élimination de la récursivite à gauche.

La récursivité à gauche apparait dans les règles de la forme A ---> A Pour l'éliminer, il faut substituer aux règles

A ---> Aα
A ---> β

les règles:

A ---> βA'

A'---> αA' / ε

exemple:

La grammaire G de la figure 3.1 après élimination de la récursivité à gauche devient G' qui est:

E ---> TE'

E' ---> +TE'
/

T ---> FT'

T' ---> *FT'
/

F ---> (E)
/ id

Figure 3.2: Grammaire G'.

Pour satisfaire la condition 2 d'une grammaire LL(1), nous devons factoriser toutes les règles de la grammaire. Cette opération revient à substituer aux règles de la forme :

A---> α β
A---> α γ

les règles:

A--->αA'
A'---> β / γ

Et de façon générale, il faut remplacer les règles de la forme:

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

par les règles:

$$A \longrightarrow \alpha A'$$
$$A' \longrightarrow \beta 1/\beta 2/..../\beta n.$$

exemple:

La grammaire G' de la figure 3.2 est déjà factorisée, de même est la grammaire de la figure 3.1.

3.2.1.4 Algorithme D'analyse. -

Nous utilisons une pile comportant les symboles de la grammaire et un vecteur contenant la chaine à analyser terminée par le symbole #.

Initialement la configuration de l'analyse est:

```
  (#s          ,a1 a2...am#)
  -------   ----------------
contenu de     vecteur contenant la
la pile        chaine à analyser.
```

S étant l'axiome de la grammaire.

Avant d'analyser une chaine, il faut convertir la grammaire décrivant le langage qui accepte la chaine à analyser en une grammaire LL(1) et ensuite construire la table prédictive de cette dernière.

algorithme:

1-soient X l'élément au sommet de la pile et ai le symbole courant de la chaine à analyser.

a- si X est un terminal alors

-si X = ai alors dépiler X, avancer dans la chaine et aller à 1.

-sinon si X = '#' et ai = '#' alors la chaine est acceptée. stop.

-sinon si X <> ai alors la chaine est erronnée.

b- si X est un nonterminal alors soit (X--->α) la règle se trouvant dans la case (X, ai)

-dépiler X, empiler α la partie droite de la régle et aller à 1.

exemple :

La table prédictive de G' est:

	+	*	(	)	id	#
E			E-> TE'		E->TE'	
T			T->FT'		T->FT'	
F			F->(E)		F->id	
E'	E->+TE'			E'->ε		E'->ε
T'	T'->ε	T'->*FT'		T'->ε		T'->ε

analysons la chaine id1 + id2*id3:

```
(#E              ,          id1+id2*id3#)
(#E'T            ,          id1+id2*id3#)
(#E'T'F          ,          id1+id2*id3#)
(#E'T'id         ,          id1+id2*id3#)
(#E'T'           ,             +id2*id3#)
(#E'             ,           + id2*id3#)
(#E'T+           ,          +id2 * id3# )
(#E'T            ,              id2*id3#)
(#E'T'F          ,              id2*id3#)
(#E'T'id         ,              id2*id3#)
(#E'T'           ,                * id3#)
(#E'T'F*         ,                *id3#)
(#E'T'F          ,                 id3#)
(#E'T'id         ,                  id#)
(#E'T'           ,                    #)
(#E'             ,                    #)
(#               ,                    #)
  o.k
```

L'arbre syntaxique produit par l'analyse de la chaine id1+id2*id3 est montré sur la figure 3.3.

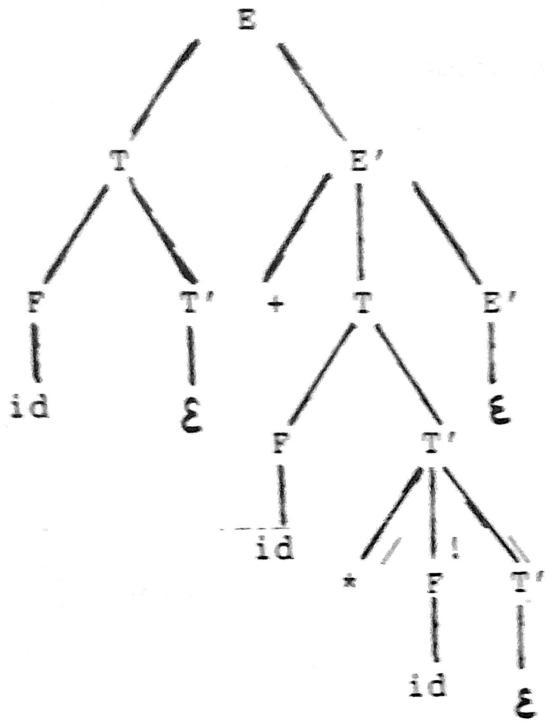

Figure 3.3.

3.2.2 Analyse Descente Récursive.

C'est une analyse simple à implémenter. Il s'agit d'écrire une procédure éventuellement récursive pour chaque règle de la grammaire. Pour éviter les retours arrière(backtracking), il faut d'abord transformer la grammaire donnée en une grammaire LL(1).

exemple:

Soit G'la grammaire de la figure 3.2, l'analyseur descent récursif pour G' consiste en l'ensemble des procédures écrites ci-dessous.

```
procedure E( )
begin
      T( );
```

```
          E'( );
    end;

   procedure E'()
   begin
     if entcour = '+' then
     begin  lex( );
            T( );
            E'();
     end
  end;

procedure T()
 begin
      F( );
      T'( );
 end;

procedure T'()
 begin
  if entcour = '*' then
    begin  lex ( );
           F( );
           T'( );
    end
 end;

procedure F( );

 begin if entcour ='(' then
       begin lex( );
             E( );
             if entcour=')' then lex( )
                            else erreur
       end
       else if entcour = id then lex( )
            else erreur
 end
```

Dans ces procédures, nous utilisons la fonction lex( ) qui consiste à fournir la prochaine entité lexicale de la chaine à analyser. Cela revient à se pointer vers l'entité

lexicale suivante.

Entcour est une variable globale désignant l'entité lexicale courante.

exemple:

L'analyse de la chaine

id1+id2*id3

utilisant la grammaire G'de la figure 3.2 conduit à la construction de l'arbre d'analyse de la figure 3.3.

3.3 ANALYSE ASCENDANTE.

Nous étudions ici trois méthodes d'analyse ascendante qui sont la SLR(1), la LR(1) et la LALR(1). A travers cette étude, nous verrons les modes d'application de chacune de ces méthodes.

Toutes les trois utilisent un programme qui simule un automate représenté par une table. Ce programme appelé conducteur(ou introducteur)est pratiquement identique dans les trois cas. Ce qui différencie les trois analyses, c'est le contenu de la table d'analyse qui particularise le type d'analyse.

Le schéma suivant explique le fonctionnement de ces analyses.