Info - TP6

Appartenance à un langage algébrique

O. Caffier

P. Depoorter

Partie 1 : Algorithme de Cocke-Younger-Kasami

Question 1 Si $i, j \in [|1;n|]$, expliquer comment déterminer l'ensemble $E_{i,i}$ à partir de G.

Corrigé :

Soit $i \in [|1;n|]$, on choisit tous les symboles non terminaux de G tel qu'il existe une règle $X \to m_i$. En somme,

$$E_{i,i} = \{X | \exists R \in \mathcal{R}, R = X \to m_i\}$$

Question 2 Si $i, j \in [|1;n|]$ et $i < j$, montrer que :

$$E_{i,j} = \bigcup_{k=i}^{j-1} \{X \in V | X \to YZ, Y \in E_{i,k} \text{ et } Z \in E_{k+1,j}\}$$

Corrigé :

D'une part, soit $X \in E_{i,j}$ ainsi $X \Rightarrow^* m_i \dots m_j$.
Étant donné que l'on a affaire à une grammaire mise sous forme normale de Chomsky, X ne peut se dériver directement en $m_i, \dots, m_j$ (car $i < j$).
Il existe donc Y et Z deux non-terminaux tq $X \to YZ \Rightarrow^* m_i \dots m_j$.
Donc si $X \in E_{i,j}$ alors $\exists k \in [|i; j-1|]$ tq $Y \in E_{i,k}$ et $Z \in E_{k+1,j}$.

D'où $E_{i,j} \subset \bigcup_{k=i}^{j-1} \{X \in V | X \to YZ, Y \in E_{i,k} \text{ et } Z \in E_{k+1,j}\}$

D'autre part, soit $X \in \bigcup_{k=i}^{j-1} \{X \in V | X \to YZ, Y \in E_{i,k} \text{ et } Z \in E_{k+1,j}\}$.
Alors $\exists k \in [|i; j-1|]$ tq $X \to YZ, Y \in E_{i,k}$ et $Z \in E_{k+1,j}$.
Ainsi $YZ \Rightarrow^* m_i \dots m_k m_{k+1} \dots m_j$ DONC $X \Rightarrow^* m_i \dots m_k m_{k+1} \dots m_j$, i.e $X \in E_{i,j}$

D'où l'inclusion réciproque

Question 3 On considère la grammaire G_{ex} dont les règles sont données par :

$$\begin{aligned} S &\to XY \\ T &\to ZT | a \\ X &\to TY \\ Y &\to YT | b \\ Z &\to TZ | b \end{aligned}$$

En utilisant l'algorithme précédent, déterminer si $abab$ appartient à $L(G_{ex})$. On dessinera explicitement la matrice E et le contenu de ses cases en fin d'algorithme.

Corrigé :

$\{T\}$	$\{X;Z\}$	$\{X;T\}$	$\{X;S;Z\}$
$\emptyset$	$\{Y;Z\}$	$\{T;Y\}$	$\{X;Z\}$
$\emptyset$	$\emptyset$	$\{T\}$	$\{X;Z\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\{Y;Z\}$

Et donc on a bien $abab \in L(G_{ex})$ car $S \in \{X;S;Z\}$ (case $e_{1,n}$)

Question 4 Déterminer sa complexité en fonction de $|m|$ et d'une autre grandeur pertinente

Corrigé :

$$\mathcal{O}(|m|^3 \times |\mathcal{R}|)$$

```c
struct regle{
    int type;
    int membre_gauche;
    char lettre;
    int variable1;
    int variable2;
};
typedef struct regle regle;
```

```c
struct grammaire{
    int nb_variables;
    int nb_regles;
    regle* productions;
};
typedef struct grammaire grammaire;
```

