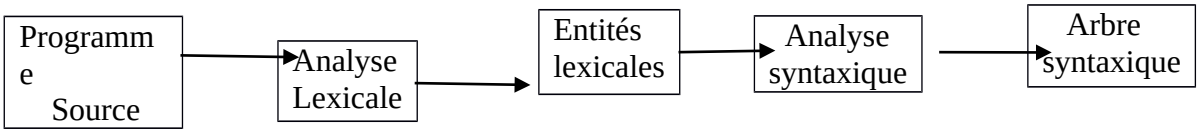
Chapitre II :

L’analyse syntaxique



L’analyse lexicale, a pour but, à partir du programme source suite de caractères, de former des entités lexicales. L’analyse syntaxique, a pour rôle la vérification de la forme ou encore de l’écriture de la suite des entités lexicales, on parlera alors de la syntaxe du langage. La vérification est faite à l’aide de la grammaire spécifique au langage, appelée grammaire syntaxique.

Remarque :

l’analyse syntaxique est l’étape la mieux formalisée de la compilation (grammaire).

II.1 Définitions :

II.1.1 Définition d’une grammaire syntaxique :

Une grammaire syntaxique est une grammaire formée de règles de production permettant d’engendrer tous les programmes écrits dans un langage donné.

Exemple :

Soit la grammaire des expressions arithmétiques :

<program> ⟶ <exp>#

<exp> ⟶ <Term> / <exp>+<Term>

<Term> ⟶ <Fact> / <Term>*<Fact>

<Fact> ⟶ i / (<exp>)

Remarque :

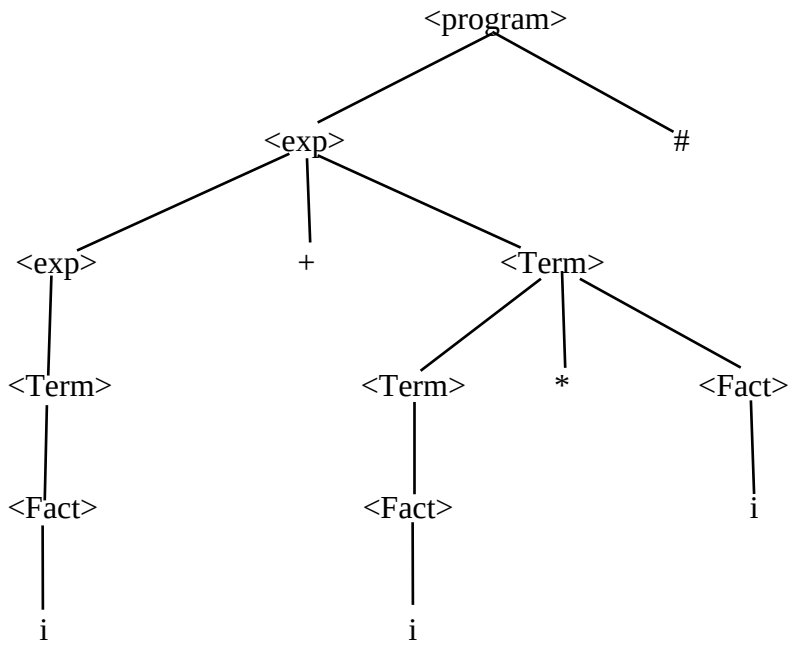
- Chacune des règles précédentes est appelée règle de production, la partie gauche est appelée MGP (membre gauche de production) et la partie droite MDP (membre droit de production).
- L’entité entre < > est appelée non-terminal.

II.1.2 Notion de dérivation :

Les opérations consistent à remplacer au fur et à mesure en partant de l’axiome des non-terminaux par leurs MDPs sont appelées dérivations.

Exemple :

Soit a analyser la chaine i+i*i#



II.1.3 Notion de réduction :

L'opération de réduction est l'inverse de la dérivation, elle consiste à remplacer, une fois que l'on reconnait un MPD, par la partie gauche MGP, ainsi jusqu' à arriver à l'axiome.

II.1.4 Notion d'arbre syntaxique :

Un arbre syntaxique est un arbre, dont la racine est l'axiome de la grammaire syntaxique, les nœuds représentent les MDPs et les feuilles terminales, la chaine à analyser syntaxiques.

Deux méthodes sont utilisées pour la construction d'un arbre syntaxique :

1- A partir de l'axiome, on effectue une série de dérivations, jusqu'à atteindre la chaine à analyser, c'est le cas descendant.
2- A partir de la chaine à analyser, on effectue des réductions, jusqu'à arriver à l'axiome, c'est le cas ascendant.

Remarque :

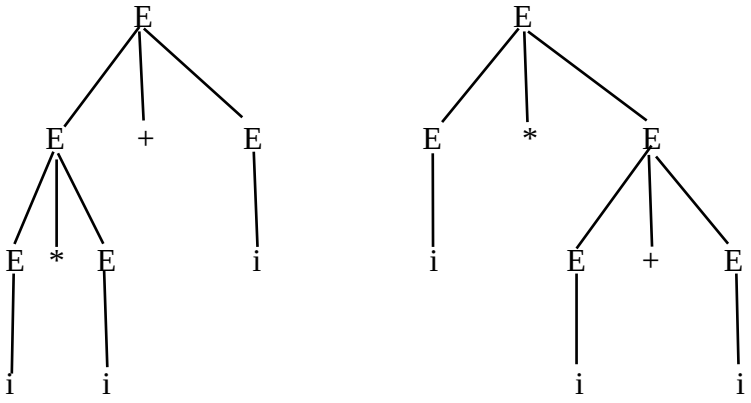
Un arbre syntaxique qui a deux fils est appelé arbre binaire.

