée: gauche-droite ou droite-gauche.

- le sens d'application des productions: dérivation ou réduction.

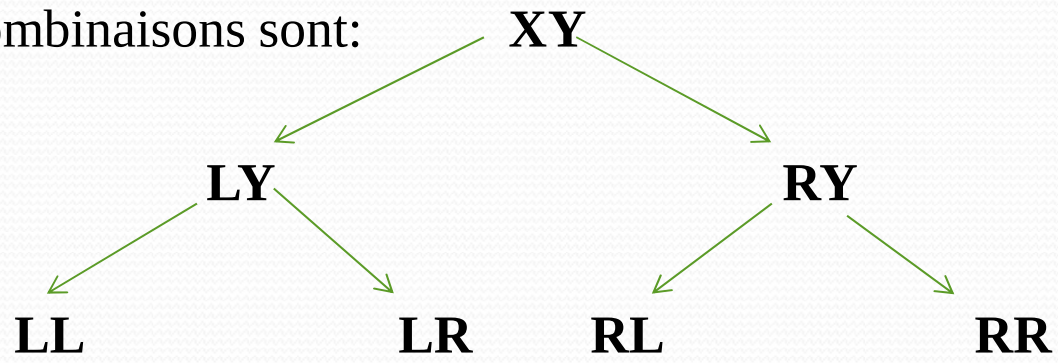
on distingue 4 méthodes d'analyse désignées par un couple **XY**.

X- le sens de parcours , **Y**- le sens d'application des productions

X = **L**,**R** **L**: Left to Right **R**: Right to Left

Y = **L**,**R** **L**: Leftmost derivation **R**: Rightmost derivation

Les 4 combinaisons sont:



5- L’analyse LL(1)

C’est une analyse avec une inspection d’**un** symbole terminal en avance sur le symbole courant traité.

Dans la pratique ce sont les grammaires les plus utilisées.

Le but de l’analyse LL(1) est de pouvoir (toujours) connaître la règle de production à appliquer et ceci en se basant sur le terminal suivant non encore traité de la phrase en cours d’examen.

Deux concepts nouveaux se révèlent nécessaires pour faciliter l'analyse LL(1):

- PREMIER (FIRST)
- SUIVANT (FELLOW)

5- L'analyse LL(1)

Définition:

Une grammaire est LL(1) si elle remplit la condition suivante:

Quel que soit ***A*** *un non terminal tel que:*

$A \longrightarrow \alpha$ *et* $A \longrightarrow \beta$ *avec* α *et* β *2 symboles grammaticaux, on a:*

$$S \underset{g}{\overset{*}{\Longrightarrow}} \omega A \gamma \underset{g}{\Longrightarrow} \begin{cases} \omega \alpha \gamma \overset{*}{\Longrightarrow} \omega x \\ \omega \beta \gamma \overset{*}{\Longrightarrow} \omega y \end{cases}$$

si ***PREMIER****(x) =* ***PREMIER****(y) alors* $\alpha = \beta$

si ***PREMIER****(x)* $\neq$ ***PREMIER****(y) alors* $\alpha \neq \beta$

5- L'analyse LL(1)

PREMIER représente l'ensemble des premiers symboles terminaux qui commencent **x** et **x** $\neq \epsilon$.

c-à-d qu'à un instant donné de la dérivation; s'il y a plusieurs chemins pour arriver à la phrase que l'on analyse, ces chemins sont identiques et la connaissance du prochain symbole de la phrase est suffisant pour déterminer la règle de production à utiliser.

5- L'analyse LL(1)

1- Les ensembles ***PREMIER*** et ***SUIVANT***:

Les notions PREMIER et SUIVANT sont utiles d'une part pour montrer qu'une grammaire est LL(1) et d'autre part pour la construction de l'analyseur syntaxique.

