

## CHAPITRE 4 : RECHERCHE DE MOTIFS

Il existe deux grandes classes d'applications des automates finis :

- recherche de motifs dans un texte (fonction *search* d'un traitement de texte)
- analyse lexicale dans la compilation des langages de programmation.

La suite du cours présente ces 2 types d'applications.

### 1. Recherche de motif dans un texte

#### 1.1 Présentation du problème

**Problème.** Rechercher toutes les occurrences d'un motif  $x$  dans un texte  $t$ . On généralisera ensuite à la recherche de plusieurs motifs  $x_1, \dots, x_n$ . Dans un chapitre ultérieur, on envisagera la recherche d'une expression régulière  $r$  (comme avec la commande *egrep* d'Unix).

Ce problème est omniprésent en informatique : éditeur de texte, moteur de recherche, analyse de séquences biologiques, imagerie informatique, composition musicale automatique, etc.

Il suscite encore de nombreuses recherches, à la fois du point de vue pratique (pour optimiser les méthodes), et du point de vue théorique.

#### 1.2 Tableau des différentes méthodes

Les méthodes de recherche de motif peuvent être classées en deux familles :

- 1- celles où le motif  $x$  est fixe,
- 2- celles où c'est le texte  $t$  qui est fixe.

Dans le cas n° 1 (motif fixe), il existe deux approches :

- on compare le motif au texte depuis le début du motif (gauche-droite)
- on fait la comparaison à partir de la fin du motif (droite-gauche).

Attention : la comparaison avec le texte, elle, se fait toujours de la gauche vers la droite (à partir du début du texte).

Exemple : recherche de  $x = bacbb$  dans  $t = abacbacbbba$

Lecture gauche-droite :

$a \ b \ a \ c \ b \ a \ c \ b \ b \ a$

**b**  échec

$b \ a \ c \ b \ b$  échec

$b$  échec

$b$  échec

$b \ a \ c \ b \ b$  = OK

Nombre de comparaisons de lettres ?

= 13 comparaisons

Lecture droite-gauche ( $x$  lu à partir de la fin) :

$a \ b \ a \ c \ b \ a \ c \ b \ b \ a$

$b \ b$  échec

$b$  échec

$b$  échec

$b \ b$  échec

$b \ a \ c \ b \ b$  = OK

Nombre de comparaisons de lettres ?

= 11 comparaisons

Ici la deuxième méthode est meilleure, mais ce n'est pas toujours le cas.

Les deux méthodes peuvent être optimisées : au lieu d'essayer toutes les positions pour  $x$ , on peut effectuer certains sauts.

Exemple : Lecture gauche-droite optimisée par un saut

$a \ b \ a \ c \ b \ a \ c \ b \ b \ a$

$b$  échec

$b \ a \ c \ b \ b$  échec

-> pour lire  $x$  plus loin, il faut commencer par un  $b$ , donc inutile d'essayer pour les  $a$  et  $c$  qu'on vient de lire : on saute directement jusqu'au  $b$  suivant

$b \ a \ c \ b \ b$  = OK

Nombre de comparaisons de lettres ?

= 11 comparaisons

**Méthode de Morris & Pratt (lecture gauche-droite avec sauts) :**

- On construit l'AFD qui reconnaît  $\Sigma^*x$
- On lit le texte  $t$  avec l'AFD : chaque fois qu'on passe par un état final, c'est qu'on a trouvé une occurrence de  $x$

Les transitions de l'AFD indiquent les sauts à effectuer.

