CHAPITRE 5 : ANALYSE SYNTAXIQUE : METHODES ASCENDANTES

5.1 Introduction à l'analyse ascendante

Les méthodes ascendantes construisent l'arbre syntaxique de bas en haut, en partant de la chaîne analysée (feuilles de l'arbre), puis, en assemblant, par des réductions, les sous-arbres sous les nouveaux nœuds non terminaux jusqu'à l'axiome (racine de l'arbre).

Le modèle général utilisé en analyse ascendante est le modèle par **décalage-réduction** (**shift-reduce**) qui autorise deux opérations :

- décaler (shift) : décaler, d'un symbole, le pointeur sur la chaîne d'entrée.
- réduire (reduce) : réduire une chaîne par un non terminal en utilisant une des règles de production, sachant que la chaîne réduite est une suite de terminaux et non terminaux à gauche du pointeur sur l'entrée et finissant sur ce pointeur.

Exemple

Soit la grammaire G ayant les règles de production suivantes :

S → aABe
A → Abc| b
B → d

On se propose d'analyser la chaîne abbcde de manière ascendante.

a b b c d e	décaler
a **b** b c d e	réduire
a **A** b c d e	décaler
a A **b** c d e	décaler
a A b **c** d e	réduire
a **A** d e	décaler
a A **d** e	réduire
a A **B** e	décaler
a A B **e**	réduire
S	Analyse réussie

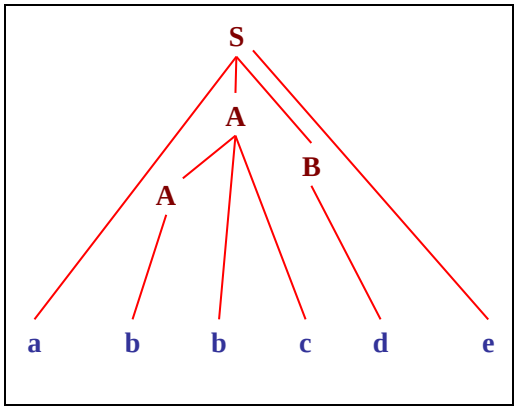

Figure 5.1. Arbre syntaxique obtenu de manière ascendante pour la chaîne abbcde

En résumé, on a donc les réductions suivantes, à partir desquelles on peut déduire, de manière ascendante, l'arbre syntaxique correspondant à la chaîne abbcde (voir figure 5.1) :

a b b c d e
a A b c d e
a A d e
a A B e
S

On constate que la séquence de réductions précédentes correspond, en sens inverse, aux dérivations droites suivantes : S → aABe → aAde → aAbcde → abbcde

Remarque : Il serait intéressant de disposer d'un moyen automatique pour choisir s'il faut effectuer un décalage ou une réduction et quelle réduction choisir lorsque le pointeur d'entrée se trouve sur une position donnée.

5.2 Introduction aux analyseurs LR

L'analyse LR (ou LR(k)) est une technique générale efficace d'analyse syntaxique ascendante, basée sur le modèle par décalage-réduction et qui peut être utilisée pour analyser une large classe de grammaires non contextuelles.

L : **L**eft to right scanning (on parcourt ou on analyse la chaîne en entrée de la gauche vers la droite)

R : constructing a **R**ightmost derivation in reverse (en construisant une dérivation droite inverse)

k : on utilise **k** symboles d'entrée de prévision à chaque étape nécessitant la prise d'une décision d'action d'analyse. Quand k est omis , il est supposé égal à 1.

Les analyseurs LR comportent :

- Un algorithme d'analyse commun aux différentes méthodes LR.
- Une table d'analyse dont le contenu diffère selon le type d'analyseur LR.

On distingue trois techniques de construction de tables d'analyse LR pour une grammaire donnée :

- **Simple LR (SLR)** : qui est la plus simple à implémenter, mais la moins puissante des trois.
- **LR canonique** : qui est la plus puissante, mais aussi la plus coûteuse.
- **LookAhead LR (LALR)** : qui a une puissance et un coût intermédiaires entre les deux autres et peut être appliquée à la majorité des grammaires de langages de programmation.

5.2.1 Modèle d'analyseur LR

Un analyseur LR peut être assimilé à un automate à pile, il est modélisé, comme le montre la figure 5.2, par un tampon d'entrée, un flot de sortie, une pile, un programme d'analyse (le même pour tous les analyseurs LR) et des tables d'analyse divisées en deux parties : Action et Successeur (Goto).