1.1- L'ensemble ***PREMIER***:

- si *a* est un terminal, alors: ***PREMIER***(*a*) = {*a*}

  de même: ***PREMIER***(ε) = {ε}

- si A ⟶ *b* α , alors *b* est un élément de ***PREMIER***(A)

- si A ⟶ B α , alors ***PREMIER***(B) C ***PREMIER***(A)

  si B peut produire la chaîne vide, alors

  ((***PREMIER***(B)\{ε}) U ***PREMIER***(α)) C ***PREMIER***(A)

D'une autre manière, PREMIER(α) inclut toujours l'ensemble PREMIER du premier symbole de α.

Si celui-ci peut produire ε, il inclut aussi l'ensemble PREMIER du second symbole de α, etc…

Si tous les symboles de α peuvent produire ϵ alors PREMIER(α) contient ε.

5- L'analyse LL(1)

1.2- L'ensemble ***SUIVANT***:

Nous appliquons les règles suivantes jusqu'à ce qu'aucun terminal ne puisse être ajouté aux ensembles ***SUIVANT***.

- mettre $ dans ***SUIVANT*** de l'axiome.

- s'il y a une production : $A \longrightarrow \alpha B \beta$,

le contenu de ***PREMIER***(β), excepté la chaîne vide est ajouté à ***SUIVANT***(B)

- s'il y a une production : $A \longrightarrow \alpha B$ ou $A \longrightarrow \alpha B \beta$ *tq* $\beta \overset{*}{\Longrightarrow} \varepsilon$

les éléments de ***SUIVANT***(A) sont ajoutés à ***SUIVANT***(B)

5- L'analyse LL(1)

Exemple:

Soit la grammaire:

$E \longrightarrow E + T \mid E - T \mid T$

$T \longrightarrow T * F \mid T / F \mid F$

$F \longrightarrow id \mid (E)$

PREMIER(F) = {id,(}

PREMIER(T) = {id,(}

PREMIER(E) = {id,(}

SUIVANT(E) = {$, + , - ,)}

SUIVANT(T) = {$, + , -, *, /,)}

SUIVANT(F) = {$, + , - , * , / ,)}

La grammaire équivalente:

$E \longrightarrow T\ E'$

$E' \longrightarrow + T\ E' \mid - T\ E' \mid \varepsilon$

$T \longrightarrow F\ T'$

$T' \longrightarrow * F\ T' \mid / F\ T' \mid \varepsilon$

$F \longrightarrow id \mid (E)$

PREMIER(F) = {id,(}

PREMIER(T) = {id,(}

PREMIER(T') = {*, / , ε}

PREMIER(E) = {id,(}

PREMIER(E') = {+, - , ε}

SUIVANT(E) = {$,)}

SUIVANT(E')= {$,)}

SUIVANT(T) = {$, +, - ,)}

SUIVANT(T')= {$,),+,-}

SUIVANT(F) = {$, + ,- , *,/ ,)}

5- L'analyse LL(1)

2- Les conditions LL(1):

Pour qu'une grammaire soit de type LL(1), c'est-à-dire qu'elle soit analysable par une ***méthode descendante déterministe****, il faut que ses règles de production respectent 2 conditions:*

soit une paire de règles: $A \longrightarrow \alpha$ et $A \longrightarrow \beta$ on a:

1ère condition:

$\boldsymbol{PREMIER}(\alpha) \cap \boldsymbol{PREMIER}(\beta) = \emptyset$

exemple: $A \longrightarrow a\ B \mid a\ C$

2ème condition:

si $\varepsilon \in \boldsymbol{PREMIER}(\beta)$ alors $\boldsymbol{PREMIER}(\alpha) \cap \boldsymbol{SUIVANT}(A) = \emptyset$

exemple: sinon en suspens

[G est LL(1) $\Leftrightarrow$ PREMIER(αSUIVANT(A)) $\cap$ PREMIER(βSUIVANT(A)) =$\emptyset$]

5- L'analyse LL(1)

Exemple 1:

Considérons la grammaire des expressions arithmétiques,

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow (E) \mid nb$

5- L'analyse LL(1)

Exemple 2: Soit la grammaire suivante

$S \rightarrow Xa$

$S \rightarrow \varepsilon$

$X \rightarrow Xb$

$X \rightarrow Y$

$Y \rightarrow aX$

$Y \rightarrow \varepsilon$

5- L’analyse LL(1)

Exemple 3:

Montrer qu'une grammaire LL(1) ne peut pas avoir des productions récursives à gauche, c'est à dire:

$$A \rightarrow A\,\alpha \mid \beta$$

5- L'analyse LL(1)

3- Construction d'un analyseur LL(1):

L'intérêt de cet analyseur est de reconnaître et de fournir ***l'unique dérivation gauche*** des phrases d'un langage.