Coût de la recherche = coût de la construction + coût de la lecture

Dans le cas n° 1, les deux approches gauche-droite et droite-gauche se traduisent par deux fameux algorithmes :

- Knuth, Morris & Pratt : c'est une variante de Morris & Pratt, c'est-à-dire avec construction de l'AFD pour  $\Sigma^*x$ . Dans ce cas, le texte  $t$  est comparé à  $x$  en commençant par le début de  $x$  (état initial de l'AFD). Les échecs se traduisent par le passage par certaines transitions de l'AFD qui reviennent à décaler  $x$  dans le texte (voir plus loin).
- Boyer & Moore : la comparaison entre  $x$  et  $t$  se fait en commençant par le fin de  $x$  (de droite à gauche), et s'il y a échec, on décale  $x$  dans le texte avec un saut judicieusement choisi.

Dans le cas n° 2 (texte fixe), la méthode générale est basée sur l'AFD reconnaissant les suffixes de  $t$ . Cet AFD ne doit pas nécessairement être complet, mais il doit être accessible et co-accessible (tout état doit pouvoir être atteint de l'état initial, et de tout état, on doit pouvoir atteindre un état final). On lit  $x$  dans l'AFD : si on peut le lire intégralement, c'est qu'on a lu un préfixe d'un suffixe de  $t$ , donc une occurrence de  $x$  dans  $t$ .

	Lecture gauche-droite	Lecture droite-gauche
Motif $x$ fixe	Morris & Pratt = AFD de $\Sigma^*x$ Knuth, Morris & Pratt (variante de la fonction d'échec)	AFD des suffixes du miroir $x\sim$ Boyer & Moore (variante)
Texte $t$ fixe	AFD des suffixes de $t$	

## 2. Recherche d'un motif fixe par lecture gauche-droite

### 2.1 Méthode de l'AFD de $\Sigma^*x$ (Morris & Pratt)

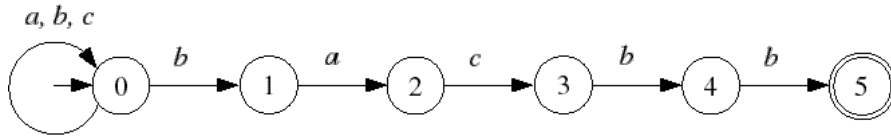
L'écorché d'un mot  $x$  de longueur  $n$  est l'AFD obtenu avec  $n + 1$  états, et  $n$  flèches correspondant aux lettres successives du mot. Dans cet automate, on voit clairement que les états ont un rôle de « mémoire » : chaque état mémorise la position dans le mot  $x$ .

Pour reconnaître  $\Sigma^*x$ , il suffit de rajouter une boucle sur l'état initial avec toutes les lettres de l'alphabet. On obtient un AFN.

On a vu que la détermination d'un AFN pouvait être très coûteuse ( $2^n$  états pour un AFN ayant  $n$  états). Il est remarquable que dans le cas de  $\Sigma^*x$ , le coût de la détermination soit faible.

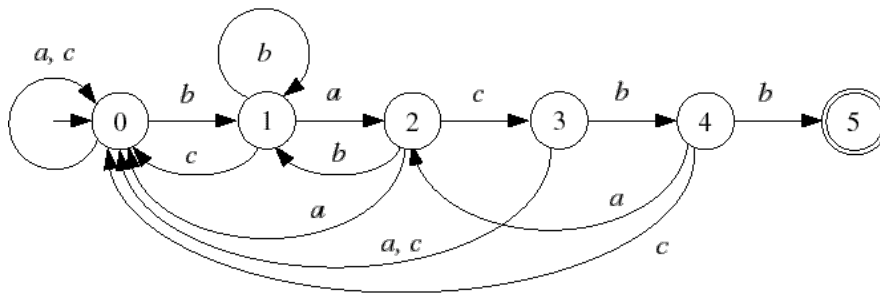
**Proposition.** L'AFD obtenu en déterminisant l'écorché d'un mot  $x$  de longueur  $n$ , augmenté d'une boucle sur l'état initial avec toutes les lettres, a au plus  $n$  états.

