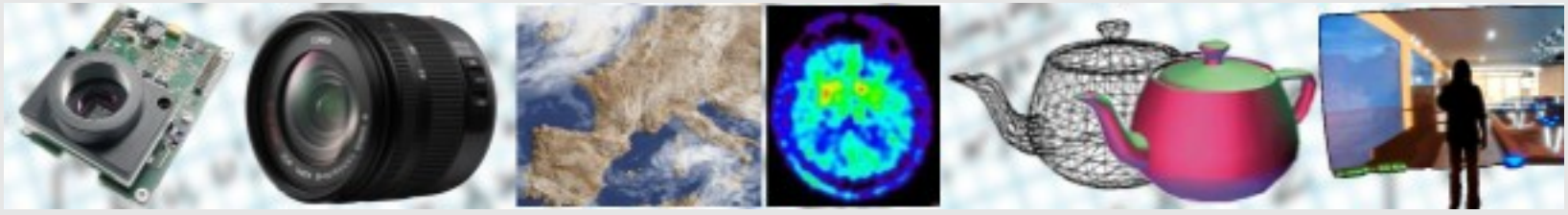


RdF – Reconnaissance des Formes
Semaine 12 : Chaînes, langages et grammaires

Master ASE : http://master-ase.univ-lille1.fr/
Master Informatique : http://www.fil.univ-lille1.fr/
Spécialité IVI : http://master-ivi.univ-lille1.fr/

IEEA

Plan du cours

1 – Reconnaissance de chaînes

correspondance, distance *edit*

2 – Langages et grammaires

exemples de grammaires, définition, types de grammaires

Reconnaissance de chaînes : introduction

Définition

une chaîne est une séquence ordonnée de symboles discrets tirés d'un alphabet

$$\{0,1\},\{0,1,2,\ldots,9\},\{a,b,c,\ldots,z\},\{A,G,C,T\}$$

Exemple

$$x = AGCTTCGAATC$$

où chaque lettre représente un acide nucléique de l'ADN (adénine, guanine, cytosine et thymine)

en raison du caractère *nominal* des éléments, pas de notion évidente/intuitive de distance entre les chaînes

les chaînes ne sont pas des vecteurs

longueurs éventuellement différentes

Correspondance de chaînes

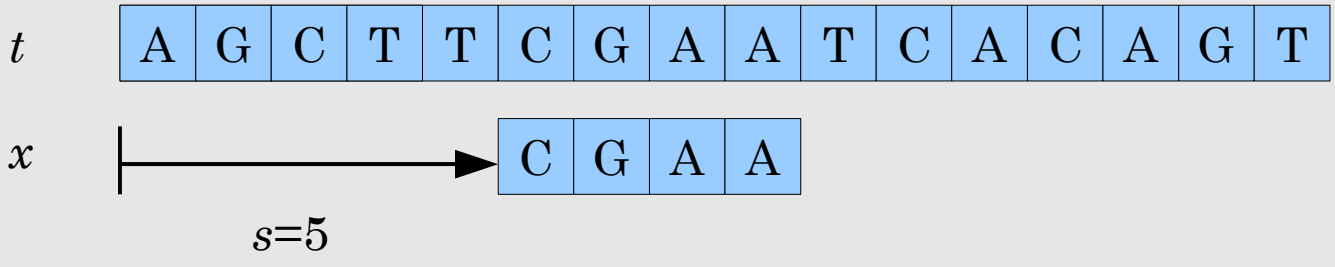
Problème (probablement le plus important en RdF chaînes)

étant donnés une chaîne *x* et un texte *t*, déterminer si *x* est une sous-chaîne de *t*, et à quelle position

= lister tous les *shifts* valides

(*shift* : décalage *s* requis pour aligner le 1er caractère de *x* avec le *s*+1ème caractère de *t*)

Exemple



Correspondance de chaînes

Algorithme naïf

```
n = longueur(t), m = longueur(x)
for i in 0..n-m
  if (x[1,m] = t[i+1,i+m]) print s
return
```

Algorithme de Boyer-Moore

efficace, implémenté dans la plupart des éditeurs de texte

```
1  begin initialize A, x, text, n ← length[text], m ← length[x]
2        F(x) ← last-occurrence function
3        G(x) ← good-suffix function
4        s ← 0
5        while s ≤ n − m
6              do j ← m
7              while j > 0 and x[j] = text[s + j]
8                    do j ← j − 1
9                    if j = 0
10                     then print "pattern occurs at shift" s
11                          s ← s + G(0)
12                     else s ← s + max[G(j), j − F(text[s + j])]
13   return
14 end
```

Correspondance de chaînes

Puissance de l'algorithme de Boyer-Moore

après un *mismatch*, deux heuristiques indépendantes et parallèles proposent des *shifts* sûrs, le plus grand est choisi