Un analyseur LL(1) est implanté sous forme d'un automate à pile.

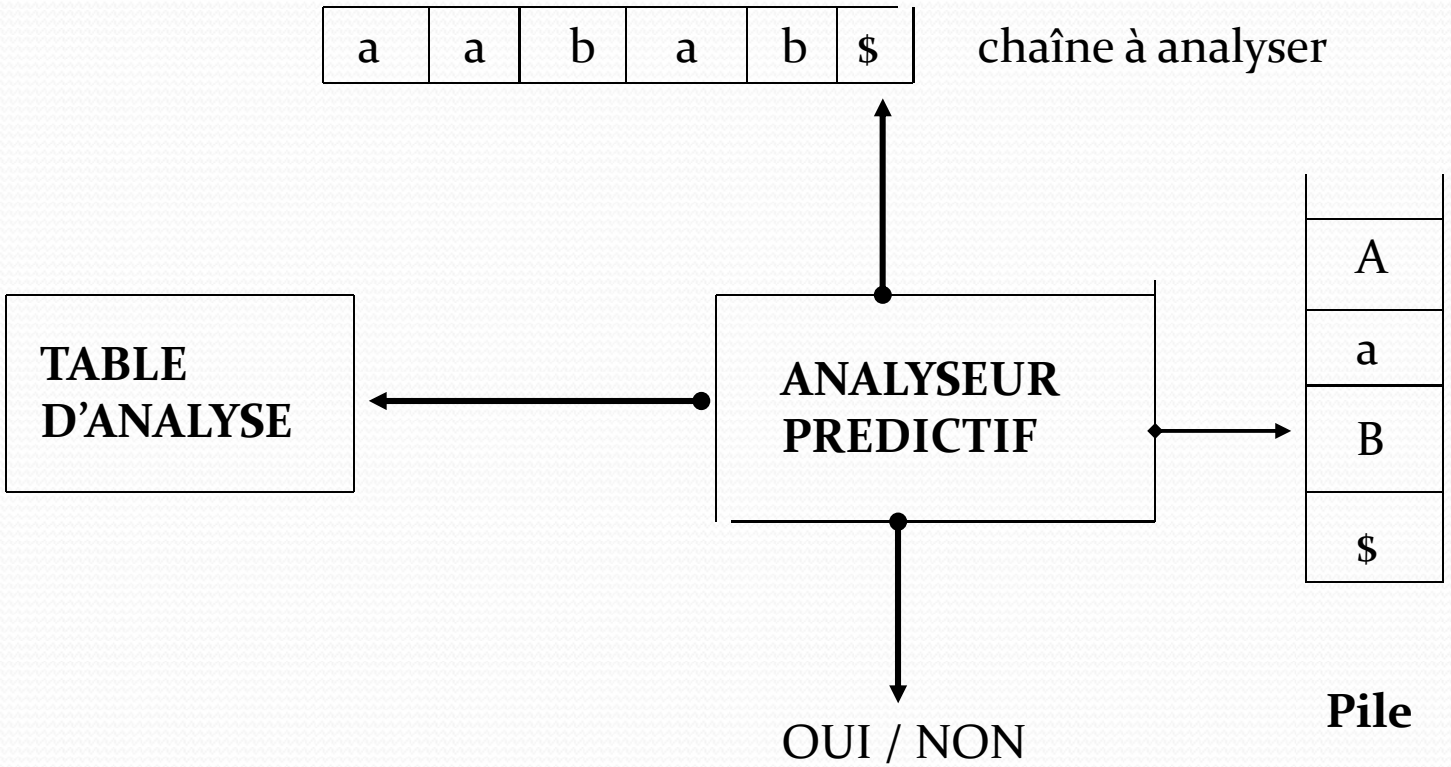
Il est constitué de:

- un ***tampon*** qui contient la chaîne du programme à analyser, suivie par le symbole **$**.