Exemple :  $x = bacbb$



Après détermination, on obtient l'AFD suivant. La correspondance entre les états de l'AFN et ceux de l'AFD est la suivante (0 est inchangé) :

- 1 = {0, 1}
- 2 = {0, 2}
- 3 = {0, 3}
- 4 = {0, 1, 4}
- 5 = {0, 1, 5}



Méthode de recherche : on lit le texte  $t$  dans l'AFD. Cela revient à le comparer à  $x$  en commençant par le début de  $x$  (état initial de l'AFD). Les échecs se traduisent par le passage par certaines transitions de l'AFD qui reviennent à décaler  $x$  dans le texte.

Exemple : recherche de  $x = bacbb$ , dans  $t = abacbcbba$

On indique les états d'arrivée après lecture de chaque lettre de  $t$ . Les états donnant un échec sont en gras.

a b a c b a c b b a

**0**

on reste à l'état 0 comme si on n'avait rien lu, donc cela revient à recommencer la lecture en se décalant d'une lettre

a b a c b a c b b a

0 1 2 3 4 **2** échec

on ne peut aller jusqu'à l'état final 5, donc échec : on revient à l'état 2,

cela revient à recommencer la lecture comme si on avait déjà lu  $ba$   
 donc à se décaler avec un saut comme on l'a vu plus haut.

$a \ b \ a \ c \ \boxed{b \ a \ c \ b \ b} \ a$   
 $0 \ 1 \ 2 \ 3 \ 4 \ 2 \ 3 \ 4 \ \underline{5} = \text{OK (5 est terminal)}$

## 2.2 Fonction d'échec : algorithme de Morris & Pratt

On peut calculer directement l'AFD pour  $\Sigma^*x$ , sans passer par la détermination de l'AFN, grâce à une fonction d'échec.

Si  $p$  est un état de l'écorché de  $x$  (autre que 0) permettant de lire  $w$ , on définit  $f(p)$  de la manière suivante :

$f(p)$  = état d'arrivée du plus long suffixe propre de  $w$  qui est aussi préfixe de  $w$  (donc de  $x$ )

Pour calculer facilement  $f(p)$ , il faut voir si on peut le déduire de  $f(p-1)$ . Il se trouve que ça marche.

On note le motif  $x = x(1) \dots x(n)$

On suppose construits  $f(1), f(2) \dots f(p-1)$  et on cherche  $f(p)$ .

$i = f(p-1)$



- si la lettre  $x(i+1) = x(p)$

alors  $x(1) \dots x(i+1)$  est à la fois préfixe et suffixe,

donc on peut prolonger le suffixe précédent :  $\boxed{f(p) = i + 1 = f(p-1) + 1}$

- si la lettre est différente, il faut essayer un suffixe plus court, lequel ?

$i = ?$



=> ça doit être un suffixe du suffixe précédent :

donc on réapplique  $f$



$$i = f(f(p-1))$$

et on recommence la comparaison entre  $x(i+1)$  et  $x(p)$

calcul de la fonction d'échec  $f$

```

f(0) ← -1
i ← -1
pour p = 1 à n
    tant que i ≥ 0 et x(i+1) ≠ x(p)
        i ← f(i)
    i ← i+1
    f(p) ← i

```

Exemple : Calcul de  $f$  pour le motif  $x = bacbb$

$$f(0) = -1, i = -1$$

$p = 1$   $i = -1$ , on incrémente  $i = 0$ , et pose  $f(1) = 0$

$p = 2$   $i = 0, x(1) = b \neq x(2) = a$ , on recule  $i = f(0) = -1$   
on incrémente  $i = 0$ , et on pose  $f(2) = 0$

$p = 3$   $i = 0, x(1) = b \neq x(3) = c$ , on recule  $i = f(0) = -1$   
on incrémente  $i = 0$ , et on pose  $f(3) = 0$

$p = 4$   $i = 0, x(1) = b = x(4)$ , on incrémente  $i = 1$ , et on pose  $f(4) = 1$