II.1.5 Notion d'ambiguïté :

On dit qu'une grammaire est ambiguë, s'il existe au moins deux arbres syntaxiques correspondant à une même chaine à analyser.

Exemple :

E⟶E+E / E*E / i et i*i+i# la chaine à analyser.



⟹ Deux arbres syntaxiques ⟹ Grammaire ambigüe.

Remarque :

Dans notre cas, nous nous intéressons qu'aux grammaires non ambiguës.

II.2 Analyse ascendante et Analyse descendante :

Il existe deux types de méthodes d'analyse syntaxique :

- Les méthodes descendantes, qui consistent à partir de l'axiome, à dériver, jusqu'à l'obtention de la chaine à analyser, en commençant par le non-terminal plus à gauche, et l'analyse se fait généralement de gauche à droite.
- Les méthodes ascendantes, contrairement, consistent à faire des réductions, jusqu'à aboutir à l'axiome.

II.2.1 Récursivité gauche directe et indirecte :

- Une grammaire est dite récursive gauche, de façon directe, s'elle s'écrit de la façon suivante :

  A ⟶ Aα / β avec A€N, α et β € (TUN)*
- Une grammaire est récursive indirectement, si elle s'écrit :

  A ⟶ Bα1 /β1

  B ⟶ Aα2 /β2 avec A, B € N et α1, α2, β1,β2 € (TUN)*

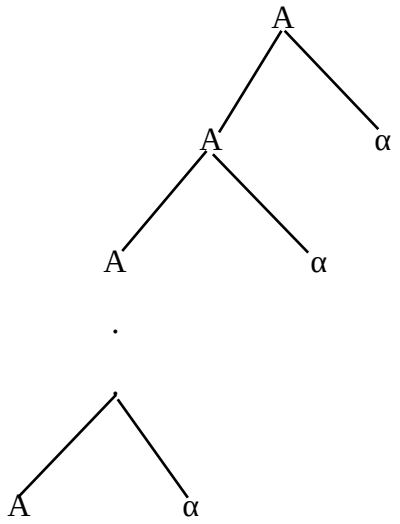
Remarque :

Si une grammaire posséde une règle de la forme : A ⟶ αA, on parlera alors de récursivité droite.

Transformation d'une grammaire récursive en une grammaire non récursive :

Un analyse descendante ne peut se faire sur une grammaire récursive gauche directement ou indirectement.

Si dans la grammaire, il existe une production du type A ⟶ Aα, l'identification de la chaine à analyser avec un des MDPs sera bloquée (boucle infinie !!).



Pour cela une transformation de la grammaire sera nécessaire :

$A \longrightarrow A\alpha 1 / A\alpha 2 / \ldots\ldots\ldots../A\alpha n / \beta 1 / \beta 2 / \ldots\ldots../\beta m$

$\Rightarrow \begin{cases} A \longrightarrow \beta 1 / \beta 2 / \ldots\ldots\ldots../\beta m / \beta 1A' / \beta 2A' / \ldots\ldots..../\beta mA' \\ A' \longrightarrow \alpha 1 / \alpha 2 / \ldots\ldots\ldots\ldots/\alpha n / \alpha 1A' / \alpha 2A' / \ldots\ldots\ldots\ldots/\alpha nA' \end{cases}$

Exemple :

$\begin{cases} S \longrightarrow aS/ b/ Ac \\ A \longrightarrow Aa/ c \end{cases} \Rightarrow \begin{cases} S \longrightarrow aS/ b/ Ac \\ A \longrightarrow c/ cA' \\ A' \longrightarrow a/ aA' \end{cases}$

Cas de la récursivité gauche indirecte :

Pour éliminer la récursivité gauche indirecte, on ordonne les non-terminaux de la manière suivante :

Si $A \longrightarrow B\alpha$ Alors $A<B$

Ensuite s'il existe un non-terminal A tel que $A<A1<\ldots\ldots\ldots<An<A$ alors il fait faire des substitutions de telle manière à faire apparaitre une récursivité gauche directe et appliquer les règles ci-dessus.

Exemple :

$\begin{cases} S \longrightarrow Ac/ b \\ A \longrightarrow Bd/ a \\ B \longrightarrow Ac/ d \end{cases}$ On $A<B<A \Rightarrow$ une récursivité indirecte dans A

Donc :

- Substitution de B dans A :

S ⟶ Ac/ b

A ⟶ Acd/ dd/ a

B ⟶ Ac/ d

- Elimination de la récursivité gauche directe dans A :

S ⟶ Ac/ b

A ⟶ dd/ a/ ddA'/ aA'

A' ⟶ cd/ cdA'

B ⟶ Ac/ d

II.2.2 Factorisation d'une grammaire :

Une des conditions permettant de faire une analyse syntaxique descendante déterministe est la factorisation.