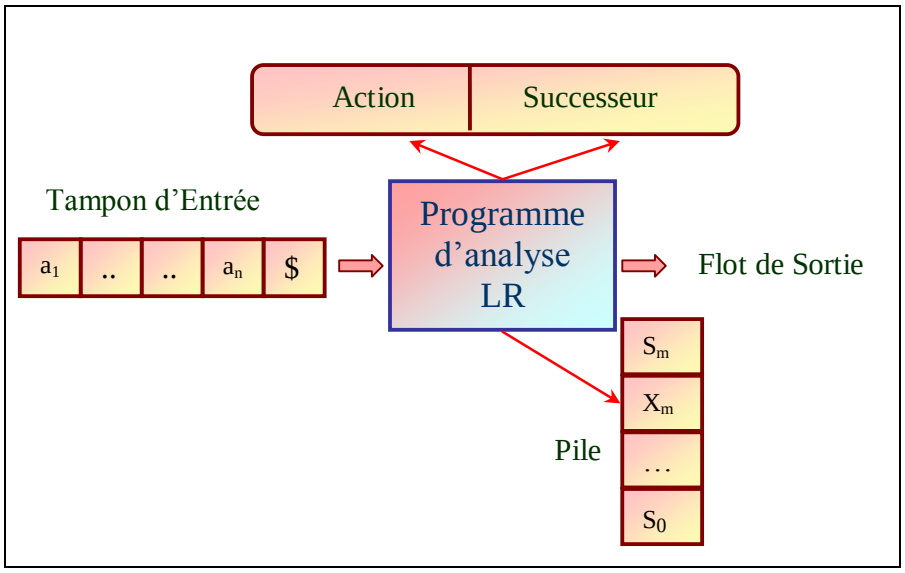

Figure 5.2. Modèle d'un analyseur LR

Le programme d'analyse range dans la pile des chaînes de la forme $S_0X_1S_1X_2S_2...X_mS_m$. Chaque X_i est un symbole de la grammaire ($X_i \in (V_T \cup V_N)$) et chaque S_i est un état (state) de l'analyseur qui résume l'information contenue dans la pile au dessous de lui, la combinaison du numéro de l'état en sommet de pile et du symbole d'entrée courant est utilisé pour indicer les tables et déterminer l'action à effectuer (décaler ou réduire).

Les tables d'analyse contiennent deux parties : une fonction d'action d'analyse (action) et une fonction de transfert (successeur). L'information contenue dans ces champs représente la différence entre les analyseurs LR.

5.2.2 Algorithme d'analyse LR

Le programme d'analyse LR détermine S_m (l'état en sommet de pile) et a_i (le symbole terminal d'entrée courant). Ceci correspond à la configuration de l'analyseur ($S_0X_1S_1X_2S_2...X_mS_m$, a_ia_{i+1} ...a_n\$) sachant que la configuration initiale est (S_0, a_1a_2 ...a_n\$). L'analyseur consulte **Action [S_m, a_i]**, dans la table des actions, qui peut avoir une des quatre valeurs : *décaler, réduire, accepter* ou *erreur*.

1er cas : Action [S_m, a_i]=décaler S (shift S) :
L'analyseur empile a_i et S, a_{i+1} devient le symbole d'entrée courant. La configuration de l'analyseur devient donc : ($S_0X_1S_1X_2S_2...X_mS_ma_iS$, a_{i+1} ...a_n\$).

2ème cas : Action [S_m, a_i]=réduire par A→β:
L'analyseur dépile 2r symboles, r symboles d'états et r symboles de grammaire, avec r=|β|=longueur de β. S_{m-r} devient le nouveau sommet de pile, ensuite l'analyseur empile A (partie gauche de la production A→β) et S (entrée pour Successeur[S_{m-r}, A]).

Remarques :
Le symbole d'entrée courant ne change pas avec une réduction.
La séquence dépilée $X_{m-r+1}...X_m$ correspond à β. La configuration de l'analyseur devient donc :
($S_0X_1S_1X_2S_2...X_{m-r}S_{m-r}AS$, a_ia_{i+1} ...a_n\$).

3 ème cas : Action [S_m, a_i]=accepter:
L'analyse est terminée en acceptant la chaîne analysée.

4ème cas : Action [S_m, a_i]=erreur :
L'analyse est terminée en rejetant la chaîne analysée.

Résumé de l'algorithme d'analyse LR

Entrée : Chaîne w et table d'analyse LR

Sortie : Résultat de l'analyse ascendante de w si w ∈ L(G) (arbre syntaxique) ou échec si w∉L(G).

Méthode : on commence avec S_0 dans la pile, w\$ dans le tampon d'entrée, le pointeur source ps est initialisé au début de w. L'analyseur exécute la boucle *Répéter* jusqu'à ce qu'il rencontre une action *Accepter* ou *Erreur*.

```
Algorithme Analyse LR;
Répéter indéfiniment
Début
  Soit S l'état en sommet de pile et a le symbole pointé par ps
  Si Action [S, a]= Décaler S' Alors
  Début
        Empiler a puis S' ;
        Avancer ps ;
  Fin ;
  Sinon Si Action [S, a]= Réduire par A→β Alors
          Début
             Dépiler 2|β| symboles ;
              Soit S' le nouveau sommet de pile ;
             Empiler A puis Successeur[S', A] ;
             Emettre en sortie A→β ;
          Fin
          Sinon Si Action [S, a]= Accepter Alors Return Sinon Erreur() ;
Fin ;
```

5.2.3 Exemple d'application de l'algorithme d'analyse LR

Soit la grammaire G ayant les règles de production suivantes :

$E \rightarrow E+T$	①	$T \rightarrow F$	④
$E \rightarrow T$	②	$F \rightarrow (E)$	⑤
$T \rightarrow T*F$	③	$F \rightarrow id$	⑥

Soit la table d'analyse LR supposée déjà construite :

Etat	**Action**						**Successeur**		
	+	*	(	)	id	$	E	T	F
0			d_4		d_5		1	2	3
1	d_6					acc			
2	r_2	d_7		r_2		r_2			
3	r_4	r_4		r_4		r_4			
4			d_4		d_5		8	2	3
5	r_6	r_6		r_6		r_6			
6			d_4		d_5			9	3
7			d_4		d_5				10
8	d_6			d_{11}					
9	r_1	d_7		r_1		r_1			
10	r_3	r_3		r_3		r_3			
11	r_5	r_5		r_5		r_5			

L'action d_i signifie décaler et empiler l'état i.
L'action r_j signifie réduire par la production dont le numéro est j.
L'action acc signifie accepter la chaîne analysée.
Une entrée vide correspond à une erreur.

