RDF : TP 11 : Chaînes, langages et grammaires

binôme : Benjamin Ruytoor et Aurore Allart
date : 4 avril 2013

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.

1. Distance de chaînes

1. A la main

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

Ce qui donne comme distance = 3.

2. Sur machine

Pour calculer la distance de ces mots, on utilise l'algorithme de Levenshtein.

	aabbc	**ababcc**	**babbcc**	**bccba**	**bbbca**	**cbbaaaa**	**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 ambiguité 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.

2. Arbre de dérivation pour une grammaire

1. A la main : la grammaire G

Question 1 :

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

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

Ceci est le type 1 de grammaire, c'est à dire en contexte.

Question 2 :

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}.

Question 3 :

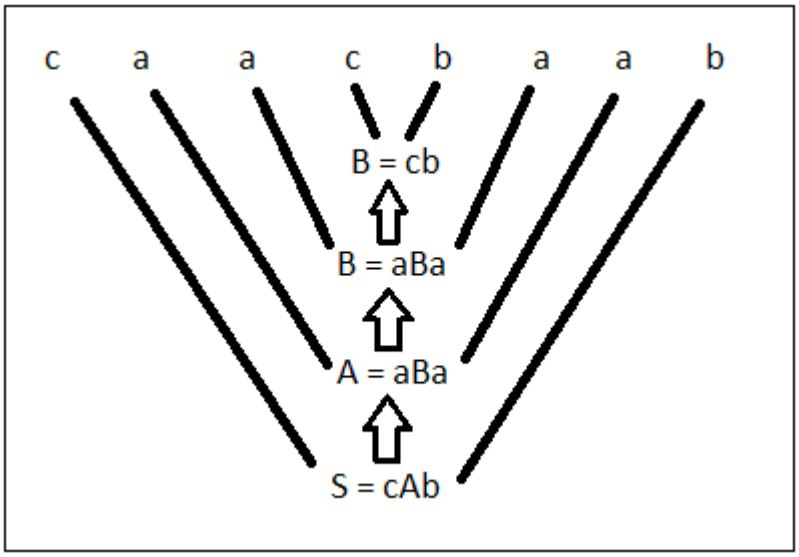

Figure 1 : arbre de décision du mot caacbaab.

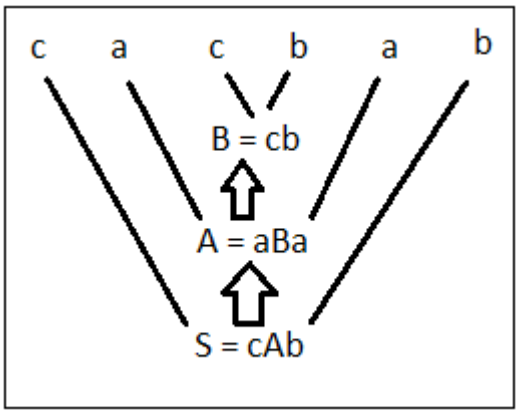

Figure 2 : arbre de décision du mot cacbab.

2. Sur machine : les palindromes

Question 1 :

- Alphabet A = {a,...,z}
- Axiome = S
- Non-terminaux = {Da,Db,...,Dz}
- Règles de production P =
  - S -> U
  - U -> ? | E (vide) | Da | ... | Dz
  - Da = aUa
  - Db = bUb
  - Dc = cUc
  - :
  - Dz = zUz

Question 2 :

Cette grammaire est de type 1.

Question 3 :

Voici le code qui nous implémente le grammaire demandé, et qui teste si un mot est un palindrome.

```
palin('').
palin(S) :- atom_length(S, L), L == 1, !.

palin(S) :- sub_atom(S, 0, 1, _, PRE), atom_length(S, L), L2 is L - 1, sub_atom(S, L2, 1, _, SU), PRE == SU , sub_atom(S, 1, _, 1, S2), palin(S2).
```

Et voici quelques tests, pour montrer que ça marche.

```
?- palin('esoperesteicietserepose').
true.

?- palin('ab').
false.
```