Analyse syntaxique descendante

TELECOM Nancy (1A)

Mathématiques Appliquées pour l'Informatique

2019-2020

Plan

Réécriture gauche, dérivation gauche

Définition (Réécriture à gauche)

La relation de réécriture à gauche est définie sur $(N \cup T)^*$ par "α se réécrit à gauche en β", noté par $\alpha \rightarrowtail \beta$, si $\alpha = wA\alpha_1$ et $\beta = w\gamma\alpha_1$ où $\alpha_1,\ \gamma \in (N \cup T)^*$, $w \in T^*$, $A \in N$ et $A \to \gamma$ est une règle de G.

Remarque : la réécriture à gauche consiste à réécrire le non terminal le plus à gauche.

Définition (Dérivation à gauche)

La relation de dérivation à gauche est la fermeture réflexive transitive de la relation de réécriture à gauche.

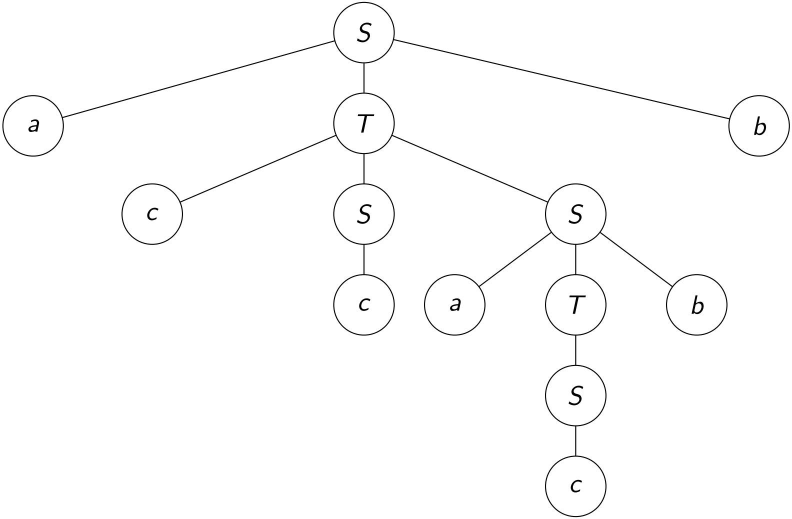
Exemples de dérivations et d'arbres syntaxiques

Soit la grammaire $G = (\{S,\ T\},\ \{a,\ b,\ c\},\ \rightarrow,\ S)$, définie par les règles $\begin{array}{l} S \rightarrow aTb \mid c \\ T \rightarrow cSS \mid S \end{array}$

- $S \rightarrowtail aTb \rightarrowtail acSSb \rightarrowtail accSb \rightarrowtail accaTbb \rightarrowtail accaSbb \rightarrowtail accacbb$ est une dérivation à gauche (à chaque étape le non terminal le plus à gauche est réécrit)
- $S \rightarrowtail aTb \rightarrowtail acSSb \rightarrowtail acSaTbb \rightarrowtail accaTbb \rightarrowtail accaSbb \rightarrowtail accacbb$
  n'est pas une dérivation à gauche.

le mot *accacbb* a plusieurs dérivations différentes et un seul arbre syntaxique.

Arbre syntaxique du mot *accacbb*



Données : une grammaire $G = (N,\ T,\ \rightarrow,\ X)$ et α un mot de T^*
Problème : trouver un algorithme qui détermine si $\alpha \in L(G)$ et si c'est le cas construit son arbre syntaxique

- La construction de l'arbre syntaxique de α se fait en préordre, c'est-à-dire de la racine (l'axiome) vers les feuilles (nœuds étiquetés par les terminaux) (on parle d'analyse descendante)
- La construction de l'arbre revient à trouver une dérivation à gauche de α
- On définit une classe de grammaires pour lesquelles il existe des algorithmes simples d'analyse syntaxique (lecture d'un seul caractère à l'avance dans le mot à analyser et pas de retours arrières (les grammaires LL(1))

Analyse descendante : exemples introductifs

On tente de construire un arbre syntaxique du mot donné de haut en bas, c'est-à-dire en partant de l'axiome de la grammaire et en parcourant le mot donné de gauche à droite.

