CORRIGÉ ABRÉGÉ DE LA SÉRIE D'EXERCICES n° 1 de ThL

EXERCICE **1** :

1) Les mot w_1 et w_3 n'appartiennent pas L(G)
   les mots w_2 et w_4 sont dans L(G) : $S \vdash aS \vdash aaS \vdash aabA \vdash aabcA \vdash aabccA \vdash aabcccA \vdash w_2$
   et pour w_4 : $S \vdash aS \vdash abA \vdash ab = w_4$.
2) Soit $L = \{ a^n b c^m \,/\, n, m \geq 0 \}$. Montrons que L(G)=L en prouvant la double inclusion :
   - $L(G) \subseteq L$: soit w un mot de L(G), donc w est généré à partir de S en appliquant n fois les règles de production de G. Montrons par récurrence sur n que $w \in L$:
     - si n=2 alors on a : $S \vdash bA \vdash b$; $w=b \in L$. Supposons que la propriété reste vraie jusqu'au rang n=k.
     - pour n=k+1 : la dernière règle appliquée est $A \to \varepsilon$; donc avec k règles on a $S \vdash_* wA$, où w est donc obtenu en appliquant k fois les règles de G. D'après l'hypothèse de récurrence $w \in L$.
   - $L \subseteq L(G)$: Soit $w \in L$. Donc w s'écrit comme $w= a^n b c^m$. w peut être dérivé de S en appliquant n fois la règle $S \to aS$ puis une fois la règle $S \to bA$, puis encore m fois la règle $A \to cA$ et enfin une fois la règle $A \to \varepsilon$. Donc $w \in L(G)$.

EXERCICE **2** : Nous donnons ici les langages engendrés par les grammaires G_i (i=1,..,5). (Pour que la réponse soit complète, il faut le prouver comme c'est fait dans l'exercice 1).

1) $L(G_1) = \{ a^n b^m c^m \,/\, n \geq 1 \,;\, m \geq 0 \}$
2) $L(G_2) = \{ a^n b^m c^{n+m} \,/\, n, m \geq 0 \}$
3) $L(G_3) = \{ w \in \{a, b\}^* \,/\, |w|_a = |w|_b$ et $\forall$ u préfixe de w, $|u|_a \geq |u|_b \}$
4) $L(G_4) = \{ a^n b^m c^n \,/\, n \geq 0 \,;\, m \geq 1 \}$
5) $L(G_5) = \{ a^n b^n c^n \,/\, n \geq 1 \}$

EXERCICE **3** :

1) G_1 est de type 2.
2) G_2 est de type 0.

EXERCICE **4** :

a) pour L_1 : il est engendré par $G_1 = (\{0\}, \{S\}, P_1, S)$, où P_1 : $S \to 00S \mid \varepsilon$
b) pour L_2 : il est engendré par $G_2 = (\{0, 1\}, \{S\}, P_2, S)$, où P_2 : $S \to 0S1 \mid \varepsilon$
c) pour L_3 : il est engendré par $G_3 = (\{a, b\}, \{S\}, P_3, S)$, où P_3 : $S \to aSbb \mid \varepsilon$
d) pour L_4 : il est engendré par $G_4 = (\{a, b\}, \{S, B\}, P_4, S)$,
   où P_4 : $S \to aSbB \mid \varepsilon$; $B \to b \mid \varepsilon$

e) pour L_5 : il est engendré par $G_5 = (\{a, b, 0, 1\}, \{S, A\}, P_5, S)$,

où P_5 : $S \rightarrow 0S1 \mid A$;

$A \rightarrow aAa \mid bAb \mid \varepsilon$

f) pour L_6 : il est engendré par $G_6 = (\{a, b\}, \{S, A\}, P_6, S)$,

où P_6 : $S \rightarrow aSb \mid aAb$;

$A \rightarrow bAa \mid ba$

g) pour L_7 : il est engendré par $G_7 = (\{a, b\}, \{S, A\}, P_7, S)$,

où P_7 : $S \rightarrow AAAS \mid AAA$;

$A \rightarrow a \mid b$

h) pour L_8 : il est engendré par $G_8 = (\{0, 1\}, \{S\}, P_8, S)$,

où P_8 : $S \rightarrow 0S1 \mid 0S \mid \varepsilon$

i) $L_9 = \{ 0^i 1^j \ / \ i > j \} \cup \{ 0^i 1^j \ / \ i < j \}$;) L_9 est engendré par $G_9 = (\{0, 1\}, \{S, S_0, S_1\}, P_9, S)$,

où P_9 : $S \rightarrow S_0 \mid S_1$;

$S_0 \rightarrow 0S_01 \mid 0S_0 \mid 0$;

$S_1 \rightarrow 0S_11 \mid S_11 \mid 1$

j) L_{10} : il est engendré par $G_{10} = (\{0, 1\}, \{S, A, B, C, D\}, P_{10}, S)$,

où P_{10} : $S \rightarrow BCD$

$C \rightarrow AC \mid a$

$Aa \rightarrow aaA$

$AD \rightarrow D$

$Ba \rightarrow aB$

$BD \rightarrow \varepsilon$

EXERCICE **5** :

Soient les langage $L = \{0, 1\}^*$ et $L' = \{ 0^n1^n \ / \ n \geq 0 \}$. L est de type 3 (vérifier le !) ; mais L', qui est inclus dans L, n'est pas de type 3 (il est de type 2).