Question 5 Implémenter totalement G_{ex}
Corrigé :

```c
grammaire* g_ex = malloc(sizeof(grammaire));
g_ex->nb_variables = 5;
g_ex->nb_regles = 8;
regle* r = malloc(8 * sizeof(regle));
regle r0 = {2,0,'c',2,3};
regle r1 = {2,1,'c',4,1};
regle r2 = {1,1,'a',-1,-1};
regle r3 = {2,2,'c',1,3};
regle r4 = {2,3,'c',3,1};
regle r5 = {1,3,'b',-1,-1};
regle r6 = {2,4,'c',1,4};
regle r7 = {1,4,'b',-1,-1};
r[0] = r0;
r[1] = r1;
r[2] = r2;
r[3] = r3;
r[4] = r4;
r[5] = r5;
r[6] = r6;
r[7] = r7;
g_ex->productions = r;
```

Question 6 Écrire une fonction **void : libere_g (grammaire* g)** qui permet de libérer l'espace mémoire utilisé par g
Corrigé :

```c
void libere_g(grammaire* g){
    free(g->productions);
    free(g);
}
```

Question 7 Écrire une fonction **bool CYK(grammaire* g, char m**[]) implémentant l'algorithme de Cocke-Younger-Kasami.

```c
bool CYK(grammaire* g, char m[]){
    int len_var = g->nb_variables;
    int len_regles= g->nb_regles;
    int len_m = 0;
    // Calcul de la taille du mot
    while (m[len_m] != '\0'){
        len_m++;
    }
    // Initialisation de la matrice
    bool*** mat_res = malloc(len_m * sizeof(bool**));
    for (int i=0; i<len_m; i++){
        mat_res[i] = malloc(len_m*sizeof(bool*));
        for (int j=0; j<len_m; j++){
            mat_res[i][j] = malloc(len_var* sizeof(bool));
            for (int k =0; k<len_var; k++){
                mat_res[i][j][k] = false;
            }
        }
    }

    // Initialiser la diagonale
    for (int i=0; i<len_m; i++){
        for (int id_regle=0; id_regle<len_regles; id_regle++){
            if (g->productions[id_regle].type == 1){
                if (g->productions[id_regle].lettre == m[i]){
                    mat_res[i][i][g->productions[id_regle].membre_gauche] = true;
                }
            }
        }
    }

    // Et maintenant les surdiagonales
    for (int d=1; d<len_m; d++){
        for (int i=0; i <len_m- d; i++){
            // Pour chaque ligne i
            int j = i+d; // On calcule la coordonnee j = i+d tq e_{i,j} soit sur la d-ieme diag
            for (int k=i; k<j; k++){
                // L'indice k du pseudo-code

                for (int id_regle=0; id_regle<len_regles; id_regle++){
                    if (g->productions[id_regle].type==2){
                        if (mat_res[i][k][g->productions[id_regle].variable1] && mat_res[k+1][j][g->
                            productions[id_regle].variable2]){
                            mat_res[i][j][g->productions[id_regle].membre_gauche]=true;
                        }
                    }
                }
            }
        }
    }
    bool res = mat_res[0][len_m-1][0];

    // ON N'OUBLIE PAS DE LIBERER LA MEMOIRE (et on conserve le result ofc)
    for (int i=0; i<len_m; i++){
        for (int j=0; j<len_m; j++){
            free(mat_res[i][j]);
        }
        free(mat_res[i]);
    }
    free(mat_res);

    return res;
}
```

Question 8 Vérifier votre réponse à la question 4 à l'aide de cet algoritme puis déterminer si $m_1 = bbaabaabbab$ et $m_2 = abaabaabbab$ font partie de $L(G_{ex})$.
Corrigé :

mot	résultat
m_1	**false**
m_2	**true**

Question 9 Expliquez comment modifier l'algorithme de façon à ce qu'il réponde correctement au problème du mot lorsqu'on lève les contraintes $\epsilon \notin L(G)$ et $m \neq \epsilon$ (on suppose toujours que G est sous forme normale de Chomsky).
Corrigé :
Sous forme normale de Chomsky, le seul symbole qui peut donner le mot vide est S, il suffit donc de vérifier si il y a $S \Rightarrow^* \varepsilon$ en ajoutant une signification au mot ε (en lui associant une valeur dans le code ASCII par exemple).

Partie 2 : Analyse descendante

Question 1 Pour un entier $n \in \mathbb{N}$ arbitraire, donner un mot de longueur au moins n engendré par la grammaire G et la dérivation à gauche correspondante.