Une grammaire G=<T, N, S, P> n'est pas factorisée, s'il existe dans P des productions de la forme :

$A \longrightarrow \alpha X/ \alpha Y/ \alpha Z/ \beta/\mu$

Avec β, μ, X, Y, Z € (TUN)*

et α € (TUN)+

Devient :

A ⟶ αB/ β/ μ

B ⟶ X/ Y/ Z

II.3 Les méthodes d'analyse descendantes :

Il existe deux types de méthodes d'analyse syntaxiques :

1- Les méthodes non-déterministes :
   . Avec retour arrière
   . Descendante parallèle
2- Les méthodes déterministes.

Dans notre cours, nous nous intéressons qu'aux méthodes déterministes.

Les méthodes d'analyse déterministes :

Ces méthodes consistent à partir de l'axiome, à dériver, jusqu'à l'obtention de la chaine à analyser, en commençant par le non-terminal le plus à gauche, et l'analyse se fait généralement de gauche à droite.

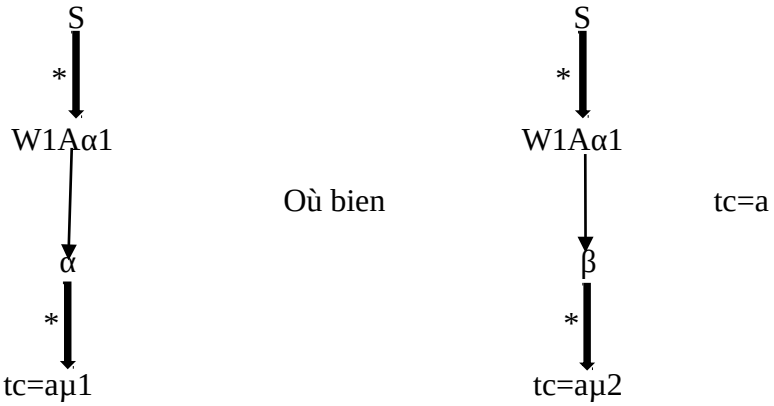
Ces méthodes sont sans retour arrière, seule l'alternative susceptible de correspondre est choisie.

II.3.1 Les grammaires LL(1) :

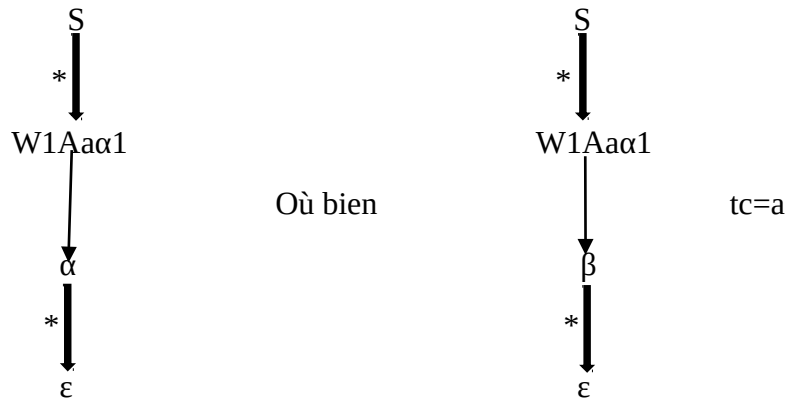
Definition1 :

Une grammaire LL(1) si et seulement si pour chaque couple de production A ⟶ α/ β avec α, β € (TUN)* on a :

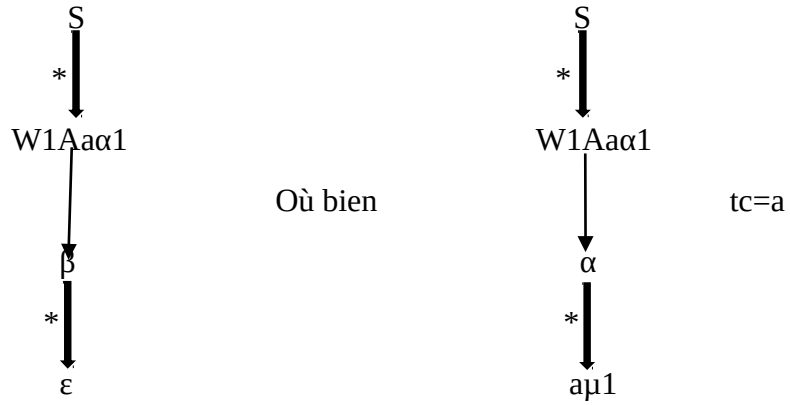
1- Si α ⇒ aμ1 Alors β ⇏ aμ2 a € T et μ1, μ2 € (TUN)* Fsi.



2- Si α ⇒ ε Alors β ⇏ ε Fsi.



3- Si β ⇒ ε et α ⇏ aμ Alors λ ∈ Si ⇒ WAaα1 Fsi.



Terminologie :

Si $\alpha \overset{*}{\Rightarrow} a\mu 1$ Alors a € T est un symbole directeur de α

Ensemble Debut :

Debut(α) ={a € T / $\alpha \overset{*}{\Rightarrow} a\mu$, μ € (TUN)+} U {t € {ε}UT / $\alpha \overset{*}{\Rightarrow} t$} avec α € (TUN)*.

