Analyse syntaxique (descendante) (2)

Construction de la table d'analyse LL

Soit $G = (N, T, \rightarrow, S)$ une grammaire, une table d'analyse M pour la grammaire G est une table à deux entrées.
Pour chaque élément A de N et chaque élément a de $T \cup \{\$\}$, $M[A,\ a]$ indique la règle de la grammaire à appliquer.

Construction de la table d'analyse

- Pour chaque règle $A \rightarrow \alpha$ de la grammaire faire
  1. pour tout $a \in Premier(\alpha)$, ajouter la règle $A \rightarrow \alpha$ dans la case $M[A,\ a]$
  2. si $\alpha \overset{*}{\rightarrowtail} \varepsilon$ alors pour tout $b \in Suivant(A)$ ajouter la règle $A \rightarrow \alpha$ dans la case $M[A,\ b]$
- chaque case vide de M correspond à une erreur de syntaxe

Exemple de construction d'une table d'analyse

Example

$G = (\{E,\ T,\ F,\ E',\ T'\},\ \{+,\ -,\ *,\ (,\ ),\ nb\ \},\ \},\ \rightarrow,\ E)$ la grammaire définie par

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid nb$

- $Vide = \{E',\ T'\ \}$
- $Premier(E) = \{(,\ nb\}$, $Premier(E') = \{+\}$, $Premier(T) = \{(,\ nb\}$, $Premier(T') = \{*\}$ et $Premier(F) = \{(,\ nb\}$
- $Suivant(E) = \{\$,\ )\}$, $Suivant(E') = \{\$,\ )\}$, $Suivant(T) = \{+,\ \$,\ )\}$, $Suivant(T') = \{+,\ \$,\ )\}$, $Suivant(F) = \{*,\ +,\ \$,\ )\}$

Table d'analyse LL

	nb	$+$	$*$	$($	$)$	$\$$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow nb$			$F \rightarrow (E)$		

Grammaire LL(1)

Definition

On appelle grammaire LL(1) une grammaire pour laquelle toutes les cases de la table d'analyse contiennent au plus une règle de la grammaire.

Dans l'abréviation LL(1) :

- le premier L signifie que le mot donné en entrée est lu de gauche à droite (Left to right scanning)
- le second L signifie que la méthode recherche une dérivation à gauche, c'est à dire que l'on cherche à réécrire le non terminal le plus à gauche (Leftmost derivation)
- le 1 signifie qu'en lisant 1 caractère à l'avance on est capable de dire quelle règle utiliser

Example

La grammaire du transparent précédent définissant les expressions arithmétiques est une grammaire LL(1).

Symboles directeurs d'une règle

On utilise parfois la notion de symbole directeur d'une règle.

Definition (Symbole directeur d'une règle)

L'ensemble des symboles directeurs de $A \longrightarrow \alpha$ est défini par

$SymbolesDirecteurs(A \longrightarrow \alpha) =$
si $non(\alpha \overset{*}{\rightarrowtail} \varepsilon)$
alors $Premier(\alpha)$
sinon $Premier(\alpha) \cup Suivant(A)$

- lors de la construction de la table d'analyse M : si $d \in SymbolesDirecteurs(A \longrightarrow \alpha)$ alors on ajoute $A \longrightarrow \alpha$ dans $M[A,\ d]$.
- pour que la grammaire soit LL(1) il faut que pour deux règles différentes $A \longrightarrow \alpha$ et $A \longrightarrow \beta$ on ait $SymbolesDirecteurs(A \longrightarrow \alpha) \cap SymbolesDirecteurs(A \longrightarrow \beta) = \emptyset$

Analyseur syntaxique

On suppose que la grammaire est LL(1) et que M est la table d'analyse de la grammaire.
On utilise une pile, à l'initialisation la pile contient $\$S$ (le sommet de la pile est S l'axiome de la grammaire).
Variables du programme :

- X : le symbole en sommet de pile
- a : le symbole (terminal) courant du mot analysé
- erreur : booléen (initialisé à faux)
- accepter : booléen (initialisé à faux)