- tous deux opèrent de droite à gauche à chaque position

1. Heuristique du « bon suffixe »

aligne le suffixe courant de *t* avec le plus grand préfixe de *x* qui correspond, le cas échéant

==> construire la table des occurrences de tous les suffixes de *x*

ex : « estimates » = s,2 ; es,1 ; tes,0 ; 0 pour tous les autres

Correspondance de chaînes

Puissance de l'algorithme de Boyer-Moore

après un *mismatch*, deux heuristiques indépendantes et parallèles proposent des *shifts* sûrs, le plus grand est choisi

- tous deux opèrent de droite à gauche à chaque position

2. Heuristique du « mauvais caractère »

utilise le caractère le plus à droite de *t* qui ne correspond pas avec le caractère aligné de *x*

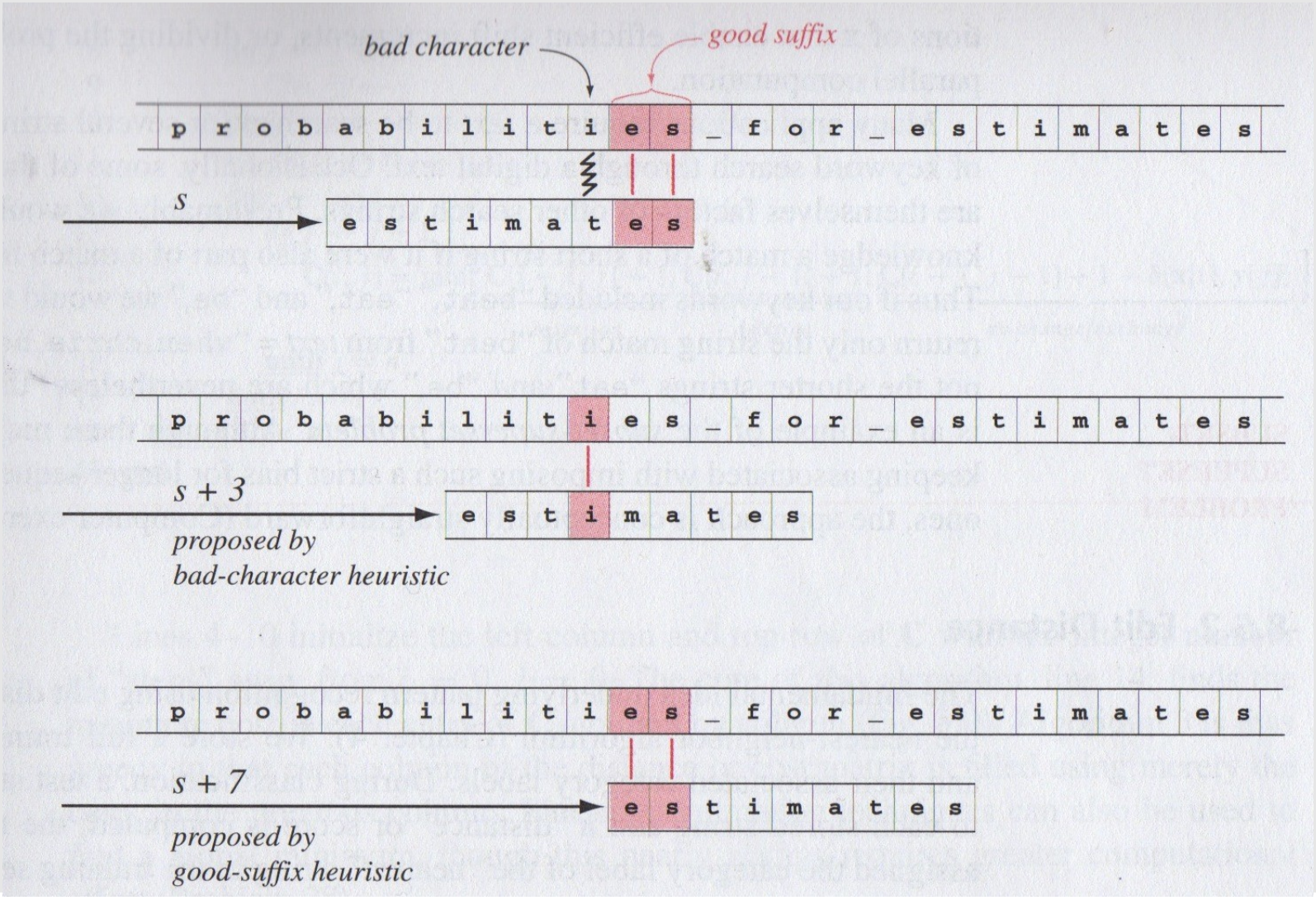
propose d'aligner ce mauvais caractère avec l'occurrence la plus à droite de ce caractère dans *x* s'il existe

==> construire la table des caractères de *x* contenant le rang de leur dernière occurrence (0 pour les autres)

ex : « estimates » = a,6 ; e,8 ; i,4 ; m,5 ; s,9 ; t,7 ; 0 pour les autres

Correspondance de chaînes

Exemple d'exécution de l'algorithme de Boyer-Moore



Distance de chaînes

Problème

étant données deux chaînes *x* et *y*, combien d'opérations fondamentales permettent de passer de l'une à l'autre?