On se propose maintenant d'analyser la chaîne id *id+id.

Pile	**Entrée**	**Sortie**
S_0	id*id+id$	Décaler (d_5)
$S_0 id S_5$	*id+id$	Réduire F→id
$S_0 F S_3$	*id+id$	Réduire T→F
$S_0 T S_2$	*id+id$	Décaler (d_7)
$S_0 T S_2 * S_7$	id+id$	Décaler(d_5)
$S_0 T S_2 * S_7 id S_5$	+id$	Réduire F→id
$S_0 T S_2 * S_7 F S_{10}$	+id$	Réduire T→T*F
$S_0 T S_2$	+id$	Réduire E→T
$S_0 E S_1$	+id$	Décaler (d_6)
$S_0 E S_{1+} S_6$	id$	Décaler (d_5)
$S_0 E S_1 + S_6 id S_5$	$	Réduire F→id
$S_0 E S_1 + S_6 F S_3$	$	Réduire T→F
$S_0 E S_1 + S_6 T S_9$	$	Réduire E→E+T
$S_0 E S_1$	$	Accepter

5.3 Construction de table d'analyse SLR

La construction d'une table d'analyse SLR (LR(0) ou SLR(1)) à partir d'une grammaire est la méthode la plus simple à implémenter (parmi les 3 méthodes SLR, LALR et LR canonique) mais c'est la moins puissante du point de vue du nombre de grammaire pour lesquelles elle réussit. Elle constitue un bon point de départ pour l'étude de l'analyse LR. Le principe de la méthode SLR est de construire, à partir de la grammaire un automate fini déterministe qui reconnaît les préfixes viables de G (préfixes pouvant apparaître sur la pile) puis de transformer cet automate en une table d'analyse.

La construction des analyseurs SLR, pour une grammaire donnée, est basée sur la collection d'ensembles d'items LR(0). Pour construire cette collection, on définit une **grammaire augmentée**, une fonction **Fermeture** et une fonction **Transition**. Dans la section 5.3.1, on commence par définir les concepts nécessaires à la construction de collection d'ensembles d'items LR(0) et de table d'analyse SLR.

5.3.1 Concepts de base

5.3.1.1 Items LR(0)

Un item LR(0) d'une grammaire G est une production de G avec un point (.) repérant une position de sa partie droite.

Exemples

La production A→XYZ fournit 4 items : A→.XYZ, A→X.YZ, A→XY.Z et A→XYZ.

La production A→ ε fournit un seul item A→.

Remarque

Intuitivement, un item indique la «quantité» de partie droite qui a été reconnue à un instant donné de l'analyse. Par exemple, l'item A→X.YZ indique qu'on vient de voir en entrée une chaîne dérivée de X et qu'on espère maintenant voir une chaîne dérivée de YZ.

5.3.1.2 Grammaire augmentée

Si G est une grammaire d'axiome S, alors la grammaire augmentée de G (notée G') aura un nouvel axiome S' et une nouvelle règle de production S'→S. Le but de l'ajout de cette production est d'indiquer à l'analyseur quand il doit s'arrêter et annoncer l'acceptation de la chaîne d'entrée (quand l'analyseur est sur le point de réduire S par S').

5.3.1.3 Fonction Fermeture d'un ensemble d'items LR(0)

Soit I un ensemble d'items pour une grammaire G. Fermeture (I) est l'ensemble d'items construit à partir de I par l'application des 2 règles suivantes :

Règle 1 : initialement placer chaque item de I dans Ferm(I)

Règle 2 : si A→α.Bβ ∈ Ferm(I) et B → δ est une production de G, alors ajouter B→.δ à Ferm(I), s'il n'y existe pas déjà. La règle 2 doit être appliquée jusqu'à ne plus pouvoir ajouter de nouveaux items à Ferm(I).

Ainsi, la fonction Ferm(I) peut être définie de la manière suivante :

```
Fonction Ferm(I);
Début
  J :=I ;
  Répéter
    Pour chaque item A→α.Bβ ∈ J et chaque production B → δ telle que B→.δ ∉ J
    Faire Ajouter B → .δ à J
  Jusqu'à ce qu'aucun item ne puisse être ajouté à J ;
  Ferm(I) :=J;
Fin ;
```

Exemple

Soit la grammaire G, ayant les règles de production suivantes :

E → E+T | T
T → T*F | F
F → (E) | id

Soit l'ensemble d'items I= {[E'→.E]}. On se propose de calculer Ferm(I). En appliquant les deux règles précédentes, on trouve : Ferm(I) = { [E'→.E], [E→.E+T], [E→.T], [T→.T*F], [T→.F], [F→.(E)], [F→.id] }

5.3.1.4 Fonction Transition

Soit I un ensemble d'items et X un symbole de la grammaire ($X \in V_T \cup V_N$). Transition (I, X) est définie comme étant la fermeture de tous les items $[A \rightarrow \alpha X.\beta]$ tels que $[A \rightarrow \alpha.X\beta] \in I$.