Etapes de construction de l'ensemble Debut :

- Si A $\longrightarrow$ α1/ α2/ ……../ αn avec A€N et αi€(TUN)*
  Alors Debut(α) = Debut(α1)U……..UDebut(αn)
  Fsi.
- Debut(a) ={a} avec a€T.
- Debut(aμ) = {a}avec a€T et μ€(TUN)+.
- Debut(ε) = {ε}.
- Debut(BX1……Xn) Ↄ Debut(B)-{ε} avec B€N et Xi€(TUN)+
  Si ε€Debut(B) Alors Debut(BX1……Xn) Ↄ Debut(X1)-{ε}
  Si ε€Debut(X1) Alors Debut(BX1….Xn) Ↄ Debut(X2)-{ε}
  .
  .
  .
  .
  Si ε€Debut(Xn-1) Alors Debut(BX1….Xn) Ↄ Debut(Xn)-{ε}
  Si ε€Debut(Xn)
  Alors ε€Debut(BX1…..Xn)
  Fsi ;
  Fsi ;
  Fsi ;
  Fsi.

Exemple :

S $\longrightarrow$ ABC / aSc
A $\longrightarrow$ ε / aAB
B $\longrightarrow$ bB / ε
C $\longrightarrow$ Bc / aC /ε

	Debut
S	a b c ε
A	a ε
B	b ε
C	a b c ε

Ensemble Suivant :

Suivant(A) = {a € TU{#} / Z $\overset{*}{\Rightarrow}$ WAα#, avec a € Debut(α#) et w, α € (TUN)*}.

Etape de construction de l'ensemble Suivant :

- Si ∃ MDP αAaβ Alors a € Suivant(A) avec A € N, α, β € (TUN)* et a € T.
- Si ∃ MDP αABβ Alors Debut(Bβ)-{ε} C Suivant(A) avec A,B € N et α, β € (TUN)*.
- Si ∃ une production B $\longrightarrow$ X1…..Xn A avec A, B € N et Xi € (TUN)

Alors Suivant(B) C Suivant(A)
Si A $\overset{*}{\Rightarrow}$ ε
Alors Suivant(B) C Suivant(Xn)
Si Xn $\overset{*}{\Rightarrow}$ ε
Alors Suivant(B) C Suivant(Xn-1)
.
.
.
Si X2 $\overset{*}{\Rightarrow}$ ε
Alors Suivant(B) C Suivant(X1)
Fsi ;
Fsi ;
Fsi ;
Fsi.

Exemple :

	Suivant
S	c #
A	a b c #
B	a b c #
C	c #

II.3.2 Analyse par les procédures récursives (Descende récursive) :

Elle consiste à associer à chaque non-terminal une procédure qui traite tous les MDPs du non-terminal.

A $\longrightarrow$ X1…Xn avec Xi € (TUN)

Si Xi € T Alors comparer Xi avec tc

Si Xi≠tc Alors Erreur

Sinon tc := tc+1 ;

Passer au traitement de Xi+1 ;

Fsi ;

Sinon Appel de la procédure associer à Xi ;

Passer au traitement ;

Fsi

L'analyse se fait à l'aide d'appels des procédures.

Exemple :

Z ⟶ S#

S ⟶ aB / Aa

A ⟶ BA / b

B ⟶ c

Procédure Z( )

Debut

Lire(entité) ;

tc := 1ere terme de la chaine ; erreur := faux ;

Si tc € Debut(S) Alors S( ) ;

Si tc ≠ # Alors erreur := vrai Fsi ;

Sinon erreur := vrai

Fsi;

Si erreur Alors 'chaine incorrecte'

Sinon 'Chaine correcte'

Fsi ;

FinZ

```
Procédure S( )
Debut
    Si tc € Debut(aB) Alors tc := tc+1 ;
                                Si tc € Debut(B) Alors B( )
                                                    Sinon erreur := vrai
                                Fsi ;
                            Sinon A( ) ;
                                  Si tc ≠ a Alors erreur := vrai
                                            Sinon Si non erreur Alors tc := tc+1 Fsi
                                  Fsi ;
     Fsi ;
FinS
Procédure A( )
Debut
     Si tc € Debut(BA) Alors B( ) ;
                                Si non erreur et tc € Debut(A)
                                        Alors A( )
                                        Sinon erreur := vrai
                                Fsi ;
                             Sinon tc := tc+1 ;
     Fsi ;
FinA
Procédure B( )
Debut
     tc := tc+1 ;
FinB
```

Analyse de la chaine cba# :

Pile	Chaine	Action
# Z	cba#	Appel S
# ZS	cba#	Appel A
# ZSA	cba#	Appel B
# ZSAB	cba#	tc :=tc+1 ; FinB
# ZSA	ba#	Appel A
# ZSAA	ba#	tc :=tc+1 ; FinA
# ZSA	a#	FinA
# ZS	a#	tc :=tc+1 ; FinS
# Z	#	chaine correcte

Remarques :

- Si la grammaire est récursive gauche, nécessité de la transformer.
- Si l'on veut faire une analyse déterministe, il faut factoriser la grammaire.
- Pour faire une analyse par la descente récursive, il faut que les conditions LL(1) soient vérifiées.
- La méthode présente un inconvénient, car elle est liée à la grammaire.
- Méthode simple à implémenter et est efficace.

