Analyse Syntaxique

Chapitre exrait du livre : Compilation Cours et exercices

Auteur : H. DRIAS

Edition : OPU

Le rôle principal de l'analyse syntaxique est de vérifier si l'écriture du programme source est conforme avec la syntaxe du langage à compiler. Cette dernière est spécifiée à l'aide d'une grammaire.

Il existe plusieurs méthodes d'analyse appartenant à l'une des deux catégories d'analyse 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. il existe actuellement plusieurs méthodes qui parmi elles, nous distinguons les méthodes non deterministes ou avec retour arrière et les méthodes déterministes telles que l'analyse prédictive.

D'une façon opposée, l'analyse ascendante établit des réductions sur la chaine à analyser pour aboutir à l'axiome de la grammaire. Comme pour l'analyse descendante, il existe des méthodes non déterministes peu efficaces et des méthodes déterministes. Dans toute méthode, un arbre syntaxique se construit soit explicitement soit implicitement.

Dans ce chapitre, nous effectuerons un tour d'horizon sur quelques méthodes d'analyse efficaces déterministes. Nous essayerons à la fin de fournir un aperçu sur le fonctionnement d'un générateur d'analyseur syntaxique nommé YACC.

3.1 RAPPELS.

Les notions de dérivation et ambiguité seront constamment utilisés dans ce chapitre.

Soit la grammaire G donnée par:

```
E  --->  E + E
       / E * E
       / a
```

dérivation.

E dérive la chaine a * a + a et nous notons ceci par

$$E \overset{*}{===>} a * a + a$$

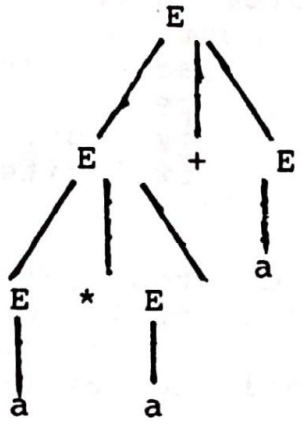
car il existe une suite de dérivations qui sont:

E ===> E + E ===> E * E + E ===> a * E + E ===> a * a + E ===> a * a + a

aboutissant à la chaine a * a + a

ambiguité.

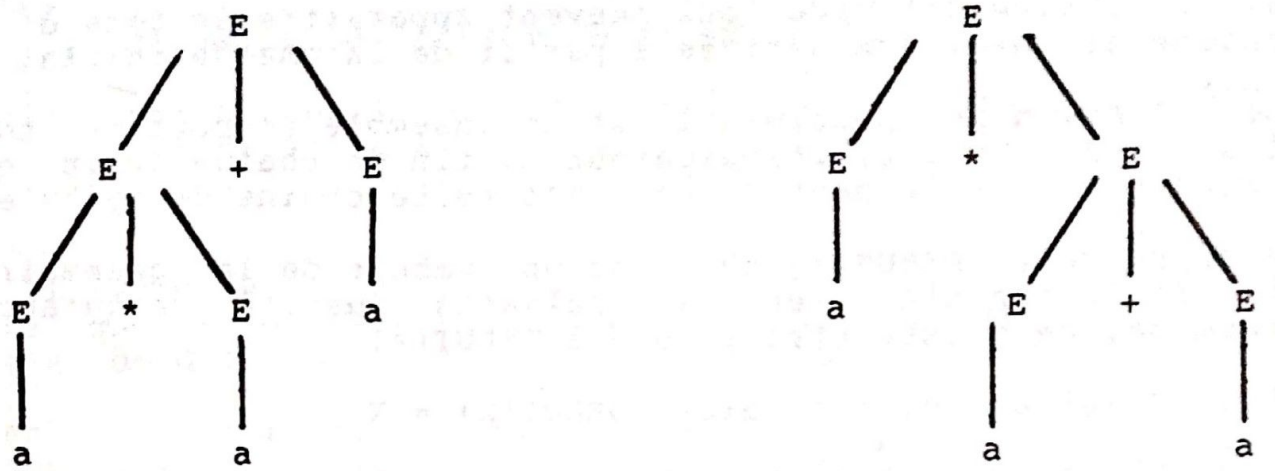
L'arbre



Schématise la dérivation précédente.

Une grammaire est dite ambigue si pour une chaine donnée engendrée par cette grammaire, il existe plusieurs arbres de dérivation distincts.

A la chaine a * a + a peuvent correspondre plusieurs arbres de dérivation parmi lesquels nous montrons les deux arbres suivants:



La grammaire G est donc ambigue.

3.2 ANALYSE DESCENDANTE.

Les méthodes déterministes les plus efficaces connues à l'heure actuelle sont: l'analyse prédictive et la descente récursive. L'une et l'autre sont en quelque sorte équivalentes dans le sens où elles effectuent exactement les mêmes dérivations et par conséquent construisent le même arbre syntaxique pour une même chaine acceptée par la grammaire. Lorsque le langage d'implémentation dispose de la récursivité, il est souhaitable et plus adéquat d'utiliser la descente recursive. Dans le cas contraire, l'analyse prédictive affirmera son existence.

3.2.1 Analyse Predictive.

Cette analyse utilise une table appelée table prédictive. les lignes de cette table sont indicées par les symboles nonterminaux du langage, par contre, les colonnes de la table sont elles indicées par les symboles terminaux et le symbole '#'.

Les cases de la table contiennent éventuellement des règles de la grammaire. Pour la construction de cette table, deux notions qui sont DEBUT et SUIVANT sont nécessaires. Intuitivement le début d'une chaine de symboles de la grammaire contient tous les terminaux (et éventuellement 'ε'

si la chaine est vide)qui peuvent apparaitre en tête d'une chaine de terminaux dérivée à partir de la chaine initiale.