$p = 5$   $i = 1, x(2) = a \neq x(5) = b$ , on recule  $i = f(1) = 0$   
on incrémente  $i = 1$ , et on pose  $f(5) = 1$

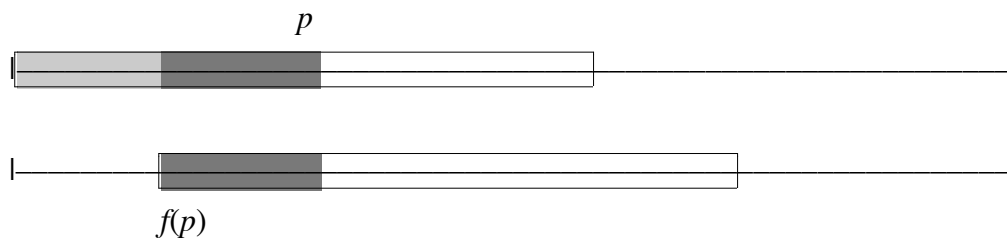
D'où la fonction d'échec :

État $p$	0	1	2	3	4	5
$f(p)$		0	0	0	1	1

Exemple :

- pour  $p = 2$ , le mot lu est  $ba$ , il n'a pas de suffixe propre qui soit préfixe, donc  $f(2) = 0$
- pour  $p = 4$ , le mot lu est  $bach$ , le plus long suffixe propre qui soit préfixe est  $b$ , donc  $f(4) = 1$
- pour  $p = 5$ , le mot lu est  $bacbb$ , le plus long suffixe propre qui soit préfixe est  $b$ , donc  $f(5) = 1$

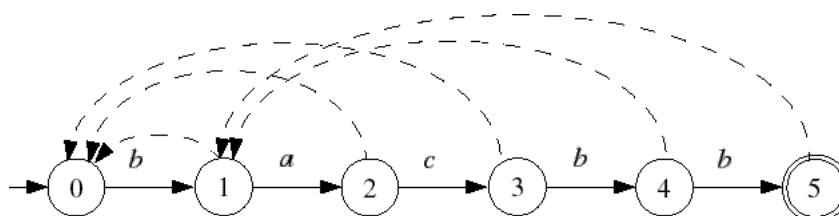
La fonction d'échec s'utilise de la manière suivante : on lit le texte  $t$  directement dans l'écorché de  $x$  (AFD). Quand dans un état donné  $p$  on ne peut pas lire la lettre courante du texte  $t$ , on réessaie à partir de  $f(p)$ , ou 0 si  $p = 0$ . Si  $w$  est le mot lu en arrivant à l'état  $p$  ( $w =$  préfixe de  $x$ ), alors par définition de la fonction d'échec, celui lu en arrivant à l'état  $f(p)$  est un suffixe de  $w$  qui est aussi préfixe de  $w$ , donc du motif  $x$ . Donc c'est un début possible pour une occurrence de  $x$ .



=> Si on veut essayer une position de la fenêtre qui chevauche la partie lue de gauche à droite, il faut qu'un suffixe de cette partie lue soit préfixe de  $x$ .

Exemple : reprenons le motif  $x = bacbb$ , et la recherche dans  $t = abacbacbba$

AFD écorché de  $x$  avec fonction d'échec  $f$ :



a b a c b a c b b a

0

dans l'état 0 de l'écorché de  $x = bacbb$ , on ne peut pas lire  $a$ , donc on reste en 0

a b a c b a c b b a

0 1 2 3 4 échec

dans l'état 4, on ne peut pas lire  $a$ ,

donc on revient à l'état  $1 = f(4)$  qui correspond à  $b =$  préfixe et suffixe de  $bach$

puis on lit le  $a$  qui fait passer en 2

$a \ b \ a \ c \ \boxed{b \ a \ c \ b \ b} \ a$   
 0 1 2 3 4 2 3 4 5 = OK (5 est terminal)