Analyseur syntaxique : algorithme

répeter
  $X \leftarrow$ sommet de pile
  $a \leftarrow$ caractére du mot à analyser
  si X est un non terminal alors
    si $M[X, a] = X \rightarrow Y_1 \ldots Y_n$
      émettre en sortie la règle $X \rightarrow Y_1 \ldots Y_n$
      dépiler X de la pile;
      empiler $Y_n, \ldots, Y_1$;
    sinon *erreur* $\leftarrow$ *vrai* – la case est vide dans la table
    fsi
  sinon – X est un terminal
     si $X =$ \$ alors – la pile est vide
      si $a =$ \$ alors *accepter* $\leftarrow$ *vrai* – pile vide et caractère de fin de mot
      sinon *erreur* $\leftarrow$ *vrai*
      fsi
    sinon – la pile n'est pas vide
      si $X = a$ alors – l'élément en sommet de pile est le caractère courant du mot
        depiler X;
        lire le caractère suivant du mot donné;
      sinon – l'élément en sommet de pile est différent du caractère courant du mot
        erreur $\leftarrow$ *vrai*
      fsi
    fsi
  fsi

jusqu'à erreur ou accepter

Exemple d'exécution de l'analyseur syntaxique

Grammaire des expressions arithmétiques, mot : $2 + 5 * 7$

PILE	*Entrée*	*Sortie*
$\$E$	$2+5*7\$$	$E \rightarrow TE'$
$\$E'T$	$2+5*7\$$	$T \rightarrow FT'$
$\$E'T'F$	$2+5*7\$$	$F \rightarrow 2$
$\$E'T'2$	$2+5*7\$$	
$\$E'T'$	$+5*7\$$	$T' \rightarrow \varepsilon$
$\$E'$	$+5*7\$$	$E' \rightarrow +TE'$
$\$E'T+$	$+5*7\$$	
$\$E'T$	$5*7\$$	$T \rightarrow FT'$
$\$E'T'F$	$5*7\$$	$F \rightarrow 5$
$\$E'T'5$	$5*7\$$	
$\$E'T'$	$*7\$$	$T' \rightarrow *FT'$
$\$E'T'F*$	$*7\$$	
$\$E'T'F$	$7\$$	$F \rightarrow 7$
$\$E'T'7$	$7\$$	
$\$E'T'$	$\$$	$T' \rightarrow \varepsilon$
$\$E'$	$\$$	$E' \rightarrow \varepsilon$
$\$$	$\$$	*accepter* (*le mot est engendré par la grammaire*)

Propriétés des grammaires LL(1)

Proposition

Une grammaire G, LL(1), possède les propriétés suivantes :

- s'il existe deux règles $A \longrightarrow \alpha$ et $A \longrightarrow \beta$ de G :
  1. $\mathit{Premier}(\alpha) \cap \mathit{Premier}(\beta) = \emptyset$
  2. au plus un de α ou β vérifie $\alpha \overset{*}{\rightarrowtail} \varepsilon$ ou $\beta \overset{*}{\rightarrowtail} \varepsilon$
  3. si $\beta \overset{*}{\rightarrowtail} \varepsilon$ alors $\mathit{Premier}(\alpha) \cap \mathit{Suivant}(A) = \emptyset$
- la grammaire G n'est pas ambiguë.
- la grammaire G ne comporte pas de récursivité à gauche (l'algorithme ne termine pas pour des grammaires comportant une récursivité gauche).

Récusrivité à gauche

Definition (Récursivité gauche immédiate)

Une grammaire est **immédiatement récursive à gauche** s'il existe un non terminal A et une règle de la forme $A \longrightarrow A\alpha$ où α est une chaîne de terminaux ou non terminaux quelconques.