Corrigé :

$$\textbf{sym}\ldots\textbf{sym\#}$$

Question 2 Donner le code des fonctions **parseL** et **parseE**

Corrigé :

```
type token = Sym | Lpar | Rpar | Eof

exception SyntaxError

let rec parseS l =
  match l with
  |(Sym|Lpar|Eof)::_ -> (match parseL l with
                          | Eof::q -> q
                          |_ -> raise SyntaxError)
  |[] | Rpar::_ -> raise SyntaxError
and parseL l =
  match l with
  |(Rpar|Eof)::_ -> l
  |(Sym|Lpar)::_ -> parseL (parseE l)
  |[] -> raise SyntaxError
and parseE l =
  match l with
  |Sym::q-> q
  |Lpar::q->
    (match parseL q with
    |Rpar::r -> r
    |_ -> raise SyntaxError)
  |[]|(Rpar|Eof)::_ -> raise SyntaxError
```

Question 3 Donner le code de la fonction **accepts**

Corrigé :

```
let accepts  (l : token list) : bool =
    try
      parseS l = []
    with SyntaxError -> false
```

Question 4 Indiquer quels sont les symboles nuls de la grammaire G

Corrigé :

Les symboles nuls sont : $\{L\}$

Question 5 Montrer que cet algorithme termine

Corrigé :

On désigne le variant suivant :

$$n = Card\{X \in V | Nul(X) = \textbf{vrai}\}$$

avec Nul le tableau de booléen manipulé par l'algorithme

Ainsi, n est bien strictement croissant et majoré par le nombre de non-terminaux $\implies$ l'algorithme termine.

Question 6 Montrer que cet algorithme détermine bien les valeurs de $NUL(X)$
Corrigé :

$$\text{Montrons que pour } X \in V, X \Rightarrow^* \varepsilon \Leftrightarrow Nul(X) = \textbf{true}$$

D'une part,

— Si $n \in \mathbb{N}^*$, notons H_n : "après n étapes, si $Nul(X)$ alors $X \Rightarrow^* \varepsilon$

— Pour $n = 1$, $Nul(X) \Leftrightarrow \exists R \in \mathcal{R}, R = X \to \varepsilon$

— $\to$ OK

D'où le raisonnement par doublej inclusion

D'autre part,

— Si $n \in \mathbb{N}^*$, notons H_n : "si $X \Rightarrow^n \varepsilon$ alors $Nul(X) = \textbf{true}$

— $n = 1$, $X \Rightarrow \varepsilon$ donc $\exists R \in \mathcal{R}, R = X \to \varepsilon$ donc initialement $Nul(X) = \textbf{true}$

— $H_n \implies H_{n+1}$,
Si $X \Rightarrow^{n+1} \varepsilon$ alors $\exists X_1, \ldots, X_p$ non terminaux tq $X \to X_1 \ldots X_p \Rightarrow^n \varepsilon$.
Ainsi, **par H.R**, $Nul(X_1) = \ldots = Nul(X_p) = \textbf{true}$.
Et donc, par construction de l'algorithme, on attribuera $Nul(X) \leftarrow \textbf{true}$

Question 7 Donner les ensembles *PREMIERS*(X) et *SUIVANTS*(X) pour la grammaire G prise en exemple
Corrigé :

$$PREMIERS(S) = \{\textbf{sym}; (; \#\}$$
$$PREMIERS(L) = \{\textbf{sym}; (\}$$
$$PREMIERS(E) = \{\textbf{sym}; (\}$$

$$SUIVANTS(S) = \emptyset$$
$$SUIVANTS(L) = \{\#;)\}$$
$$SUIVANTS(E) = \{\textbf{sym}; (;); \#\}$$

Question 8 Montrer que $L(G) = L(G')$
Corrigé :
Commençons par un lemme

Lemme

Si $L \Rightarrow^* m_1 ... m_k$ avec $k \geq 1$ alors $\exists i \in [|0; k|], L \Rightarrow^* m_1 ... m_i$ et $E \Rightarrow^* m_{i+1} ... m_k$

DEMO : (par récurrence sur le nombre de dérivations)

— Initialisation : Si $n = 3$ (3 est le plus petit entier possible ici) :

On ne peut qu'avoir m = sym avec $L \Rightarrow EL \Rightarrow E \Rightarrow sym$ (Un autre choix de dérivations entraînerait plus de 3 dérivations.)