Opérations

Substitution: un caractère de *x* est remplacé par un autre dans *y*
Insertion: un caractère est inséré dans *x* (longueur: +1)
Suppression: un caractère est supprimé dans *x* (longueur: -1)

Algorithme « Edit Distance »

```
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]
```

Alternatives de correspondance

Correspondance avec erreurs

étant donnés une chaîne *x* et un texte *t*, déterminer le *shift s* pour lequel la distance *edit* entre *x* et la sous-chaîne de *t* à la position *s*+1 est minimale
algorithme similaire à la distance edit

==> minimiser distance(*x*,*y*), avec *y* une sous-chaîne de *t*

Correspondance avec le symbole « don't-care »

on a en plus le caractère ø, qui peut correspondre à tous

formellement, pas de différence dans les algorithmes
- mais ils deviennent moins efficaces

Langages et grammaires : introduction

Théorie des langages

à l'intersection de l'informatique théorique, de la logique, et de la linguistique

Applications

compilation (analyse syntaxique), programmation logique, traitement des langues naturelles

Langages et grammaires : un exemple

Soit l'exemple suivant

on se donne un alphabet de symboles composé de *a* et de *b*, et on étudie le langage *L* contenant l'ensemble des mots composés de la manière suivante :

- **un nombre arbitraire de *a* (au moins 1), suivis de**
- **un nombre arbitraire de *b* (éventuellement 0)**

on peut décrire l'ensemble des mots corrects avec :

$$a^n b^m, n \geq 1, m \geq 0$$

exemples de mots bien formés (appartenant à *L*)

$$a, aaaaab, abbbbb, ab, aaaaaaaaaaaaa$$

exemples de mots mal formés (n'appartenant pas à *L*)

$$bbb, aabbaa, abc, abab$$

Langages et grammaires : un exemple

Suite de l'exemple

l'ensemble des mots bien formés de L peut être généré par des règles de production

$$R_1 : S \rightarrow A\,B$$
$$R_2 : A \rightarrow a$$
$$R_3 : A \rightarrow aA$$
$$R_4 : B \rightarrow \epsilon$$
$$R_5 : B \rightarrow bB$$

$$R_1 : S \rightarrow A\,B$$
$$R_2 : A \rightarrow aA \mid a$$
$$R_3 : B \rightarrow bB \mid \epsilon$$

A, B et S sont des symboles non-terminaux (uniquement en partie droite des règles), S est l'axiome (point de départ)
application d'une règle : remplacer dans un mot la partie gauche de la règle par la partie droite
dérivation : application d'une séquences de règles de production en partant de l'axiome

Autre exemple

Langue naturelle

soient les règles suivantes

$R_1 : P \rightarrow S\ V\ C$
$R_2 : S \rightarrow N$
$R_3 : C \rightarrow N$
$R_4 : N \rightarrow Art\ Nom$
$R_5 : N \rightarrow Art\ Adj\ Nom$
$R_6 : Art \rightarrow$ le
$R_7 : Nom \rightarrow$ chat
$R_8 : Nom \rightarrow$ rat
$R_9 : Adj \rightarrow$ vieux
$R_{10} : Adj \rightarrow$ petit
$R_{11} : V \rightarrow$ attrape

exemples de phrases grammaticalement correctes :

- « le chat attrape le petit chat »