- une ***pile*** qui contient une séquence de symboles grammaticaux, avec **$** au fond de la pile.

- une ***table d'analyse*** à deux dimensions qui permet de choisir la règle de production à utiliser.

5- L’analyse LL(1)



5- L'analyse LL(1)

3.1- Table d'analyse:

Elle est constituée de l'ensemble des règles à utiliser, DEPILER, ERREUR, ACCEPTE.

Donnée: une grammaire.

Résultat: une table d'analyse **M** pour **G**.

Méthode:

M[A,a]= { - A $\longrightarrow$ α si a $\in$ ***PREMIER***(α),

- A $\longrightarrow$ α si α $\overset{*}{\Longrightarrow}$ ϵ et a $\in$ ***SUIVANT***(A),

- ERREUR }

M[a,b]= { - DEPILER si a=b,

- ACCEPTE si a=b=$,

- ERREUR }

Algorithme de construction de la table d'analyse syntaxique

Début
  M ← ∅;
    pour chaque règle A → α faire
      pour chaque a ∈ Premiers(α) faire
        M[A,a] ← M[A,a] ∪ {A → α }
      fait;
      si ε ∈ Premiers(α) alors
        pour chaque b ∈ Suivants(A) faire
          M[A,b] ← M[A,b] ∪ {A → α }
        fait;
      fsi;
    fait;
Fin.

5- L'analyse LL(1)

3.2 Exemple 1:

Grammaire symbolisant if-then-else après factorisation (elle est ambiguë):

S → i E t S S' | a

S' → e S | ε

E → b

	S	S'	E
PREMIER	i , a	e , ϵ	b
SUIVANT	$, e	$, e	t

	a	b	e	i	t	$
S	S→a	erreur	erreur	S→iEtSS'	erreur	erreur
S'	erreur	erreur	S'→eS S'→ε	erreur	erreur	S'→ε
E	erreur	E →b	erreur	erreur	erreur	erreur

5- L'analyse LL(1)

Exemple 2 : soit la grammaire ,

$E \rightarrow T\ E'$ $\qquad$ $E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$ $\qquad$ $T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid nb$

on associe la table d'analyse suivante:

	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)		

5- L'analyse LL(1)

La trace d'analyse:

Pour analyser une chaîne, on initialise l'automate à pile, en empilant $ puis l'axiome.

Ensuite, on combine le s symboles du texte d'entrée, les uns après les autres, avec le symbole courant au sommet de la pile.

Lorsque cette combinaison nous donne, à partir de la table d'analyse, une règle de production à appliquer, on remplace le symbole au sommet de la pile par la partie droite de la production.

	pile	tampon
Etat initial	S$	ω$
Etat final	$	$

La trace d'analyse, nous donne l'arbre d'analyse, et le processus de l'unique dérivation gauche de la phrase analysée.

3.3- La trace d'analyse:

Soit la phrase à analyser: «nb + nb * nb »

Pile	Tampon	Action
E$	nb+nb*nb$	E→TE'
TE'$	nb+nb*nb$	T→FT'
FT'E'$	nb+nb*nb$	F→nb
nbT'E'$	nb+nb*nb$	DEPILER
T'E'$	+nb*nb$	T'→ε
E'$	+nb*nb$	E'→+TE'
+TE'$	+nb*nb$	DEPILER
TE'$	nb*nb$	T→FT'
FT'E'$	nb*nb$	F→nb
nbT'E'$	nb*nb$	DEPILER
T'E'$	*nb$	T'→*FT'
*FT'E'$	*nb$	DEPILER
FT'E'$	nb$	F→nb
nbT'E'$	nb$	DEPILER
T'E'$	$	T'→ε
E'$	$	E'→ε
$	$	ACCEPTE