Exemple

En utilisant la grammaire précédente, on se propose de déterminer, pour I= {[E'→E.], [E→E.+T]}, la fonction Trans(I, +).

Trans(I, +)= Ferm ([E→E+.T]) = {[E→E+.T], [T→.T*F], [T→.F], [F→.(E)], [F→.id]}

5.3.2 Construction de la collection d'ensembles d'items LR(0)

La collection canonique d'ensembles d'items LR(0) correspond aux états de l'AFD permettant de reconnaître les préfixes viables de la grammaire. La détermination de cette collection constitue la base de la construction des tables d'analyses SLR. L'algorithme suivant permet de déterminer C : la collection canonique d'ensemble d'items LR(0) pour une grammaire augmentée G'

```
Procédure CollectionEnsembleItems;
Début
C := {Ferm({[S'→.S]})} ;
Répéter
    Pour chaque ensemble d'items I de C et pour chaque symbole
    de la grammaire X tel que Trans(I, X) soit non vide et non encore dans C
    Faire ajouter Trans(I, X) à C
  Jusqu'à ce qu'aucun ensemble d'items ne puisse être ajouté à C;
Fin ;
```

Exemple

On se propose de déterminer la collection canonique d'ensembles d'items de la grammaire augmentée suivante :

E'→E
E → E+T | T
T → T*F | F
F → (E) | id

I_0 = Ferm {[E'→.E]}
= E'→.E
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id

I_1= Trans(I_0, E) = Ferm {[E'→E.], [E'→E.+T]}
= E'→E.
E→E.+T

I_2 = Trans(I_0, T) = Ferm {[E→T.], [T→T.*F] }
= E→T.
T→T.*F

I_3 = Trans(I_0, F) = Ferm {[T→F.]}
= T→F.

I_4 = Trans(I_0, ( )= Ferm {[F→(.E)]}
= F→(.E)
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id

I_5 = Trans(I_0,id) = Ferm {[F→id.]}
= F→id.

I_6 = Trans(I_1,+) = Ferm {[E→E+.T]}
= E→E+.T
T→.T*F
T→.F
F→.(E)
F→.id

I_7 = Trans(I_2,*) = Ferm {[T→T*.F]}
= T→T*.F
F→.(E)
F→.id

I_8 = Trans(I_4,E) = Ferm {[F→(E.)], [E→E.+T] }
= F→(E.)
E→E.+T
I_2 = Trans(I_4, T)= Ferm {[E→T.], [T→T.*F]}
I_3 = Trans(I_4, F) = Ferm {[T→F.]}
I_4 = Trans(I_4, () = Ferm {[F→(.E)]}
I_5 = Trans(I_4,id) = Ferm {[F→id.]}

I_9 = Trans(I_6, T)= Ferm {[E→E+T.], [T→T.*F]}
= E→E+T.
T→T.*F
I_3 = Trans(I_6, F) = Ferm {[T→F.]}
I_4 = Trans(I_6, () = Ferm {[F→(.E)]}
I_5 = Trans(I_6,id) = Ferm {[F→id.]}

I_{10} = Trans(I_7,F)= Ferm {[T→T*F.]}
= T→T*F.
I_4= Trans(I_7, () = Ferm {[F→(.E)]}
I_5= Trans(I_7.id) = Ferm {[F→id.]}

I_{11}=Trans(I_8,))= Ferm {[F→(E).]}
= F→(E).
I_6 = Trans(I_8, +) = Ferm {[E→E+.T]}
I_7 = Trans(I_9, *) = Ferm {[T→T*.F]}

La fonction de transition pour la collection canonique d'ensembles d'items de la grammaire augmentée considérée dans cet exemple est donnée sous forme d'un AFD représenté par la figure 5.3.