II.3.3 Analyse LL(1) :

Il est possible de construire un analyseur prédictif non récursif en utilisant une pile de façon explicite. Le problème est en ayant une grammaire et une chaine d'entrée, de déterminer les productions à appliquer, en portant de l'axiome afin d'arriver à la chaine. L'analyseur fait la recherche dans une table, appelée table d'analyse (table LL(1)).

La table d'analyse est à deux dimensions, les lignes contiennent les non-terminaux et les colonnes les terminaux (T[1…|N|, 1…|TU{#}|]). T[A, a] donne le numéro de la règle, si elle existe, à utiliser pour dériver A afin d'atteindre a.

Algorithme de construction de la table d'analyse LL(1) :

Debut

Pour chaque non-terminal A€N

Faire Pour chaque régle A ⟶ α

Faire Pour chaque a€Debut(α)-{ε}

Faire T[A, a] := N° de la règle A ⟶ α Fait ;

Si ε € Debut(α)

Alors Pour chaque a € Suivant(A)

Faire T[A, a] := N° de la règle A ⟶ α Fait ;

Fsi ;

Fait ;

Fait ;

Fin

Remarques :

- Chaque entrée non définie de la table d'analyse LL(1) correspond à une erreur.
- Une table d'analyse d'une grammaire LL(1) doit être monodéfinie (Au plus une règle par entrée).

Définition 2 :

Une condition nécessaire et suffisante pour qu' une grammaire soit LL(1) est qu'elle soit :

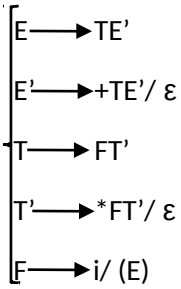
- Non récursive gauche.
- Factorisée.
- Table d'analyse LL(1) monodéfinie.

Définition 3 :

Une grammaire est LL(1), Si :

- Non récursive gauche.
- Factorisée.
- A ⟶ α1/ α2/……./αn
  . Debut(αi)∩Debut(αj)=Ø ∀ i, j avec i≠j
  . Si αi $\overset{*}{\Rightarrow}$ ε Alors Debut(αj)∩Suivant(A)=Ø ∀ j avec j≠i Fsi ;
  . Si αi $\overset{*}{\not\Rightarrow}$ ε Alors αj $\overset{*}{\not\Rightarrow}$ ε ∀ j avec j≠i Fsi ;

Exemple :



Fonctionnement de l'analyseur :

Nous disposons de la chaine d'entrée, d'une table d'analyse LL(1), la chaine d'entrée comporte une marque de fin (#).

La pile à un instant donné contient des symboles de la grammaire, avec '#', aussi comme marque de fond initialement, la pile contient l'axiome de la grammaire au dessus de '#'.

Algorithme

Debut

Lire (chaine) ; tc := 1ére terme de la chaine ; empiler(#S) ; finanalyse := faux ;

Tantque non finanalyse

Faire Si SommetPile € T

Alors Si tc=SommetPile Alors tc :=tc+1 ; depiler ;

Sinon 'Erreur : chaine incorrecte' ; finanalyse :=vrai ;

Fsi ;

Sinon Si SommetPile € N

Alors Si T[SommetPile, tc]=Ø

Alors 'Erreur : chaine incorrecte' ; finanalyse :=vrai ;

Sinon A :=SommetPile ; depiler ;

Empiler l'image miroir du MDP de la règle T[A, tc] ;

Fsi ;

Sinon Si tc=# Alors 'chaine correcte' ; finanalyse :=vrai ;

Sinon 'Erreur : chaine incorrecte' ; finanalyse :=vrai ;

Fsi ;

Fsi ;

Fsi ;

Fait ;

Fin.

II.4 Méthodes d'analyse ascendantes :

Dans l'analyse ascendante, contrairement à la descendante, à partir de la chaine d'entrée, par une série de réductions, on atteinte l'axiome de la grammaire.

Les algorithmes d'analyse ascendante sont souvent plus compilqués que ceux de l'analyse descendante. Ils s'appliquent toutefois à un nombre plus grand des grammaires. C'est pour cette raison qu'ils sont très souvent utilisés. Ils sont ainsi à la base du système Yacc qui sert à écrire des compilateus sous le système Unix.

Le problème de l'analyse ascendante est de reconnaitre une entité dans la partie droite pour la réduire par le membre gauche.

Définition 1 :

Soit une grammaire G=<T, N, S, P>, une chaine x est une forme sentencielle, si x peut être obtenue par dérivation à partir de l'axiome.

$S \overset{*}{\Rightarrow} x \quad x \in (TUN)^*$

Définition 2 :

Soit une grammaire G=<T, N, S, P> et w=xuy une forme sentencielle, u est une phrase simple de la forme sentencielle w, relativement au non-terminal U, si :

$S \overset{*}{\Rightarrow} xUy \overset{*}{\Rightarrow} xuy$ (u MDP de U $\longrightarrow$ u)

Définition 3 :

Une réduction consiste à remplacer une phrase simple u dans une forme sentencielle w=xuy par U pour avoir xUy (u MDP de U $\longrightarrow$ u).