Exemple (1)

$G = (\{S,\ T\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ une grammaire définie par

$S \rightarrow aSbT \mid cT \mid d$

$T \rightarrow aT \mid bS \mid c$

on considère le mot $w = accbbadbc$

Exemple (2)

$G = (\{S,\ A\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ une grammaire définie par

$S \rightarrow aAb$

$A \rightarrow cd \mid c$

on considère le mot $w = acb$

Exemple (3)

$G = (\{S\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ une grammaire définie par

$S \rightarrow aSb \mid aSc \mid d$

on considère le mot $w = aaaadbcbcb$

A chaque étape, on dispose du non terminal à réécrire, et du terminal lu dans le mot à reconnaître et l'on doit trouver quelle règle utiliser pour réécrire le non terminal. A l'initialisation le non terminal est l'axiome de la grammaire et le terminal est la première lettre du mot à reconnaître.

- Exemple 1. Mot à reconnaître : *accbbadbc*

non terminal à réécrire	terminal courant	règle utilisée
S	a	$S \rightarrow aSbT$
S	c	$S \rightarrow cT$
T	c	$T \rightarrow c$
T	b	$T \rightarrow bS$
S	a	$S \rightarrow aSbT$
S	d	$S \rightarrow d$
T	c	$T \rightarrow c$

A chaque étape on détermine exactement la règle à utiliser.

- Exemple 2. Mot à reconnaître : acb

non terminal à réécrire	terminal courant	règle utilisée
S	a	$S \rightarrow aAb$
A	c	deux choix possibles: $A \rightarrow cd$ ou $A \rightarrow c$

  Deux choix sont a priori possibles, mais c'est la règle $A \rightarrow c$ qui convient.

- Exemple 3. Mot à analyser $aaaaadbcbcb$.
  A l'initialisation on a $(S,\ a)$, a priori les deux règles $S \rightarrow aSb$ et $S \rightarrow aSc$ peuvent convenir pour réécrire S...
  on remarque que la première lettre a du mot est appariée avec b la dernière lettre du mot,
  il n'est pas possible de déterminer la règle à utiliser en lisant seulement une lettre à l'avance.

Analyse descendante : exemples introductifs (fin)

Exemple de grammaire décrivant les expressions arithmétiques, qui illustre la suite du cours.

- Exemple (4)
  $G = (\{E,\ T,\ F,\ E',\ T'\},\ \{+,\ -,\ *,\ (,\ ),\ nb\ \},\ \ \},\ \ \rightarrow,\ E)$ une grammaire définie par
  $E \rightarrow TE'$
  $E' \rightarrow +TE' \mid -\ TE' \mid \varepsilon$
  $T \rightarrow FT'$
  $T' \rightarrow *FT' \mid /FT' \mid \varepsilon$
  $F \rightarrow (E) \mid nb$
  on considère le mot $w = 3 * 4 + 10 * (5 + 11)/34 + 12$

Non terminaux produisant le mot vide ε

Définition

Soit $G = (N,\ T,\ \rightarrow,\ S)$ une grammaire algébrique et $X \in N$,
$P_\epsilon(G) = \{X\ ;\ X \in N \textit{ et } X \overset{*}{\rightarrowtail} \varepsilon\}$.
$P_\epsilon(G)$ est l'ensemble des terminaux qui produisent le mot ε.

Algorithme de calcul de l'ensemble des producteurs du mot vide $P_\epsilon(G)$

On définit la suite $\mathcal{U}$ d'ensembles d'éléments de N par :

- $\mathcal{U}_0 = \emptyset$
- $\mathcal{U}_{n+1} = \mathcal{U}_n \cup \{A,\ A \in N \textit{ et } \exists R = A \rightarrow \alpha \textit{ règle de } G \textit{ telle que } \alpha \in \mathcal{U}_n{}^*\}$, si $n \geq 0$

$\mathcal{U}$ est une suite croissante majorée par N et comme N est un ensemble fini, $\mathcal{U}$ ne peut prendre qu'un nombre fini de valeurs, elle est donc stationnaire à partir d'un certain rang. La valeur stationnaire de la suite $\mathcal{U}$ est égale à l'ensemble $P_\epsilon(G)$.

