Compte rendu TP12 RDF – Langage et grammaire

Introduction

Dans ce TP, nous allons voir 2 exercices liés aux chaînes de caractère comme : ababc. Le premier consistera à calculer à la main puis à utiliser l'algorithme de Levenshtein, pour trouver la distance entre 2 mots. Dans le deuxième, exercice, on fera l'arbre de dérivation grâce à un type de grammaire donné, ainsi que coder une fonction qui retrouve vraie si le mot est un palindrome.

Distance de chaînes

A la main

Rappel : Algorithme de comparaison de chaines :

```
for i in 0..m : c[i,0] = i
for j in 0..n : c[0,j] = j
for i in 0..m
     for j in 0..n
          c[i,j] = min(c[i-1,j]+1, // insertion
          c[i,j-1]+1, // suppression
          c[i-1,j-1]+1 – (x[i]=y[j]?1:0) // substitution
     //ou même car
return c[m,n] ;
```

X -> excused
Y -> exhausted

0	**e**	**x**	**h**	**a**	**u**	**s**	**t**	**e**	**d**
e	0	1	2	3	4	5	6	7	8
x	1	0	1	2	3	4	5	6	7
c	2	1	1	2	3	4	5	6	7
u	3	2	2	2	2	3	4	5	6
s	4	3	3	3	3	2	3	4	5
e	5	4	4	4	4	3	3	3	4
d	6	5	5	5	5	4	4	4	**3**

La distance minimale obtenue est 3.

Sur machine

levenshtein('excused','exhausted',D).

D = 3.

Donc on obtient le même résultat que le résultat théorique.

Tests sur les exemples d'apprentissage :

	aabbc	ababcc	babbcc	bccba	bbbca	cbbaaa a	caaaa	cbcaab	baaca
abacc	3	1	2	4	3	5	4	4	3
ccab	4	4	5	3	4	5	3	2	4
ccbba	3	5	4	2	3	4	3	4	4
bbaaac	4	3	4	4	3	2	3	3	3

Pour 'abacc', par le calcul de la distance levenshtein, on trouve que ce mot est plus proche de la classe 1 que des autres sans ambigüité grâce au mot 'ababcc'. Pour 'ccab', il est plus proche de la classe 3 grâce au mot 'cbcaab'. Pour 'ccbba', ce mot est plus proche de la classe 2 grâce au mot 'bccba'. En enfin pour le mot 'bbaaac', on remarque qu'il est proche du mot 'cbbaaaa' de la classe 2.

Récapitulatif :

- Abacc : Classe 1
- Ccab : Classe 3
- Ccbba : Classe 2
- Bbaaac : Classe 2

Arbre de dérivation pour une grammaire

A la main : la grammaire G

Considérez la grammaire G:

```
- Alphabet A = {a,b,c}
- Axiome = S
- Non-terminaux = {A,B}
- Règles de production P =
```

S --> cAb
A --> aBa
B --> aBa
B --> cb

C'est une grammaire de type 1 : type contexte.

Pour une meilleure compréhension, nous allons définir X les lettres qui nous manquent et qui définissent le langage généré par la grammaire G.

D'après cette grammaire, on doit avoir en première lettre un 'c' et en dernière lettre un 'b', cela nous donne cXb.

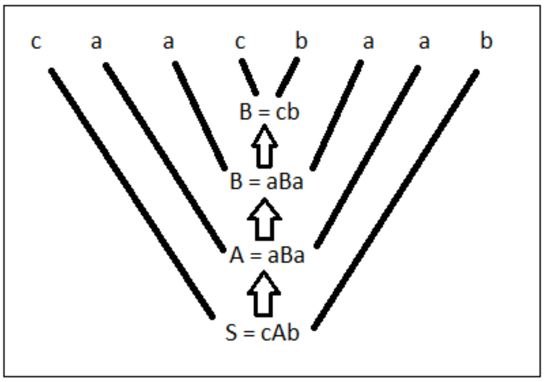
Ensuite, on remarque avec la règle A->aBa, qu'il y a un 'a' en deuxième et avant dernière place. Ce qui nous donne caXab.