Remarques :

- La forme sentencielle initiale est la chaine d'entrée.
- Les réductions sont de gauche à droite si les phrases simples les plus à gauche sont réduites avant les autres.
- à la réduction de u en U, x ne doit pas contenir de phrases simples.

Exemple :

<Progr> $\longrightarrow$ debut <LD>,<LI>fin.

<LD> $\longrightarrow$ <LD>;d/ d

<LI> $\longrightarrow$ <LI>;i/ i

Analyser la chaine debut d ;d ;i ;i fin.

Pile	Chaine	Action
#	debut d ;d ;i ;i fin.#	décaler
#debut	d ;d ;i ;i fin.#	décaler
#debut d	;d ;i ;i fin.#	réduire d par <LD>

#debut <LD>	;d ;i ;i fin.#	décaler
#debut <LD>;	d ;i ;i fin.#	décaler
#debut <LD>;d	;i ;i fin.#	réduire <LD>;d par <LD>
#debut <LD>	;i ;i fin.#	décaler
#debut <LD>;	i ;i fin.#	décaler
#debut <LD>;i	;i fin.#	réduire i par <LI>
#debut <LD>;<LI>	;i fin.#	décaler
#debut <LD>;<LI>;	i fin.#	décaler
#debut <LD>;<LI>;i	fin.#	réduire <LI>;i par <LI>
#debut <LD>;<LI>	fin.#	décaler
#debut <LD>;<LI> fin	.#	décaler
#debut <LD>;<LI> fin. <Progr>	#	réduire debut<LD>;<LI> fin. par
#<Progr>	#	chaine correcte

Le problème de l'analyse ascendante est de délimiter ou déterminer la phrase simple la plus à gauche dans la chaine à analyser, afin de faire une réduction.

II.4.1 L'analyse LR(1) :

LR : L veut dire le parcours de l'entrée de la gauche vers la droite.

R signifie en construisent une dérivation droite inverse.

L'analyse LR une méthode d'analyse efficace déterminste. C'est une méthode d'analyse ascendante de gauche à droite. Elle utilise une pile et une table d'analyse LR. Pour construire la table d'analyse LR, on dispose de deux méthodes : méthodes des contextes et méthodes des Items.

Méthode des Items :

Une méthode pratique d'analyse LR est la méthode des items. Un item est une production de la grammaire, avec un point repérant une position de son MDP. La partir à gauche du point représente le sous arbre de la grammaire, ayant été déjà réduit, à un moment donné, au cours de l'analyse, et la partir à droite représente ce qui reste à analyser.

Définition 1 :

Un item LR(1) est de la forme [A ⟶ α.β, t], α : constitue ce qui a été déjà réduit avec A€N, α,β€ (TUN)* et t€Debut(T*.#). A ⟶ αβ est une règle de la grammaire.

Pour la construction des ensembles d'items d'une grammaire, nous avons besoin de deux fonctions :

- La fermeture d'un ensemble d'items.
- La fonction Goto.

Fermeture d'un ensemble d'items LR(1) :

Fonction fermeture(I)

Debut

Répéter

Pour chaque item de la forme [A⟶α.Bβ, t] de l'ensemble des items I

Faire Pour chaque règle B⟶μ

Faire Pour chaque t'€Debut(βt)

Faire Ajouter dans I, l'item [B⟶.μ, t']

Fait;

Fait;

Fait;

Jusqu' à ce qu'il n'ait plus d'item à rajouter dans I

Fermeture(I) := I;

Fin.

Exemple :

Soit G : S⟶Sbc/ a

Les items LR(1) :

I0={[Z⟶.S, #], [S⟶.Sbc, #], [S⟶.a, #], [S⟶.Sbc, b], [S⟶.a, b]}

Fonction Goto :

Goto(I, X)

I : ensemble d'items.

X : symbole de la grammaire, X€(TUN).

Fonction Goto(I, X)

Debut

J :=Ø;

Pour chaque item de I de la forme [A⟶α.Xβ, t]

Faire J :=JU{[A⟶αX.β, t]} ;

Fait ;

Goto(I, X) := fermeture(I) ;

Fin.

Exemple :

Soit G : S⟶Sbc/ a

Les items LR(1) :

I0={[Z⟶.S, #], [S⟶.Sbc, #], [S⟶.a, #], [S⟶.Sbc, b], [S⟶.a, b]}

I1=Goto(I0, S)={ [Z⟶S., #], [S⟶S.bc, #], [S⟶S.bc, b]}

I2=Goto(I0, a)={ [S⟶a., #], [S⟶a., b]}

Ø=Goto(I0, b)

Construction de l'ensemble canonique des items LR(1) :

C : ensemble des états de l'automate.

Debut

I0={[Z⟶.S, #]};

C := {fermeture(I0)};

Répéter

Pour chaque ensemble d'items I de C

Faire Pour chaque X€TUN

Faire J :=Goto(I, X) ;

Si J≠Ø Alors ajouter J dans C Fsi ;

Fait ;

Fait ;

Jusqu'à ce qu'il n'y ait plus d'élément à rajouter dans C

Fin.