Calcul de $P_\epsilon(G)$ (exemple)

$G = (\{S,\ A,\ D,\ E,\ F\},\ \{a,\ c,\ d\},\ \rightarrow,\ S)$ une grammaire définie par

$$\begin{array}{l} S \ \rightarrow AD \mid cS \\ A \rightarrow a \mid EF \\ D \ \rightarrow d \mid AD \\ E \rightarrow a \mid \varepsilon \\ F \ \rightarrow c \mid \varepsilon \end{array}$$

Exécution :

- $\mathcal{U}_0 = \emptyset$
- $\mathcal{U}_1 = \mathcal{U}_0 \cup \{E,\ F\} = \{E,\ F\}$ à cause des règles $E \rightarrow \varepsilon$ et $F \rightarrow \varepsilon$.
- $\mathcal{U}_2 = \mathcal{U}_1 \cup \{A\} = \{E,\ F,\ A\}$ à cause de $A \rightarrow EF$
- $\mathcal{U}_3 = \mathcal{U}_2 = \{E,\ F,\ A\}$

d'où $P_\epsilon(G) = \{E,\ F,\ A\}$

Définition (Premiers)

Soit $G = (N,\ T,\ \rightarrow,\ S)$ une grammaire algébrique et $\alpha \in (N \cup T)^*$,
$Premier(\alpha) = \{x\ ;\ x \in T\ et\ \alpha \overset{*}{\rightarrowtail} xw\}$.

Calcul des premiers pour les non terminaux

1. pour tout non terminal X de la grammaire G, initialiser $Premier(X)$ à l'ensemble vide fpour
2. pour toute règle de la forme $X \rightarrow Y_1 \ldots Y_n$

   si $Y_1 \in T$ alors -ajouter Y_1 à $Premier(X)$
   sinon -ajouter $Premier(Y_1)$ à $Premier(X)$;
   -pour tout $j \in \{2, \ldots, n\}$ tq $\forall i \in \{1, \ldots, j\text{-}1\}$ $Y_i \in P_\epsilon(G)$:
   ajouter $Premier(Y_j)$ à $Premier(X)$
   fpour
   fsi
   fpour
3. recommencer l'étape 2 jusqu'à ce qu'il n'y ait plus de changement

Exemple 1

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

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

on a $P_\epsilon(G) = \{E',\ T'\ \}$

$Premier(E)$	$Premier(E')$	$Premier(T)$	$Premier(T')$	$Premier(F)$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{+,\ -\}$	$\emptyset$	$\{*,\ /\}$	$\{(,\ nb\}$
$\emptyset$	$\{+,\ -\}$	$\{(,\ nb\}$	$\{*,\ /\}$	$\{(,\ nb\}$
$\{(,\ nb\}$	$\{+,\ -\}$	$\{(,\ nb\}$	$\{*,\ /\}$	$\{(,\ nb\}$
idem	*idem*	*idem*	*idem*	*idem*

Exemple 2

$G = (\{S,\ A,\ B,\ C\},\ \{a,\ b,\ c,\ d,\ e\},\ \rightarrow,\ S\})$ une grammaire définie par

$S \rightarrow ABCe$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bB \mid cB \mid \varepsilon$
$C \rightarrow de \mid da \mid dA$

on a $P_\epsilon(G) = \{A,\ B\}$

$Premier(S)$	$Premier(A)$	$Premier(B)$	$Premier(C)$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{a\}$	$\{b,\ c\}$	$\{d\}$
$\{a,\ b,\ c,\ d\}$	$\{a\}$	$\{b,\ c\}$	$\{d\}$
idem	*idem*	*idem*	*idem*

Extension : définition des premiers pour les éléments de $(N \cup T)^*$

Définition

Soit $\alpha \in (N \cup T)^*$, on étend la fonction $Premier(\alpha)$ aux éléments de $(N \cup T)^*$ de la façon suivante :