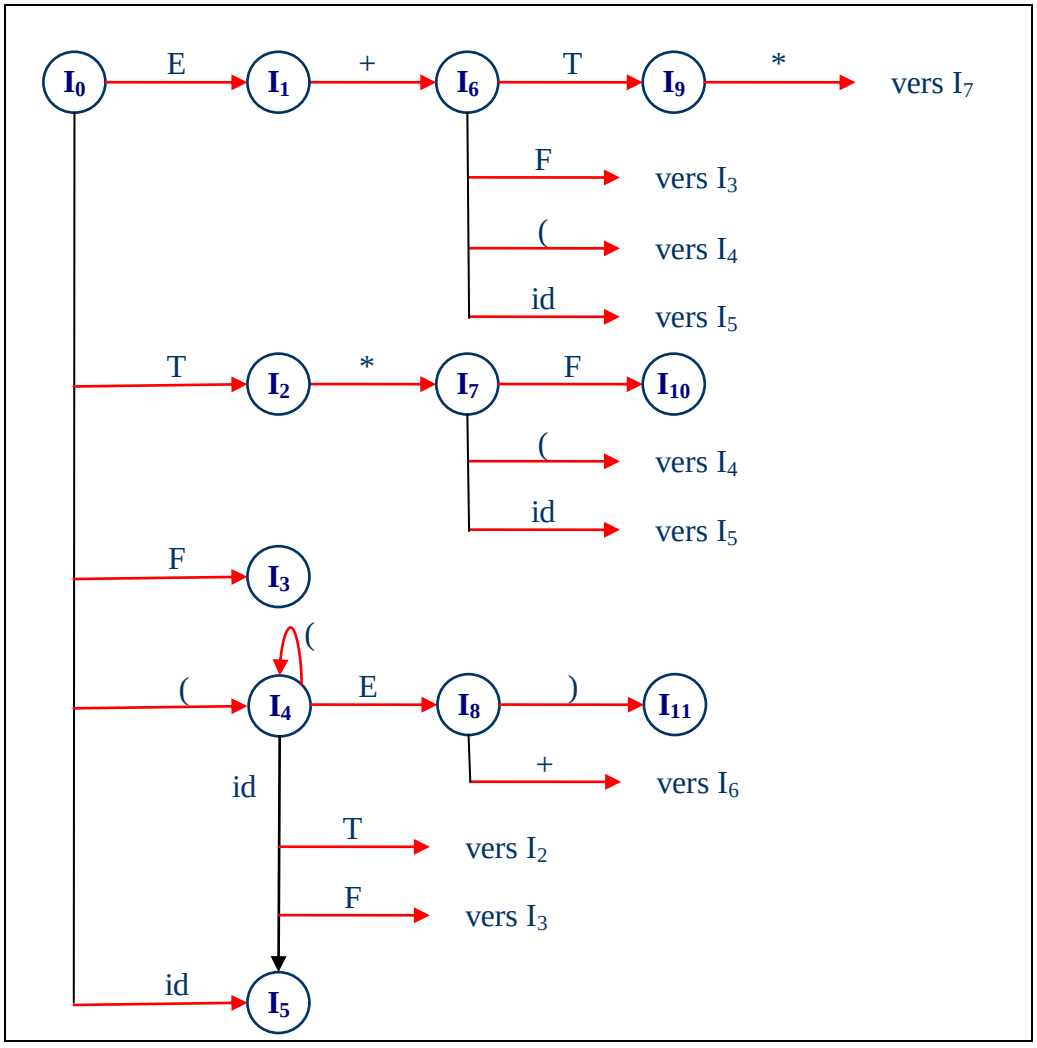

Figure 5.3. Représentation de l'AFD correspondant à l'exemple

5.3.3 Algorithme de construction de table d'analyse SLR

La construction d'une table d'analyse SLR pour une grammaire G est effectuée de la manière suivante :

- Effectuer une augmentation de G pour obtenir la grammaire augmentée G'
- Construire C la collection canonique d'ensembles d'items LR(0) pour G', soit C={$I_0, I_1, I_2, \ldots\ldots I_N$}
- Construire les fonctions ACTION (actions d'analyse) et SUCCESSEUR (transition) à partir de C en utilisant l'algorithme suivant :

Algorithme ConstructionTableSLR;
Début
1/ L'état i est construit à partir de I_i. La partie ACTION pour l'état i est déterminée comme suit:
a. **Si** $[A \rightarrow \alpha.a\beta] \in I_i$ et Trans (I_i, a)= I_j (avec $a \in V_T$) **Alors** ACTION [i, a]$\leftarrow d_j$ (décaler j)
b. **Si** $[A \rightarrow \alpha.] \in I_i$ (avec $A \neq S'$) **Alors** ACTION [i, a] $\leftarrow r_{num}$ pour tout $a \in$ Suiv(A)
(réduire par la régle $A \rightarrow \alpha$ dont le numéro est num)
c. **Si** $[S' \rightarrow S.] \in I_i$ **Alors** ACTION [i, $] $\leftarrow$ accepter
2/ La partie SUCCESSEUR pour l'état i est déterminée comme suit :
Si Trans (I_i, A)= I_j **Alors** SUCCESSEUR [i, A] $\leftarrow$j
3/ Toutes les entrée restantes dans la table sont mises à ERREUR
4/ L'état initial de l'analyseur est construit à partir de l'ensemble d'items contenant $[S' \rightarrow .S]$
Fin ;

Exemple

En appliquant l'algorithme ci-dessus, on se propose de construire la table SLR correspondant à la grammaire G de l'exemple précédent (dont on a déterminé la collection canonique d'ensembles d'items). On obtient la table SLR suivante :

Etat	Action						Successeur		
	+	*	(	)	id	$	E	T	F
0			d_4		d_5		1	2	3
1	d_6					acc			
2	r_2	d_7		r_2		r_2			
3	r_4	r_4		r_4		r_4			
4			d_4		d_5		8	2	3
5	r_6	r_6		r_6		r_6			
6			d_4		d_5			9	3
7			d_4		d_5				10
8	d_6			d_{11}					
9	r_1	d_7		r_1		r_1			
10	r_3	r_3		r_3		r_3			
11	r_5	r_5		r_5		r_5			

Sachant que :

Suiv(E)= {+,), $}

Suiv(T)= {∗, +,), $}

Suiv(F)= {∗, +,), $}

5.3.4 Grammaire SLR (1)

Les tables obtenues avec l'algorithme défini dans la section précédente (5.3.3) sont appelées les **tables SLR(1)**. Un analyseur utilisant ces tables est appelé **analyseur SLR(1)**. Une grammaire est dite SLR(1) ou SLR si elle peut être représentée par une table SLR(1) dont chaque entrée est définie de façon unique (d /r /acc /erreur).