Puis, on a 2 possibilités soit, on peut boucler sur la lettre 'a', soit il y a qu'un nombre n =1, de a.

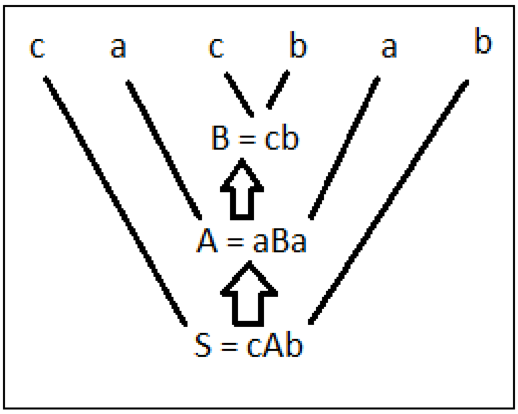
Enfin, les lettres du centre sont cb. On a donc L(G) = { ca^ncba^nb , avec n>=1}.

Arbres de dérivation :

- "caacbaab"



- "cacbab"



Sur machine : les palindromes

Question 1 :

- Alphabet Palindromes = {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}

- Axiome = S

- Non-terminaux = {S}

- Règles de production Palindromes =

S --> aSa | bSb | cSc | dSd | eSe | fSf | gSg | hSh

S --> iSi | jSj | kSk | lSl | mSm | nSn | oSo | pSp

S --> qSq | rSr | sSs | tSt | uSu | vSv | wSw | xSx

S --> ySy | zSz |

S --> a | b | c | d | e | f | g | h | i | j | k

S --> l | m | n | o | p | q | r | s | t | u | v

S --> w | x | y | z |

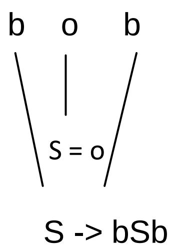
S --> epsilon

Tests :

palin('bob').

true.

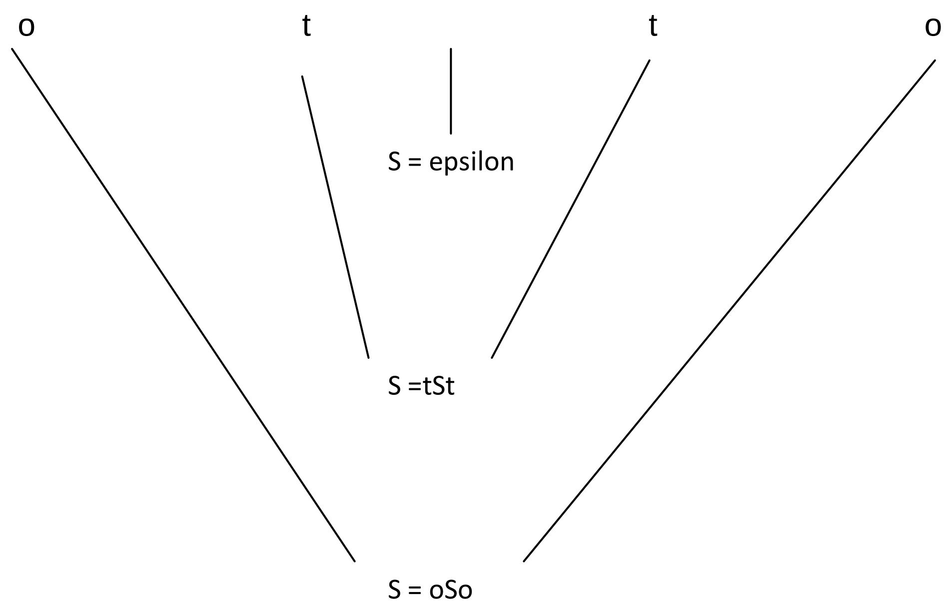
Arbre de décision :



palin('otto').

True

Arbre de décision :



Question 2 :

Cette grammaire est de type 1.

Question 3 :

palin('esoperesteicietserepose').
true.