- $Premier(a\beta) = \{a\}$ si $a \in T$
- $Premier(X)$ est défini précédemment pour $X \in N$
- $Premier(X\beta) =$ si $X \notin P_\epsilon(G)$ alors $Premier(X)$ sinon $Premier(X) \cup Premier(\beta)$ fsi où $\beta \neq \varepsilon$

Définition (Suivants)

Soit la grammaire $G = (N,\ T,\ \rightarrow,\ S)$ et soit $A \in N$, $Suivant(A)$ est l'ensemble des éléments a de $T \cup \{\$\}$ qui peuvent apparaître immédiatement après A dans une dérivation (c'est-à-dire les éléments a tel que $S \overset{*}{\rightarrowtail} \alpha A a \beta$)

Calcul des suivants des symboles non-terminaux

1. initialiser $Suivant(S)$ à $\{\$\}$;
   pour tout non terminal $X \neq S$ de la grammaire G, initialiser $Suivant(X)$ à l'ensemble vide fpour
2. pour chaque règle de la forme $A \rightarrow \alpha B \beta$ où $B \in N$ ajouter $Premier(\beta)$ à $Suivant(B)$
3. pour chaque règle $A \rightarrow \alpha B$, ajouter $Suivant(A)$ à $Suivant(B)$
4. pour chaque règle $A \rightarrow \alpha B \beta$ tel que $\beta \overset{*}{\rightarrowtail} \varepsilon$ ajouter $Suivant(A)$ à $Suivant(B)$

recommencer à partir de l'étape 3 jusqu'à ce que l'on n'ajoute rien de nouveau dans les ensembles *Suivant*

Calcul des suivants

Exemple

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

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

$P_\epsilon(G) = \{E',\ T'\}$

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

$Suivant(E)$	$Suivant(E')$	$Suivant(T)$	$Suivant(T')$	$Suivant(F)$
$\$$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\$\ )$	$\$$	$+\ -\ \$$	$\emptyset$	$*\ /$
$\$\ )$	$\$\ )$	$+\ -\ \$\ )$	$+\ -\ \$$	$*\ /\ +\ -\ \$$
$\$\ )$	$\$\ )$	$+\ -\ \$\ )$	$+\ -\ \$\ )$	$*\ /\ +\ -\ \$\ )$

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 $a \in \mathit{Premier}(\alpha)$, ajouter la règle $A \rightarrow \alpha$ dans la case $M[A,\ a]$
2. si $\alpha \overset{*}{\rightarrowtail} \varepsilon$ alors pour tout $b \in \mathit{Suivant}(A)$ ajouter la règle $A \rightarrow \alpha$ dans la case $M[A,\ b]$

Exemple

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

$$\begin{array}{lll} E \rightarrow TE' & T \rightarrow FT' & F \rightarrow (E) \mid nb \\ E' \rightarrow +TE' \mid \varepsilon & T' \rightarrow *FT' \mid \varepsilon & \end{array}$$

- $P_\epsilon(G) = \{E',\ T'\}$
- $Premier(E) = \{(,\ nb\}$, $Premier(E') = \{+\}$, $Premier(T) = \{(,\ nb\}$, $Premier(T') = \{*\}$ 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)

Définition

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.

Remarque

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

Exemple

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

On utilise aussi la notion de symbole directeur d'une règle, pour construire les tables.

Définition (Symbole directeur d'une règle)

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

$$
\begin{array}{l}
\mathit{SymbolesDirecteurs}(A \to \alpha) = \\
\qquad \text{si } \mathit{non}(\alpha \overset{*}{\rightarrowtail} \varepsilon) \quad \begin{array}{l} \text{alors } \mathit{Premier}(\alpha) \\ \text{sinon } \mathit{Premier}(\alpha) \cup \mathit{Suivant}(A) \end{array}
\end{array}
$$

Remarques

- lors de la construction de la table d'analyse M :
  si $d \in \mathit{SymbolesDirecteurs}(A \to \alpha)$ alors on ajoute $A \to \alpha$ dans $M[A,\ d]$.
- pour que la grammaire soit LL(1) il faut et il suffit que pour tout couple de règles différentes de la forme $A \to \alpha$ et $A \to \beta$ on ait :

$$\mathit{SymbolesDirecteurs}(A \to \alpha) \cap \mathit{SymbolesDirecteurs}(A \to \beta) = \emptyset$$