En fait, l'algorithme de Morris & Pratt ne construit pas explicitement l'AFD de  $\Sigma^*x$ . Comme on l'a vu dans l'exemple étudié ci-dessus, il se contente de lire le texte  $t$  en avançant dans l'écorché de  $x$  (AFD), et en effectuant les retours arrière définis par la fonction d'échec. Il fonctionne de la manière suivante, en supposant données

- la fonction de transition  $\delta(p, a)$  de l'écorché de  $x$ ,
- la fonction d'échec  $f(p)$ .

```

p ← 0 (état initial)
tant que t non vide
  avancer dans t, a ← 1ère lettre de t
  tant que p ≠ 0 et  $\delta(p, a)$  non défini
    p ← f(p)
  si  $\delta(p, a)$  défini alors p ←  $\delta(p, a)$ 
  si p ∈ F alors signaler occurrence de x dans t
  
```

Une variante de cet algorithme est l'algorithme de Knuth, Morris & Pratt. Il consiste à optimiser la fonction d'échec (voir TD).

### 2.3 Construction directe de l'AFD de $\Sigma^*x$ avec la fonction d'échec

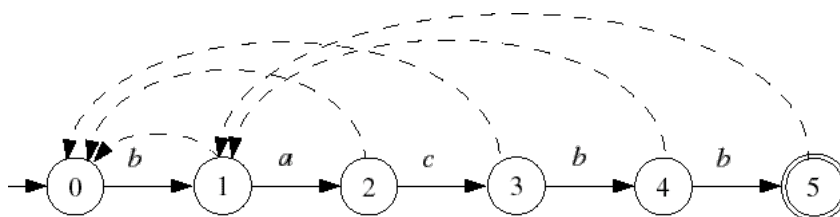
Avec la fonction d'échec, on peut aussi reconstruire directement l'AFD reconnaissant  $\Sigma^*x$ . Dans l'exemple ci-dessus, après échec en 4, on reprend en  $1 = f(4)$  pour pouvoir lire  $a$  (de l'état 1 à 2) après avoir lu le préfixe  $b$ . Dans l'AFD de  $\Sigma^*x$ , cela correspond exactement à la transition de 1 vers 2 étiquetée par  $a$  (voir dessin de l'AFD).

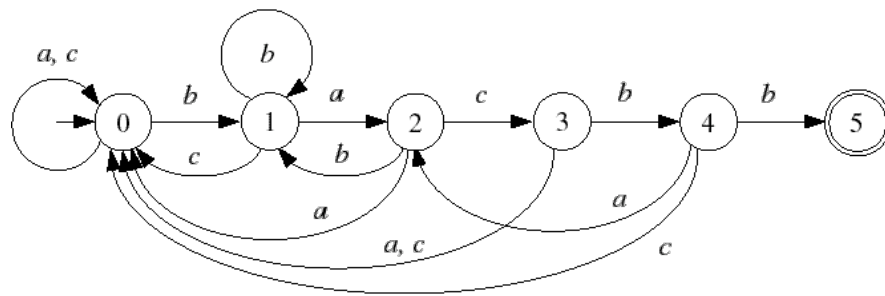
On complète la fonction de transition de la manière suivante :

pour toute lettre  $a$  telle que  $\delta(p, a)$  non défini, on pose

- $\delta(p, a) = \delta(f(p), a)$  si  $p \neq 0$ ,
- $\delta(0, a) = 0$ .

On vérifie facilement qu'en rajoutant ces transitions, on obtient exactement l'AFD déterminisé vu plus haut.