Implémentation de la méthode LR par la méthode des items :

- Construction de l'ensemble canonique des items C.
- 1. Si Ij=Goto(Ii, A), A€N Alors T[Ii, A] :=Ij Fsi.
- 2. Si Ij=Goto(Ii, a), a€T Alors T[Ii, a] :=D,Ij (*décalage d'un caractère*) Fsi.
- 3. Si dans Ii, on a un item de la forme [A ⟶ α., t]
  Alors T[Ii, t] :=RN°de la règle A ⟶ α (*réduction numéro de la règle A ⟶ α*)
  Fsi;

Définition 2 :

G est LR(1), ssi la table d'analyse LR(1), déduite de l'ensemble canonique d'items C est monodéfinie.

Exemple :

Soit G : S⟶1 Sbc/ a 2

Les items LR(1) :

I0={[Z ⟶ .S, #], [S ⟶ .Sbc, #], [S ⟶ .a, #], [S ⟶ .Sbc, b], [S ⟶ .a, b]}

I1=Goto(I0, S)={ [Z ⟶ S., #], [S ⟶ S.bc, #], [S ⟶ S.bc, b]}

I2=Goto(I0, a)={ [S ⟶ a., #], [S ⟶ a., b]}

I3=Goto(I1, b)={ [S ⟶ Sb.c, #], [S ⟶ Sb.c, b]}

I4=Goto(I3, c)={ [S ⟶ Sbc., #], [S ⟶ Sbc., b]}

Table d'analyse LR(1) :

	a	b	c	#	S
I0	D,I2				I1
I1		D,I3		Acc	
I2		R2		R2	
I3			D,I4		
I4		R1		R1	

Analyse de la chaine abc# :

Pile	Chaine	Action
I0	abc#	D,I1
I0 a I1	bc#	R : S ⟶ a
I0 S I1	bc#	D,I3
I0 S I1 b I3	c#	D,I4
I0 S I1 b I3 c I4	#	R : S ⟶ Sbc
I0 S I1	#	chaine accéptée

Conclusion :

Les analyseurs LR sont construits pour reconnaitre pratiquement tous langages de programmation. Ils sont très performantes, mais leurs tables d'analyes peuvent atteindre des centaines de milliers d'états, de ce fait sont couteûses à implémenter.

Pour cela, on a pensé aux grammaires SLR et LALR qui permettent l'optimisation des grammaires LR.

II.4.2 Grammaires SLR :

Le problème majeur associé à l'analyser LR(1) est la taille de la table d'analyse. Pour cela, une méthode d'analyse plus simple, et dont la table est plus réduite a été conçu, analyse SLR. Cependant peu de grammaires sont SLR.

Construction de la table d'analyse SLR(1) par la méthode des items LR(0) :

- Construction de l'ensemble canonique C des items LR(0).

- Déduire la table d'analyse SLR(1) à partir de C en appliquant :
- 1. Si Ij=Goto(Ii, A), A€N Alors T[Ii, A] :=Ij Fsi.
- 2. Si Ij=Goto(Ii, a), a€T Alors T[Ii, a] :=D,Ij (*décalage d'un caractère*) Fsi.

- 3. Si dans Ii, on a un item de la forme [A ⟶ α.]
  Alors Pour chaque t€Suivant(A)
  Faire
  T[Ii, t] :=RN°de la règle A ⟶ α (*réduction numéro de la règle A ⟶ α*)
  Fait ;
  Fsi;

Définition :

G est SLR(1), ssi la table d'analyse SLR(1), déduite de l'ensemble canonique d'items C est monodéfinie.

Exemple :

Soit G : S⟶aSb/ ε

Les items LR(0) :

I0={[Z⟶.S], [S⟶.aSb], [S⟶.]}

I1=Goto(I0, S)={[Z⟶S.]}

I2=Goto(I0, a)={[S⟶a.Sb], [S⟶.aSb], [S⟶.]}

I3=Goto(I2, S)={[S⟶aS.b]}

I2=Goto(I2, a)

I4=Goto(I3, b)={[S⟶aSb.]}

	Debut	Suivant
S	a ε	b #

Table d'analyse SLR(1) :

	a	b	#	S
0	D,2	R,2	R,2	1
1		Acc		
2	D,2	R,2	R,2	3
3		D,4		
4		R,1	R,1	

Table d'analyse SLR(1) monodéfinie alors G est SLR(1).

II.4.3 Grammaires LALR(1) :

L'analyse SLR est une méthode optimale, facile à implémenter, cependant peu de grammaires sont SLR.

La méthode LALR a alors été conçu, méthode intermédiaire du point de vue puissance. Elle est souvent suffisante et produit des tables de même importance que SLR.

Méthode de construction de la table d'analyse LALR :

1- Construire l'ensemble canonique d'items C={I0,…….,In}, items LR(1) pour la grammaire G.
2- Pour chaque cœur presénte dans C, trouver tous les items ayant ce même cœur et remplacer ces items par leur union.
3- Soit C'={J0, J1,….., Jm} la collection des ensembles d'items LR(1) resultante.
Pour chacun de ceux-ci, construire la table d'analyse LR(1). Si ces conditions conduisent à un conflit, la grammaire G est non LALR(1).