L'analyseur prend en entrée :

- la table M d'une grammaire $G = (N, \ T, \ \rightarrow, \ S)$ qui est LL(1)
- un mot m de T^* à analyser

L'analyseur détermine si le mot m est engendré par G et si c'est le cas, produit en sortie dans l'ordre les règles ayant permis d'engendrer m (cette séquence de règles permet de construire la dérivation à gauche de m et l'arbre syntaxique de m.
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)

```
répéter
  X ← sommet de pile
  a ← caractére du mot à analyser
  si X est un non terminal alors
    si M[X, a] = X → Y1 ... Yn
      émettre en sortie la règle X → Y1 ... Yn
      dépiler X de la pile;
      empiler Yn, ..., Y1;
    sinon erreur ← 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 ← vrai – pile vide et caractère de fin de mot
      sinon erreur ← 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 ← vrai
      fsi
    fsi
  fsi
jusqu'à erreur ou accepter
```

Exemple d'exécution de l'analyseur syntaxique

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

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*)

Proposition

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

- s'il existe deux règles $A \to \alpha$ et $A \to \beta$ 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).

Représentation des arbres syntaxiques forme postfixée

Définition

Soit $G = (N,\ T,\ \rightarrow,\ S)$ une grammaire algébrique, on ajoute un numéro de règle à chaque règle de la grammaire, on considére les représentations postfixées des arbres en faisant figurer les éléments de T et les numéros des règles.

Exemple

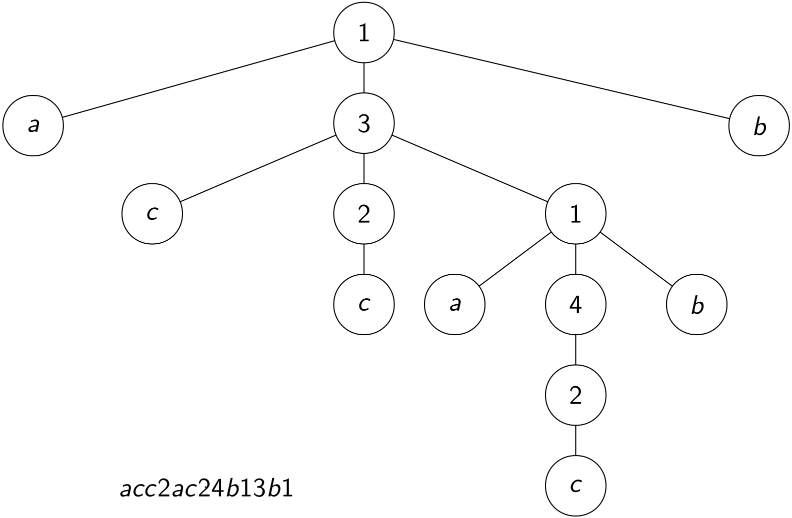
Soit la grammaire $G = (\{S,\ T\},\ \{a,\ b,\ c\},\ \rightarrow,\ S)$, définie par les règles

$S \rightarrow aTb\ |1|\ c|2|$

$T \rightarrow cSS\ |3|\ S\ |4|$

l'arbre syntaxique du mot *accacbb* s'écrit $acc2ac24b13b1$ sous forme postfixée

Arbre du mot *accacbb* avec numéro des règles (forme postfixée)



$acc2ac24b13b1$

- écriture d'un programme "adhoc" pour chaque grammaire LL(1)
- le programme prend en entrée un mot formé de terminaux et détermine si le mot est engendré par la grammaire et si c'est le cas détermine son arbre syntaxique sous forme postfixée
- le programme est structuré en procédure (récursives) de la façon suivante :
  - un programme principal
  - à chaque non terminal X de la grammaire on fait correspondre une procédure, dont la structure est calquée sur les seconds membres des règles dont la partie droite est X (on utilise les cœfficients directeurs des règles)
  - une procédure générique, qui teste si le terminal lu est bien celui attendu
- cette démarche est adaptable :
  - pour générer un arbre abstrait à la place d'un arbre syntaxique
  - aux grammaires $LL(k)$ $(k > 1)$