Exemple

Soit la grammaire G, ayant les règles de production suivantes :

$S \rightarrow G=D \mid D$

$G \rightarrow *D \mid id$

$D \rightarrow G$

On se propose de montrer si cette grammaire est SLR(1).

On commence d'abord par augmenter la grammaire :

$S' \rightarrow S$

$S \rightarrow G=D$ ①

$S \rightarrow D$ ②

$G \rightarrow *D$ ③

$G \rightarrow id$ ④

$D \rightarrow G$ ⑤

Détermination de la collection canonique d'ensembles d'items :

I_0= Ferm {[S'→.S]}

= S'→.S

S→.G=D

S→.D

G→.*D

G→.id

D→.G

I_1= Trans (I_0, S) = Ferm{[S'→S.]}
= S'→S.
I_2= Trans (I_0, G) = Ferm {[S→G.=D], [D→G.]}
= S→G. =D
D→G.
I_3=Trans (I_0, D) = Ferm {[S→D.]}
= S→D.
I_4= Trans(I_0, *) =Ferm{[G→*.D]}
= G→*.D
D→.G
G→.*D
G→.id
I_5= Trans(I_0, id) = Ferm {[G→id.]}
= G→id.
I_6= Trans (I_2,=) = Ferm {[S→G=.D]}
=S→G=.D
D→.G
G→.*D
G→.id
I_7= Trans (I_4, D) = Ferm {[G→*D.]}
= G→*D.
I_8= Trans (I_4,G) = Ferm {[D→G.]}
= D→G.
I_4= Trans (I_4,*) = Ferm {[G→*.D]}
= G→*.D
I_5= Trans (I_4,id) = Ferm {[G→id.]}
I_9= Trans (I_6,D) = Ferm {[S→G=D.]}
= S→G=D.
I_8= Trans(I_6,G) = Ferm {[D→G.]}
= D→G.
I_4= Trans(I_6, *) = Ferm {[G→*.D]}
I_5= Trans(I_6, id) = Ferm {[G→id.]}
= G→id.

Construction de la table d'analyse SLR :

Etat	Action				Successeur		
	=	*	id	$	S	G	D
0		d_4	d_5		1	2	3
1				acc			
2	d_6/r_5			r_5			
3				r_2			
4		d_4	d_5			8	7
5	r_4			r_4			
6		d_4	d_5			8	9
7	r_3			r_3			
8	r_5			r_5			
9				r_1			

Sachant que :
Suiv(D)= {=, $}
Suiv(G)= {=, $}
Suiv(S)= { $}

Remarque

On constate que Action[2,=] est une case définie de façon multiple. L'état 2 présente un conflit d/r (déclarer/réduire) ou s/r (shift/reduce) sur le symbole d'entrée « = » donc la grammaire **n'est pas SLR (1)**.

Dans notre cas, la grammaire n'est pas ambiguë mais elle n'est pas SLR(1). D'ailleurs, beaucoup de grammaires non ambiguës ne sont pas SLR(1). Le conflit shift/réduce (décaler/réduire) dans l'exemple précédent provient du manque de puissance de la méthode SLR qui ne peut pas « se rappeler » assez de contexte gauche pour décider de l'action de l'analyseur sur l'entrée « = » après avoir « vu » une chaîne pouvant être réduite vers G.

Les méthodes LR canonique et LALR réussissent pour un nombre plus important de grammaires que la méthode SLR.

5.4 Construction de tables d'analyse LR canonique ou LR(1)

La construction de tables d'analyse LR canonique ou LR(1) est la technique la plus générale de construction de tables d'analyse LR pour une grammaire. Cette méthode permet d'attacher plus d'informations à un état pour éviter un certain nombre d'actions invalides et donc de conflits.

5.4.1 Items LR(1)

La forme générale d'un item LR(1) est $[A\rightarrow\alpha.\beta, a]$ sachant que :
$A\rightarrow\alpha\beta$ est une règle de production de la grammaire et $a \in V_T \cup \{\$\}$.
Le 1 du LR(1) fait référence à la longueur d second composant, appelé **symbole de pré-vision** de l'item. La prévision n'a aucun effet dans un item de la forme $[A\rightarrow\alpha.\beta, a]$ avec $\beta \neq \varepsilon$, mais un item de la forme $[A\rightarrow\alpha., a]$ implique « réduire par $A\rightarrow\alpha$ uniquement lorsque le prochain symbole entrée est a ». Ceci signifie que la réduction par $A\rightarrow\alpha$ ne se fait que sur les symboles d'entrée a pour lesquels $[A\rightarrow\alpha., a]$ est un item LR(1) de l'état en sommet de pile. L'ensemble de tels a sera toujours un sous ensemble de Suiv(A).