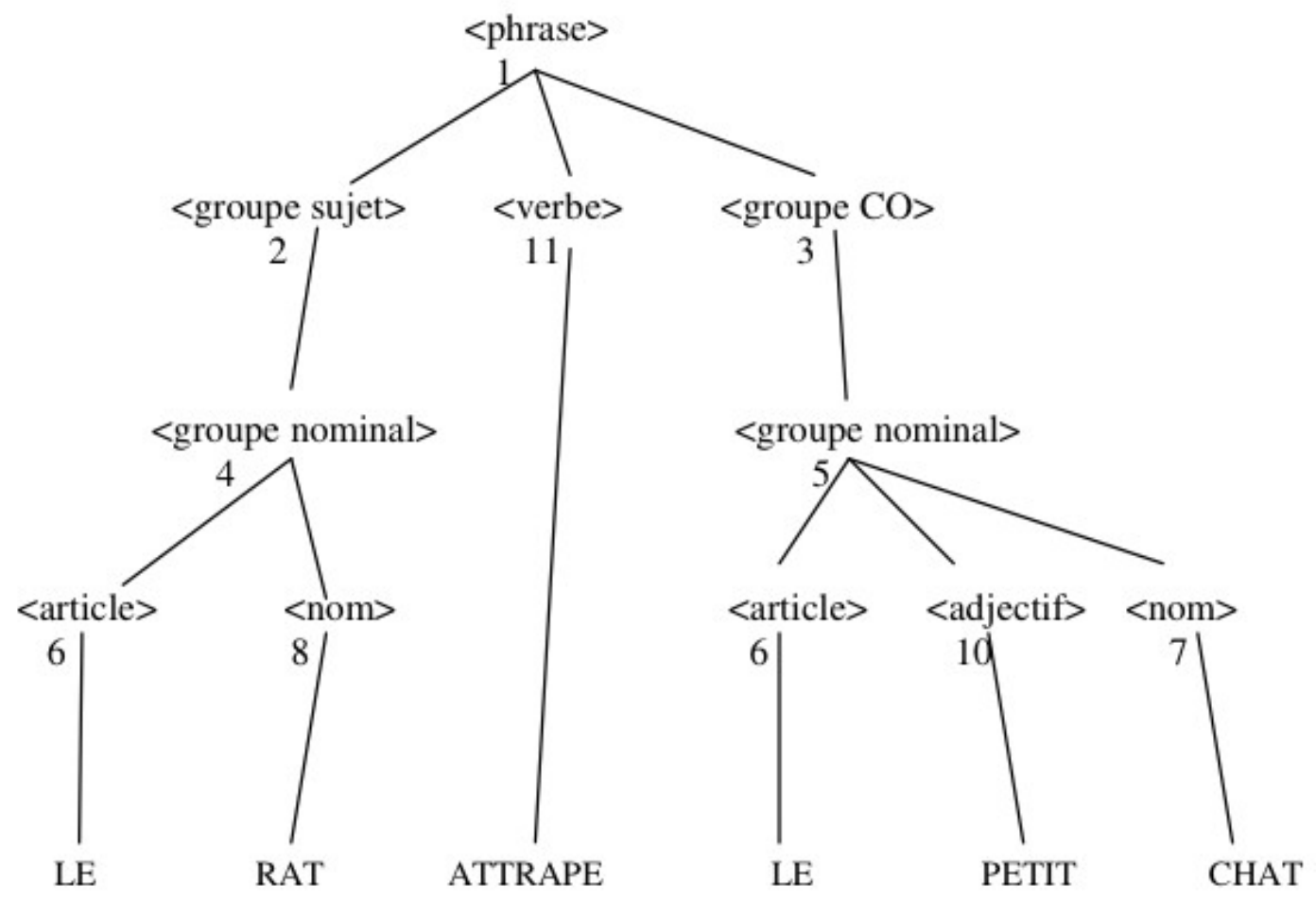
- « le vieux rat attrape le rat »

on peut associer à chaque phrase un arbre de dérivation

Autre exemple

Arbre de dérivation

séquence des règles appliquées



Définition

Définition formelle d'une grammaire : 4 objets suivants

- **un ensemble fini de symboles appelés symboles terminaux**
- **un ensemble fini de symboles appelés non terminaux**
- **un élément de l'ensemble des non-terminaux, appelé axiome**
- **un ensemble de appelés règles de production, qui sont des paires formées d'une non-terminal et d'une suite de terminaux et de non-terminaux**

(axiome : postulat, proposition évidente, vérité indémontrable, qui est admise)

Hiérarchie de Chomsky

Quatre types de grammaire

- **type 0 (libre) : aucune restriction dans les règles**
- **type 1 (en contexte) : règles de la forme**

  $R_i : \alpha I \beta \rightarrow \alpha x \beta$

  ***alpha* et *beta* sont le contexte (terminaux ou non-terminaux)**
- **type 2 (algébrique, ou hors-contexte) : règles de la forme**

  $R_i : I \rightarrow x$

  pas de notion de contexte (*x* terminal ou non)
- **type 3 (régulière) : règles de la forme**

  $R_i : \alpha \rightarrow z\beta | z$

Forme normale de Chomsky (CNF)

toute grammaire hors-contexte peut être convertie en CNF

$R_i : A \rightarrow AB | z$

Pour approfondir

Duda, Hart, Stork, « Pattern Classification », 2ème édition, Wiley-Interscience, 2001.
http://rii.ricoh.com/~stork/DHS.html

Boyer, Moore, « A fast string searching algorithm », Communications of the ACM. 20:762-772, 1977.
http://download-book.net/quinlan-c4.5-pdf-doc.html