Exemple :

G : $\begin{cases} S \longrightarrow CC \\ C \longrightarrow cC / d \end{cases}$

G est-elle LR(1)?

Les items LR(1) :

I0=[Z⟶.S, #]

[S⟶.CC, #]

[C⟶.cC, c/d]

[C⟶.d, c/d]

I1=Goto(I0, S)=[Z⟶S., #]

I2=Goto(I0, C)=[S⟶C.C, #]

[C⟶.cC, #]

[C⟶.d, #]

I3=Goto(I0, c)=[C⟶c.C, c/d]

[C⟶.cC, c/d]

[C⟶.d, c/d]

I4=Goto(I0, d)=[C⟶d., c/d]

I5=Goto(I2, C)=[S⟶CC., #]

I6=Goto(I2, c)=[C⟶c.C, #]

[C⟶.cC, #]

[C⟶.d, #]

I7=Goto(I2, d)=[C⟶d., #]

I8=Goto(I3, C)=[C⟶cC., c/d]

I9=Goto(I6, C)=[C⟶cC., #]

I3=Goto(I3, c)

I4=Goto(I3, d)

I6=Goto(I6, c)

I7=Goto(I6, d)

Table d'analyse LR(1) :

	c	d	#	S	C
0	D,3	D,4		1	2
1			Acc		
2	D,6	D,7			5
3	D,3	D,4			8
4	R3	R3			
5			R1		
6	D,6	D,7			9
7			R3		
8	R2	R2			
9			R2		

Table d'analyse LR(1) monodéfinie alors G est LR(1).

I36=[C⟶c.C, c/d/#]

[C⟶.cC, c/d/#]

[C⟶.d, c/d/#]

I47=[C⟶d., c/d/#]

I89=[C⟶cC., c/d/#]

Table d'analyse LALR(1) :

	c	d	#	S	C
0	D,36	D,47		1	2
1			Acc		
2	D,36	D,47			5
36	D,36	D,47			89
47	R3	R3	R3		
5			R1		

89	R2	R2	R2

Table d'analyse LALR(1) est monodéfinie alors Gest LALR(1).

Remarques :

- G est LL(1) ⟹ G est LR(1).
- G est SLR(1) ⟹ G est LALR(1) ⟹ G est LR(1).
- Une grammaire est ambiguë n'est ni LL(1), ni LR(1), ni SLR(1), ni LR(1).
  Pour reconnaitrer les mots d'un tel langage on a deux possibilites :
  . Transformer la grammaire en une grammaire équivalente LL(1) où LR(1) où SLR(1) où LALR(1).
  . Construire la table d'analyse de la grammaire ambiguë et essayer d'éliminer les multidéfinitions en adoptant certaines conventions.

II.5 Etude des multidéfinitions dans la table d'analyse LR(1) :

1ère cas : Décalage/Réduction

1- T[Ii, a] := D,Ij
2- T[Ii, a] := RN°A ⟶ α

1⟹] dans Ii, un item de la forme [B ⟶ β.aμ, t]

2⟹] dans Ii, un item de la forme [A ⟶ α., a]

2ième cas : Réduction/Réduction

1- T[Ii, a] := RN°A ⟶ α
2- T[Ii, a] := RN°B ⟶ μ

1⟹] dans Ii, un item de la forme [A ⟶ α., a]

2⟹] dans Ii, un item de la forme [B ⟶ μ., a]

Remarque :

Le cas décalage/décalage ne peut pas se présenter

1- T[Ii, a] := D, Ij
2- T[Ii, a] := D, Il

1⟹] dans Ii, un item de la forme [A ⟶ α.aβ, t]

2⟹] dans Ii, un item de la forme [B ⟶ μ.aα', t']

IJ Ξ Il

Exemple :

G : S ⟶ Sbc/AB
A ⟶ aA/b
B ⟶ bB/ε

Les items LR(1) :

I0={[Z⟶.S, #], [S⟶.Sbc, #/b], [S⟶.AB, #/b], [A⟶.aA, #/b], [A ⟶.b, #/b]}

I1=Goto(I0, S)={[Z⟶S., #], [S⟶S.bc, #/b]}

I2=Goto(I0, A)={[S⟶A.B, #/b], [B⟶.bB, #/b], [B⟶., #/b]}

I3= Goto(I2, b)={[B⟶b.B, #/b]}

➡ Multidéfinition de la forme Réduction/Décalage

T[I2, b] := D, I3

T[I2, b] := RN°B⟶ε

Exemple :

G : S⟶Sbc/AB/Bb
A⟶aA/ε
B⟶bB/ε

I0={[Z⟶ .S, #], [S⟶ .Sbc, #/b], [S⟶ .AB, #/b], [A⟶ .aA, #/b], [A⟶ ., #/b], [S⟶ .Bb, #/b], [B .bB, b], [B ., b]}

➡ Multidéfinition de la forme Réduction/Réduction

	a	b	c	#	S	A	B
I0		R5/R7		R5			

Avec R5=A⟶ε et R7=B⟶ε