5.4.2 Fermeture d'ensemble d'items LR(1)

La fonction Ferm(I) peut être définie de la manière suivante :

```
Fonction Ferm(I);
Début
  Répéter
    Pour chaque item [A→α.Bβ, a] ∈ I, chaque production B → δ ∈ G'
    et chaque terminal b∈Prem(βa) tel que [B→.δ, b] ∉ I
    Faire Ajouter [B → .δ, b] à I
  Jusqu'à ce qu'aucun item ne puisse être ajouté à I;
  Ferm(I) := I;
Fin ;
```

5.4.3 Fonction Transition

Soit I un ensemble d'items LR(1) et X un symbole de la grammaire ($X\in V_T \cup V_N$). Transition (I, X) est définie de la manière suivante :

```
Fonction Trans(I, X);
Début
  Soit J l'ensemble des items [A→αX.β, a] tels que [A→α.Xβ, a] ∈ I.
  Trans(I, X) := Ferm(J);
Fin ;
```

5.4.4 Construction de la collection d'ensembles d'items LR(1)

L'algorithme suivant permet de déterminer la collection d'ensemble d'items LR(1) pour une grammaire augmentée G'

Procédure CollectionEnsembleItems;
Début
C := {Ferm({[S'→.S, $]})} ;
Répéter
  Pour chaque ensemble d'items I de C et pour chaque symbole
  de la grammaire X tel que Trans(I, X) soit non vide et non encore dans C
  Faire ajouter Trans(I, X) à C
Jusqu'à ce qu'aucun ensemble d'items ne puisse être ajouté à C;
Fin ;

5.4.5 Algorithme de construction de tables d'analyse LR(1)

La construction d'une table d'analyse LR(1) pour une grammaire G est effectuée de la manière suivante :

- Effectuer une augmentation de G pour obtenir la grammaire augmentée G'.
- Construire C la collection d'ensembles d'items LR(1) pour G'.
- Construire les fonctions ACTION (actions d'analyse) et SUCCESSEUR (transition) à partir de C en utilisant l'algorithme suivant :

Algorithme ConstructionTableLR;
Début
1/ L'état i est construit à partir de I_i. La partie ACTION pour l'état i est déterminée comme suit:
  a. **Si** $[A\rightarrow\alpha.a\beta, b] \in I_i$ et Trans $(I_i, a)= I_j$ (avec $a \in V_T$) **Alors** ACTION [i, a]$\leftarrow d_j$ (décaler j)
  b. **Si** $[A\rightarrow\alpha., a] \in I_i$ (avec $A\neq S'$) **Alors** ACTION [i, a] $\leftarrow r_{num}$
  (réduire par la régle $A\rightarrow\alpha$ dont le numéro est num)
  c. **Si** $[S'\rightarrow S., \$] \in I_i$ **Alors** ACTION [i, $] $\leftarrow$ accepter
2/ La partie SUCCESSEUR pour l'état i est déterminée comme suit :
  Si Trans $(I_i, A)= I_j$ **Alors** SUCCESSEUR [i, A] $\leftarrow$j
3/ Toutes les entrée restantes dans la table sont mises à ERREUR
4/ L'état initial de l'analyseur est construit à partir de l'ensemble d'items contenant $[S'\rightarrow.S, \$]$
Fin ;

5.4.6 Grammaire LR(1)

Les tables obtenues avec l'algorithme défini dans la section précédente (5.4.5) sont appelées les **tables canoniques d'analyse LR(1)**. Un analyseur utilisant ces tables est appelé **analyseur LR(1) canonique**. Une grammaire est dite LR(1) ou LR si elle peut être représentée par une table LR(1) dont chaque entrée est définie de façon unique.

Toute grammaire SLR(1) est une grammaire LR(1), mais, pour une grammaire SLR(1), l'analyseur LR canonique peut avoir un nombre d'états supérieur à l'analyseur SLR pour la même grammaire.

5.4.7 Exemple de construction de tables d'analyse LR(1)

Soit la grammaire G, ayant les règles de production suivantes :

$S \rightarrow CC$

$C \rightarrow cC \mid d$

On se propose de montrer si cette grammaire LR(1) *(Question 1 de l'exercice 6 de la série de TD no : 4)*.

- On commence d'abord par augmenter la grammaire :
  S'→S
  S → CC ①
  C → cC ②
  C → d ③
- Détermination de la collection d'ensembles d'items LR(1) :

I_0= Ferm {[S'→.S, $]}
= S'→.S, $ *Prem(βa)= Prem($)= {$}*
S→.CC, $ *Prem(βa)= Prem(C$)= {c, d}*
C→.cC, c/d
C→.d, c/d

I_1= Trans (I_0, S) = Ferm{[S'→S., $]}
= S'→S., $

I_2= Trans (I_0, C) = Ferm {[S→C.C, $]}
= S→C.C, $
C→.cC, $
C→.d, $

I_3=Trans (I_0, c) = Ferm {[C→c.C, c/d]}
= C→c.C, c/d
C→.cC, c/d
C→.d, c/d

I_4= Trans(I_0, d) =Ferm{ [C→d., c/d]}
= C→d., c/d

I_5= Trans(I_2, C) = Ferm {[S→CC., $]}
= S→CC., $

I_6= Trans (I_2, c)= Ferm {[C→c.C, $]}
= C→c.C, $
C→.cC, $
C→.d, $

I_7= Trans (I_2, d) = Ferm {[C→d., $]}
= C→d., $

I_8= Trans (I_3, C) = Ferm {[C→cC., c/d]}
= C→cC., c/d

I_3= Trans (I_3, c)= Ferm {[C→c.C, c/d]}
I_4= Trans (I_3, d) = Ferm {[C→d., c/d]}
I_9= Trans (I_6,C) = Ferm {[C→cC., $]}
= C→cC., $
I_6= Trans(I_6, c) = Ferm {[C→c.C, $]}
I_7= Trans(I_6, d) = Ferm {[C→d., $]}