- Soit $G = (N, \ T, \ \rightarrow, \ S)$ une grammaire

  Données: un mot α de T^* suivi du caractère \$ (marqueur de fin)

  Résultats: un booléen *erreur* et une chaîne de caractères *res*

  A la fin de l'exécution du programme, on conclut que le mot appartient ou non à $L(G)$ et si $\alpha \in L(G)$, *res* contient l'arbre syntaxique de α sous forme postfixée.
- Variables du programme: *res* : chaîne de caractères, *erreur* : booléen, y : le caractère courant du mot d'entrée α
- Programme principal:

  erreur $\leftarrow$ *faux*;

  res $\leftarrow \varepsilon$;

  lire(y); –lecture du premier caractère de α qui est affecté à y

  Ana-S; –appel de *Ana-S* correspondant à l'axiome de la grammaire

  si *erreur* alors écrire "mot non engendré par la grammaire"
  sinon si $y = \$$ alors écrire "mot engendré par la grammaire", *res*
          sinon écrire "mot non engendré par la grammaire"
          fsi
  fsi

- procédure associée à un terminal de G:

  Procédure Ana(x : T) ;
    si $\neg$*erreur* alors
      si $y = x$ alors
        debut
          res $\leftarrow$ *res* $\oplus$ x ; *lire*(y); – $\oplus$ concatène deux
                                chaînes de caractères
        fin
      sinon *erreur* $\leftarrow$ *vrai*
      fsi
    fsi

  Procédure qui teste si le caractère en argument est égal au caractère courant du mot donné.

- à chaque non terminal X, on associe une procédure *Ana*-X (voir exemple)

Soit la grammaire $G = (\{S,\ A\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ où

$S \rightarrow aAb\ |1|$

$A \rightarrow cB\ |2|$

$B \rightarrow d\ |3|\ \varepsilon\ |4|$

- procédure pour le non terminal S : *Ana-S*

  procédure *Ana-S*;
  si $\neg$*erreur* alors
    debut
      Ana("*a*"); *Ana-A*; *Ana*("*b*"); $res \leftarrow res \oplus 1$
    fin
  fsi

- procédure pour le non terminal A : *Ana-A*

  procédure *Ana-A*;
  si $\neg$*erreur* alors
    debut
      Ana("*c*"); *Ana-B*; $res \leftarrow res \oplus 2$
    fin
  fsi

procédure pour le non terminal B : *Ana-B*

procédure *Ana-B*;
si $\neg$*erreur* alors
  debut
    si $y =$ "d" alors debut *Ana*("d"); $res \leftarrow res \oplus 3$ fin
    sinon si $y =$ "b" alors $res \leftarrow res \oplus 4$
    sinon $erreur \leftarrow vrai$
  fin
fsi

Exercice : exécuter les procédures pour les mots *acdb* et *acb*

Avant d'entreprendre l'analyse LL d'une grammaire, on doit effectuer dans l'ordre les traitements suivants :

- vérifier que la grammaire est réduite, sinon la réduire
- vérifier que la grammaire n'est pas récursive à gauche, sinon la dérécursiver
- vérifier que la grammaire est factorisée, sinon la factoriser

La réduction des grammaires algébriques est une opération préalabale à l'analyse descendante de la grammaire. Elle est vue en TD.

Conditions de terminaison des procédures de la descente récursive

Théorème

Les procédures d'analyse syntaxique d'une grammaire G s'arrêtent pour toute donnée α si et seulement si la grammaire n'est pas récursive gauche, c'est-à-dire s'il n'existe pas de non terminal A vérifiant $A \overset{+}{\rightarrowtail} A\beta$.

Idée de démonstration : les procédures d'analyse syntaxique ne terminent pas, si le texte en entrée n'est plus consommé à partir d'un certain temps. Cela ne peut se produire que dans le cas où l'on a une suite infinie de règles de la forme suivante :
$A_1 \rightarrow A_2\alpha_2$, $A_2 \rightarrow A_3\alpha_3$, ..., $A_k \rightarrow A_{k+1}\alpha_{k+1}$, ... où les A_i sont des non terminaux.