Le SUIVANT d'un nonterminal est un ensemble comportant tous les terminaux (#, l'indicateur de fin de chaine inclu) qui peuvent suivre ce nonterminal dans toute chaine de symboles.

Pour calculer DEBUT(X) où X est un symbole de la grammaire, il faut éxécuter les pas suivants jusqu'à ce qu'aucun terminal ne puisse être ajouté à DEBUT(X)

1-si X est un terminal alors DEBUT(X) = X

2-si X est un nonterminal et si X ---> aα est une règle de la grammaire, a etant un terminal alors ajouter a à DEBUT(X).
Si X --->ε est une règle de la grammaire alors ajouter ε à DEBUT(X).

3-si X---->y1y2...ym est une règle de grammaire et si pour j=1,...,i-1 DEBUT(yj)contient ε alors ajouter tous les éléments différents de ε qui appartiennent à DEBUT(yi) à DEBUT(X).
Si pour j=1,...,m DEBUT(yj) contient ε alors ajouter ε à DEBUT(X).

Pour déterminer SUIVANT(X) où X est un nonterminal, suivre les pas suivants jusqu'à ce qu'aucun élément ne puisse être ajouté à SUIVANT(X).

1-si X est l'axiome de la grammaire alors ajouter #(marqueur de fin de chaine) à SUIVANT(X).

2-si A ---> XaB appartient à la grammaire et si a est un terminal alors ajouter a à SUIVANT(X)

3-si A --->αX est une règle de la grammaire alors ajouter tous les suivants de A à SUIVANT(X).

3.2.1.1 Algorithme De Construction De La Table Predictive.

Pour chaque règle A ---> α de la grammaire où α <> ε faire
pour chaque élément terminal a appartenant à DEBUT(α)

faire
placer dans la case (A,a) la règle A ---> α.

Pour chaque règle de la grammaire de la forme A ---> ε faire
pour chaque terminal x dans SUIVANT(A) faire
placer dans la case (A,x) la règle A ---> ε.

Si la table, une fois construite est multidéfinie c'est à dire que si une case de la table est occupée par plus d'une règle, on dit que la grammaire n'est pas LL(1).

Dans le cas contraire c'est à dire lorsque dans toutes les cases de la table, il existe au plus une règle, on dit que la grammaire est LL(1).

exemple

Soit G la grammaire suivante:

E ----> E + T
/ T

T ----> T * F
/ F

F ----> (E)
/ id

Figure 3.1: Grammaire des expressions arithmétiques.

Le DEBUT et le SUIVANT de chaque nonterminal sont montrés dans la table ci-dessous.

	DEBUT	SUIVANT
E	(id	# +)
T	(id	# +) *
F	(id	# +) *

La table prédictive de G est donc la suivante:

	+	*	(	)	id	#
E			E-->E+T E-->T		E-->T	
T			T-->T*F T-->F		T-->F	
F			F-->(E)		F-->id	

Cette table est multidéfinie. Par conséquent G n'est pas LL(1).

3.2.1.2 Definition Formelle D'une Grammaire LL(1). -

Une grammaire G est dite LL(1) si ses règles de production vérifient les conditions suivantes:

1-si A ---> α et A ---> β sont deux règles de G alors on doit avoir ou bien

$\alpha \overset{*}{\Longrightarrow} \varepsilon$ ou bien $\beta \overset{*}{\Longrightarrow} \varepsilon$ mais pas les deux.

2-si A ---> α et A ---> β sont deux règles de G alors

$$DEBUT(\alpha) \cap DEBUT(\beta) = \emptyset$$

3-si A--->α et A--->β sont deux règles de G et si $\beta \overset{*}{\Longrightarrow} \varepsilon$ alors SUIVANT (A) $\cap$ DEBUT(α) = $\emptyset$.

Ces conditions entrainent l'unicité de l'existence d'une règle dans une case de la table prédictive.

3.2.1.3 Conditions Nécéssaires Pour Qu'une Grammaire Soit LL(1). -

Pour q'une grammaire soit LL(1), il faut que toutes ses règles soient non récursives à gauche et factorisées.

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:

$$A \longrightarrow \alpha\beta1/\alpha\beta2/..../\alpha\beta n$$

par les règles:

$$A \longrightarrow \alpha A'$$
$$A' \longrightarrow \beta1/\beta2/..../\beta n.$$

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*ïd3#)
(#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.

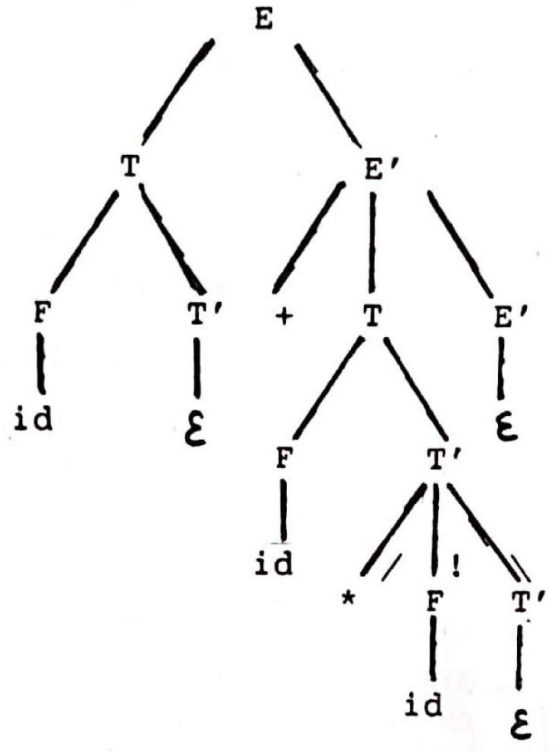


Figure 3.3.