Ainsi $L \Rightarrow \epsilon$ et $E \Rightarrow sym$ donc $i = 0$

— Hérédité : Si $n \geq 3$:
On suppose le résultat vrai pour tout $q \in [|0; n|]$ et $L \Rightarrow^* m_1 ... m_k$.
On peut dire que : $\exists p \in [|1; k|], E \Rightarrow^* m_1 ... m_p$ et $L \Rightarrow^r m_{p+1} ... m_k$ avec $r < n$
$r < n$ donc on peut utiliser l'hypothèse de récurrence : $\exists i \in [|p; k|], L \Rightarrow^* m_{p+1} ... m_i$ et $E \Rightarrow^* m_{i+1} ... m_k$
Donc : $L \Rightarrow EL \Rightarrow^* m_1 ... m_p L \Rightarrow^* m_1 ... m_p m_{p+1} ... m_i$ et $E \Rightarrow^* m_{i+1} ... m_k$
Ainsi, i convient.

Montrons que $L(G) = L(G')$. Montrons par rec. sur le nombre de dérivations que $X \Rightarrow^* m_1 \dots m_k$ alors $X' \Rightarrow^* m_1 \dots m_k$ pour tout $X \in V$.

— Si $n = 1$:
  — $S : \emptyset$
  — $L : L \to \varepsilon$ et $L' \to \varepsilon$ **OK**
  — $E \to$ **sym** et $E' \to$ **sym** **OK**
— Soit $n \geq 1$, $H_1, \dots, H_n \implies H_{n+1}$
  — Soit m tq $S \Rightarrow^{n+1} m_1 \dots m_k$.
    On a $S \Rightarrow L\# \Rightarrow^n m_1 \dots m_{k-1}\#$ avec $L \Rightarrow^n m_1 \dots m_{k-1}$ et H_n vraie donc par H.R $L' \Rightarrow^n m_1 \dots m_{k-1}$
    Donc $S' \to L'\# \Rightarrow^n m_1 \dots m_{k-1}\#$ donc $S' \Rightarrow m$

OK

  — Soit m tq $E \Rightarrow^{n+1} m_1 \dots m_k$.
    On a $E \Rightarrow (\ L\ ) \Rightarrow^n (\ m_2 \dots m_{k-1}\ )$ avec donc $L \Rightarrow^n m_2 \dots m_{k-1}$ donc par H.R, $L \Rightarrow^n m_2 \dots m_{k-1}$.
    Donc $E' \Rightarrow (\ L'\ ) \Rightarrow^n (\ m_2 \dots m_{k-1}\ )$, i.e $E' \Rightarrow^* m$

OK

  — Soit m tq $L \Rightarrow^{n+1} m_1 \dots m_k$,
    Alors, d'après notre lemme, $\exists i \in [|0;k|], L \Rightarrow^{r_1}$ et $E \Rightarrow^{r_2} m_{i+1} \dots m_k$ avec $r_1, r_2 \leq n$ par H.R $L' \Rightarrow^{r_1} m_1 \dots m_i, E' \Rightarrow^{r_2} m_{i+1} \dots m_k$.
    Donc $L' \Rightarrow L'E' \to^n m_1 \dots m_k = m$, i.e $L' \Rightarrow^* m$

OK

Question 9 Construire la table LL pour cette seconde grammaire. Permet-elle de coder un algorithme pour l'analyse syntaxique des mots générés par cette grammaire ?

Corrigé :

	sym	(	)	#
S'	$L'\#$	$L'\#$		$L'\#$
L'	$\varepsilon, L'E'$	$\varepsilon, L'E'$	ε	ε
E'	**sym**	(L')		

On constate que dans ce tableau, il y a deux cases où deux choix apparaissent, ce qui ne permet pas l'analyse.