Example

La grammaire suivante comporte plusieurs récursivité gauches immédiates :

$S \longrightarrow ScA \mid B$
$A \longrightarrow Aa \mid \varepsilon$
$B \longrightarrow Bb \mid d \mid e$

Suppression de la récursivité à gauche immédiate

Proposition

On remplace des règles de la forme

$$A \longrightarrow A\alpha_1 \mid \ldots \mid A\alpha_k \mid \beta_1 \mid \ldots \mid \beta_p$$

par les règles suivantes :

$$\begin{array}{l} A \longrightarrow \beta_1 A' \mid \ldots \mid \beta_p A' \\ A' \longrightarrow \alpha_1 A' \mid \ldots \mid \alpha_k A' \mid \varepsilon \end{array}$$

Démonstration : cela provient du fait que l'on a $A \overset{*}{\rightarrowtail} \{\beta_1, \ldots, \beta_k\}\{\alpha_1, \ldots, \alpha_k\}^*$

Exemple de suppression de la récursivité à gauche immédiate

Example

$S \longrightarrow ScA \mid B$
$A \longrightarrow Aa \mid \varepsilon$
$B \longrightarrow Bb \mid d \mid e$

$S \longrightarrow BS'$
$S' \longrightarrow cAS' \mid \varepsilon$
$A \longrightarrow A'$
$A' \longrightarrow aA' \mid \varepsilon$
$B \longrightarrow dB' \mid eB'$
$B' \longrightarrow bB' \mid \varepsilon$

Récursivité à gauche

Definition (Récursivité à gauche)

Une grammaire est récursive à gauche s'il existe un non terminal A et une dérivation de la forme $A \overset{*}{\rightarrowtail} A\alpha$.

Example

$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid c$
Le non terminal S est récursif à gauche car $S \rightarrowtail Aa \rightarrowtail Sda$ (mais S n'est pas immédiatement récursif à gauche).

Suppression de la récursivité à gauche pour des grammaires sans règle $A \rightarrow \varepsilon$ et sans cycle

Hypothèses : on suppose que la grammaire donnée ne possède pas de règles $A \rightarrow \varepsilon$ ni de cycle (i.e. il n'existe pas A tel que $A \overset{+}{\rightarrowtail} A$

Ordonner les non terminaux de la grammaire $A_1, \ldots, A_n$
pour $i = 1$ à n faire
  pour $j = 1$ à i-1 faire
    remplacer chaque règle de la forme $A_i \rightarrow A_j\alpha$ où $A_j \rightarrow \beta_1 \mid \ldots \mid \beta_p$
    par $A_i \rightarrow \beta_1\alpha \mid \ldots \mid \beta_p\alpha$
  fpour
  éliminer les récursivités à gauche immédiates des règles dont les membres gauches sont A_i

Exemple de suppression de récursivité

Example

$S \longrightarrow Aa \mid b$
$A \longrightarrow Ac \mid Sd \mid c$
On ordonne les non terminaux : $A_1 = S$, $A_2 = A$

1. $i = 1$ (la boucle interne est vide et il n'y a pas de récursivité immédiate de S)
   $S \longrightarrow Aa \mid b$
2. $i = 2$
   on remplace $A \longrightarrow Sd$ par $A \longrightarrow Aad \mid bd$,
   on obtient ainsi $A \longrightarrow Ac \mid Aad \mid bd \mid c$
   On élimine la récursivité immédiate à gauche de A
   $A \longrightarrow bdA' \mid cA'$
   $A' \longrightarrow cA' \mid adA' \mid \varepsilon$

Le résultat est la grammaire suivante :
$S \longrightarrow Aa \mid b$
$A \longrightarrow bdA' \mid cA'$
$A' \longrightarrow cA' \mid adA' \mid \varepsilon$

Contre-exemple de suppression de la récursivité à gauche

Example

$S \longrightarrow Sa \mid TSc \mid d$
$T \longrightarrow SbT \mid \varepsilon$
On ordonne les non terminaux : $A_1 = S$, $A_2 = T$,