Comme les A_i sont finis, d'apès le principe des tiroirs, il existe un non terminal A_j tel que $A_j \overset{*}{\rightarrowtail} A_j\beta$

Remarque : cette condition est une condition générale nécessaire à toutes les démarches concernant l'analyse descendante.

Définition (Récursivité gauche immédiate)

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

Exemple

La grammaire suivante comporte plusieurs récursivités gauches immédiates :

$$\begin{array}{l} S \rightarrow ScA \mid B \\ A \rightarrow Aa \mid \varepsilon \\ B \rightarrow Bb \mid d \mid e \end{array}$$

Proposition

On remplace des règles de la forme

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

par les règles suivantes :

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

où A' est un nouveau non terminal.

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

Exemple

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

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

Définition (Récursivité à gauche)

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

Exemple

$$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow Ac \mid Sd \mid c \end{aligned}$$

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 (c'est-à-dire qu'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
fpour

Exemple

$S \to Aa \mid b$
$A \to Ac \mid Sd \mid c$

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)
   $S \to Aa \mid b$
2. $i = 2$
   on remplace $A \to Sd$ par $A \to Aad \mid bd$,
   on obtient ainsi $A \to Ac \mid Aad \mid bd \mid c$
   On élimine la récursivité immédiate à gauche de A
   $A \to bdA' \mid cA'$
   $A' \to cA' \mid adA' \mid \varepsilon$

Le résultat est la grammaire suivante :

$S \to Aa \mid b$
$A \to bdA' \mid cA'$
$A' \to cA' \mid adA' \mid \varepsilon$

Contre-exemple de suppression de la récursivité à gauche

Exemple

$S \rightarrow Sa \mid TSc \mid d \qquad T \rightarrow SbT \mid \varepsilon$

On ordonne les non terminaux : $A_1 = S$, $A_2 = T$,

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

On obtient la grammaire suivante :

$S \rightarrow TScS' \mid dS' \qquad S' \rightarrow aS' \mid \varepsilon$

$T \rightarrow dS'bTT' \mid T' \qquad T' \rightarrow ScS'bTT' \mid \varepsilon$

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 $A \rightarrow \alpha\beta_1$ ou $A \rightarrow \alpha\beta_2$, 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 que l'on ait lu suffisamment de texte.

Exemple

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

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

Algorithme

pour chaque non terminal A

1. trouver le plus long préfixe commun α à deux (ou plus) membres droits de règles
2. si $\alpha \neq \varepsilon$, remplacer $A \to \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \ldots \mid \gamma_p$ (où α n'est pas préfixe de γ_i) par
   $A \to \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_p \qquad A' \to \beta_1 \mid \ldots \mid \beta_n$

Recommencer jusqu'à ce qu'il n'y ait plus de factorisation à effectuer

Exemple d'exécution de l'algorithme

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

- pour S
  - le plus long préfixe est $aAbS$, on obtient
    $S \to aAbSS' \mid a \qquad S' \to eB \mid \varepsilon$
  - le plus long préfixe est a, on obtient
    $S \to aS'' \qquad S'' \to AbSS' \mid \varepsilon$

Exemple d'exécution de l'algorithme (suite)

$S \to aS''$ $\qquad A \to bcB \mid bca$ $\qquad B \to ab$

$S'' \to AbSS' \mid \varepsilon$

$S' \to eB \mid \varepsilon$

- pour A le plus long préfixe est bc, on obtient
  $A \to bcA' \qquad A' \to B \mid a$
- pour B le plus long préfixe est ε

La grammaire obtenue par factorisation est :

$S \to aS''$

$S'' \to AbSS' \mid \varepsilon$

$S' \to eB \mid \varepsilon$

$A \to bcA'$

$A' \to B \mid a$

$B \to ab$

Analyse d'une grammaire G

- 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 (après avoir calculé $P_\epsilon(G)$, *Premier* et *Suivant*)

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

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