EXERCICE **6** :

1) L peut être généré par la grammaire, de type 3, $G = (\{a, b, c\}, \{S, C\}, P, S)$

où P : $S \rightarrow aaS \mid bcC$

$C \rightarrow ccC \mid \varepsilon$

2) Une autre grammaire de type 2, et qui n'est pas de type 3, qui engendre L :

$G' = (\{a, b, c\}, \{S, A, C\}, P', S)$

où P' : $S \rightarrow AbcC$

$A \rightarrow aaA \mid \varepsilon$

$C \rightarrow ccC \mid \varepsilon$

EXERCICE **7** :

1) L peut être généré par la grammaire, de type 3, $G = (\{0, 1\}, \{S, A\}, P, S)$

où P : $S \rightarrow 0S \mid 1A \mid \varepsilon$

$A \rightarrow 0A \mid 1S$

2) Une autre grammaire de type 2, et qui n'est pas de type 3, qui engendre L :

$G' = (\{0, 1\}, \{S\}, P', S)$

où P' : $S \rightarrow 0S \mid S1S1S \mid \varepsilon$

EXERCICE **8** :

1) $L(G) = \{ a^n b^{2.[n/2]} / n \geq 0\}$; ([x] est la partie entière de x)

On peut aussi écrire L(G) comme $\{ a^{2n+1} b^{2n} / n \geq 0 \} \cup \{ a^{2n} b^{2n} / n \geq 0 \}$

2) Grammaire de type 2 équivalente à G : G' = ({a, b}, {S}, P', S)

où P' : $S \rightarrow aaSbb \mid a \mid \varepsilon$

EXERCICE **9** :

1) $L(G) = \{ a^n b^{2n} / n \geq 0\}$;

2) Grammaire de type 2 équivalente à G : G' = ({a, b}, {S}, P', S)

où P' : $S \rightarrow aSbb \mid \varepsilon$

EXERCICE **10** :

1) Une grammaire de type 2 pour L pourrait être G = (π, N, P, S) ; où N = {S}

et P : S → S+S | S*S | a | (S)

2) Une grammaire de type 2 pour L' pourrait être G' = (π, N', P', S) ; où N' = {S, A}

et P' : S → S+S | S*S | a | (A)

A → S+S | S*S | a

EXERCICE **11** :

Pour générer ces identificateurs on utilisera la grammaire G = (π, N, P, <Id1>) ;

où π = {A..Z, a..z, 0..9} ; N = {<Id1>, <Id2>, <Id3>, <Lettre>, <Chiffre>}

et P : <Id1> → <Lettre> <Id2>

<Id2> → <Id3> <Id2> | ε

<Id3> → <Lettre> | <Chiffre>

<Lettre> → A | B | .. | Z | a | b | .. | z

<Chiffre> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

EXERCICE **12** :

On pourrait procéder, en TurboPascal, comme suit :

on lit la chaîne dans le string t et on vérifie si elle se termine par '#'. Si c'est le cas on passe à l'étape suivante en invoquant la procédure S et ainsi le processus de vérification est entamé ; S appelle A qui s'appelle elle-même lorsqu'elle rencontre un 'a' ou s'arrête sinon. Bien sûr qu'à chaque étape on vérifiera qu'il y a les bons caractères aux bons endroits ; sinon il y a erreur.

```
uses wincrt;
var car : char;
    erreur : boolean;
    t : string;
    k,n : integer;

procedure lex;
begin
  car := t[k];
  k := k+1;
end;
```

```
procedure A;
begin
  if car='a' then
  begin
    lex; (* appel de la procédure lex pour renvoyer le prochain caractère *)
    A; (* appel de la procédure A *)
    if car<>'c' then
      erreur := true
    else
      lex (* appel de la procédure lex pour renvoyer le prochain caractère *)
  end
  else
    if car<>'b' then
      erreur:=true
    else
      lex (* appel de la procédure lex pour renvoyer le prochain caractère *)
end;

procedure S;
begin
  lex; (* appel de la procédure lex pour renvoyer le prochain caractère *)
  A; (* appel de la procédure A *)
  if car<>'#' then
    erreur := true
end;

Begin
  erreur := false;
  t := '';
  k := 1;
  repeat
    write(' donner une chaine se terminant par # : ');
    readln(t);
    n := length(t);
    if (n=0) or (t[n]<>'#') then
      writeln('la chaine ne se termine pas par # ');
  until (n>0) and (t[n]='#');
  S; (* appel de la procédure S *)
  if erreur then
    writeln(' la chaine ',t,' n''appartient pas au langage ')
  else
    writeln(' la chaine ',t,' appartient au langage ');
  writeln;
End.
```

<u>Question :</u> comment peut-on optimiser ce programme ?