1. $i = 1$
   On supprime la récursivité immédiate à gauche de S
   $S \longrightarrow TScS' \mid dS'$
   $S' \longrightarrow aS' \mid \varepsilon$
2. $i = 2$
   On remplace $T \longrightarrow SbT$ par
   $T \longrightarrow TScS'bT \mid dS'bT$
   On obtient donc $T \longrightarrow TScS'bT \mid dS'bT \mid \varepsilon$
   On élimine la récursivité immédiate à gauche de T
   $T \longrightarrow dS'bTT' \mid T'$
   $T' \longrightarrow ScS'bTT' \mid \varepsilon$

On obtient la grammaire suivante :
$S \longrightarrow TScS' \mid dS'$
$S' \longrightarrow aS' \mid \varepsilon$
$T \longrightarrow dS'bTT' \mid T'$
$T' \longrightarrow ScS'bTT' \mid \varepsilon$
On n'a pas supprimé la récursivité gauche car
$S \rightarrowtail TScS' \rightarrowtail T'ScS' \rightarrowtail ScS'$

Factorisation à gauche

Lorsque deux règles sont de la forme $A \longrightarrow \alpha\beta_1$ ou $A \longrightarrow \alpha\beta_2$, il n'est en général pas possible de déterminer quelle règle on va utiliser en lisant un caractère à l'avance. L'idée est donc de différer la décision jusqu'à ce qu'on ait lu suffisamment de texte.

Example

$S \longrightarrow abcdAab \mid abcdBbd$
$A \longrightarrow aC$
$B \longrightarrow bD$
$C \longrightarrow \ldots$
$D \longrightarrow \ldots$

Il faut lire le 5ième caractère à l'avance pour déterminer quelle règle choisir entre $S \longrightarrow abcdAab$ et $S \longrightarrow abcdBbd$, la grammaire n'est pas LL(1).

Factorisation : algorithme

Algorithme

pour chaque non terminal A
trouver le plus long préfixe commun α à deux ou plus de deux membres droits de règles
si $\alpha \neq \varepsilon$, remplacer $A \rightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \ldots \mid \gamma_p$ (où α n'est pas préfixe de γ_i) par
$A \rightarrow \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_p$
$A' \rightarrow \beta_1 \mid \ldots \mid \beta_n$
Recommencer jusqu'à ce qu'il n'y ait plus de factorisation à effectuer

Example

$S \rightarrow aAbS \mid aAbSeB \mid a$
$A \rightarrow bcB \mid bca$
$B \rightarrow ab$

Exécution de l'algorithme

- pour S
  - le plus long préfixe est $aAbS$, on obtient
    $S \rightarrow aAbSS' \mid a$
    $S' \rightarrow eB \mid \varepsilon$
  - le plus long préfixe est a, on obtient
    $S \rightarrow aS''$
    $S'' \rightarrow AbSS' \mid \varepsilon$
- pour A le plus long préfixe est bc, on obtient
  $A \rightarrow bcA'$
  $A' \rightarrow B \mid a$

Factorisation

La grammaire obtenue par factorisation est :

$S \longrightarrow aS''$
$S'' \longrightarrow AbSS' \mid \varepsilon$
$S' \longrightarrow eB \mid \varepsilon$
$A \longrightarrow bcA'$
$A' \longrightarrow B \mid a$
$B \longrightarrow ab$

Conclusion

Etant donnée une grammaire

- Réduire la grammaire (en éliminant les règles et les symboles inutiles)
- Rendre la grammaire non ambiguë si nécessaire (il n'y a pas d'algorithme pour déterminer si une grammaire est ambiguë et pour en trouver une non ambiguë)
- Eliminer la récursivité à gauche si nécessaire
- Factoriser la grammaire si nécessaire
- Construire la table d'analyse (aprés avoir calculé *Premier* et *Suivant*)

Si la grammaire n'est pas LL(1), utiliser une autre méthode pour l'analyse syntaxique.

Il existe des méthodes qui généralisent LL(1). On peut caractériser les grammaires LL(k) en généralisant les définitions de *Premier* et *Suivant*.

Les méthodes d'analyse ascendante seront vues en deuxième année, dans le module de Traduction.