- Construction de la table d'analyse LR :

Etat	**Action**			**Successeur**	
	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	r_3	r_3			
5			r_1		
6	d_6	d_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

5.5 Construction de tables d'analyse LALR

La méthode LALR(1) (Look Ahead LR) est souvent utilisée en pratique car c'est une méthode ayant un coût et une efficacité intermédiaire en comparaison avec SLR(1) et LR(1). Pour une grammaire donnée, la table d'analyse LALR(1) occupe autant de place (même nombre d'états) que la table SLR(1) (moins que la table LR(1)) et satisfait une grande classe de grammaires. Ainsi, il est plus intéressant, plus facile et plus économique de construire des tables SLR ou LALR que des tables LR canoniques. A titre d'exemple, les méthodes SLR et LALR donnent un nombre d'états de plusieurs centaines pour un langage tel que Pascal alors qu'on arrive à plusieurs milliers d'états avec la méthode LR canonique.

5.5.1 Algorithme de construction de tables LALR(1)

Cet algorithme se base sur la construction de la collection d'ensembles d'items LR(1) ainsi que la table d'analyse LR(1). Il utilise la notion de cœur d'item LR(1), sachant que : **item LR(1)=[cœur, symbole de pré-vision]**.

L'idée générale de cet algorithme est de construire les ensembles d'items LR(1) et de fusionner les ensembles ayant un cœur commun. La table d'analyse LALR(1) est construite à partir de la collection des ensembles d'items fusionnés. Signalons que cet algorithme est le plus simple, mais il existe d'autres algorithmes plus efficaces pour la construction de tables LALR(1), qui ne se basent pas sur la méthode LR(1).

Algorithme ConstructionTableLALR;
Début
1/ Construire la collection d'ensembles d'items LR(1) pour la grammaire augmentée G'.
2/ Pour chaque cœur présent parmi les ensembles d'items LR(1), trouver tous les états ayant ce même cœur et remplacer ces états par leur union.
3/ La table LALR(1) est obtenue en condensant la table LR(1) par superposition des lignes correspondant aux états regroupés.
Fin ;

Pendant la superposition des lignes, il y a un risque de conflit :

Cas 1 : Conflit décaler/décaler ou shift/shift (d_i/d_j)
Ce cas de conflit ne peut pas se produire car le décalage se base sur l'élément après le point.

Cas 2 : Conflit décaler/réduire ou shift/reduce (d/r)
Ce cas de conflit ne peut se produire que s'il existe dans la table LR(1) pour l'un, au moins des états d'origine.

Cas 3 : Conflit réduire/réduire ou reduce/reduce (r_i/r_j)
C'est le seul conflit possible. Il se produit quand deux items LR(1) ayant respectivement la [A→C.,x] et [B→C.,x] appartiennent à un ensemble obtenu après regroupement d'états.

Conclusion: Pendant la superposition des lignes, le seul cas de conflit qui peut se produire est réduire/réduire.

Remarque

S'il n'y a pas de conflit dans la table LALR(1), la grammaire sera considérée LALR(1) ou LALR.

5.5.2 Exemple de construction de tables d'analyse LALR(1)

On se propose de construire la table d'analyse LALR(1) pour la grammaire G, traitée dans la section 5.4.7 *(Question 2 de l'exercice 6 de la série de TD no :4)*:

$S \rightarrow CC$
$C \rightarrow cC \mid d$

Pour la détermination de la collection d'ensembles d'items LALR(1), on recherche les ensembles d'items LR(1) pouvant être fusionnés, on en trouve trois paires : I_3 avec I_6, I_4 avec I_7 et I_8 avec I_9 qui seront remplacés par leurs unions respectives :

I_{36}=C→c.C, c/d/$
C→.cC, c/d/$
C→.d, c/d/$

I_{47}= C→d., c/d/$

I_{89}= C→cC., c/d/$
Les autres ensembles d'items (non concernés par les fusions) restent inchangés.

La construction de la table d'analyse LALR est obtenue par superposition des lignes correspondant aux états fusionnés:

Etat	**Action**			**Successeur**	
	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	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		

5.5.3 Analyse de chaînes par la méthode LALR(1)

Si la chaîne analysée est correcte syntaxiquement (appartient au langage), l'analyse LALR(1) progressera de la même manière que LR(1). Les seules différences sont dans l'appellation ou la numérotation des états de transition.

Si la chaîne analysée est erronée syntaxiquement (n'appartient pas au langage), l'erreur sera détectée plus rapidement par l'analyseur LR(1) alors que l'analyseur LALR(1) peut effectuer plusieurs réductions avant de signaler l'erreur.

Ces deux cas sont illustrés à travers la réponse aux questions 4 et 5 de l'exercice 6.