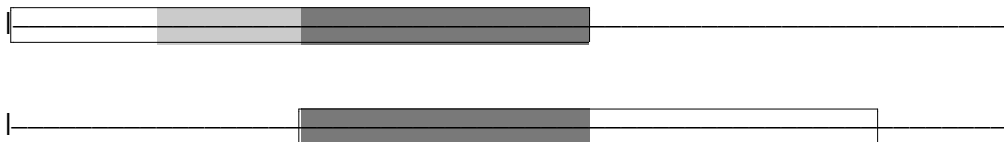
### 3. Recherche d'un motif fixe par lecture droite-gauche

#### 3.1 Méthode de l'AFD des suffixes du motif miroir

Le miroir  $x^\sim$  d'un mot  $x$  est le mot lu à l'envers (en commençant par la fin).

#### Méthode :

- On construit l'AFD qui reconnaît les suffixes du mot miroir  $x^\sim$
- On lit le texte  $t$  dans l'AFD par une fenêtre de longueur  $|x|$ , dans laquelle on lit de droite à gauche : en cas d'échec, on revient au dernier état final de l'AFD rencontré. Il correspond à un suffixe  $s$  de  $x^\sim$ . Donc de gauche à droite,  $s^\sim$  est un préfixe de  $x$ . On se décale dans  $t$  pour commencer une nouvelle lecture à partir de  $s^\sim$ . Ces sauts accélèrent considérablement la recherche.



=> Si on veut essayer une position de la fenêtre qui chevauche la partie lue de droite à gauche, il faut prendre le plus grand suffixe de cette partie lue (donné par un état final de l'AFD) qui soit préfixe de  $x$ .

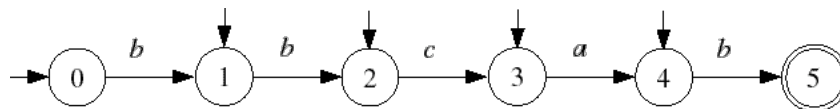
On notera qu'aucune position de la fenêtre située avant le début de ce suffixe (en particulier avant la partie chevauchante) ne peut convenir, car si on pouvait lire  $x$  en entier de cette position, on aurait lu un plus long suffixe dans la partie lue de droite à gauche.



Exemple : motif  $x = bacbb$

AFN pour les suffixes de  $x^\sim = bbcab$

On part de l'écorché de  $x\sim$ , et on rend initiaux tous les états sauf le dernier.

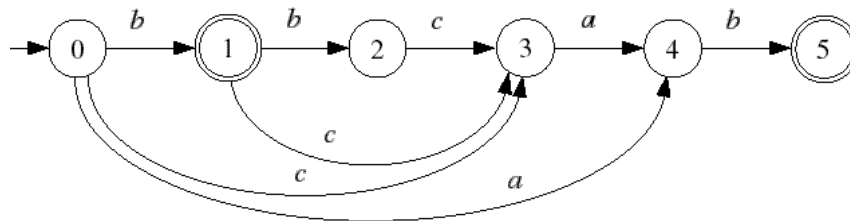


Après détermination, on obtient l'AFD suivant. La correspondance entre les états de l'AFN et ceux de l'AFD est :

0 = {0, 1, 2, 3, 4}

1 = {1, 2, 5}

2, 3, 4, 5 sont identiques



Recherche de  $x = bacbb$ , dans  $t = abacbcbba$

a b a c b a a c b b a  
 5 4 3 1

dans l'état 5, on ne peut lire  $a$ ,

5 étant terminal, il correspond au suffixe  $bcab$  de  $x\sim = bbcab$ ,

donc on reprend la lecture avec son miroir  $bacb$ , qui est préfixe de  $x$   
 ce qui revient à se décaler d'une lettre

a b a c b a c b b a  
 5 4

dans l'état 5, on ne peut lire  $c$ ,

5 étant terminal, il correspond au suffixe  $ab$  de  $x\sim = bbcab$ ,

donc on reprend la lecture avec son miroir  $ba$ , qui est préfixe de  $x$   
 ce qui revient à se décaler de trois lettres

a b a c b a c b b a  
 5 4 3 2 1 = OK (on a tout lu et 5 est terminal)

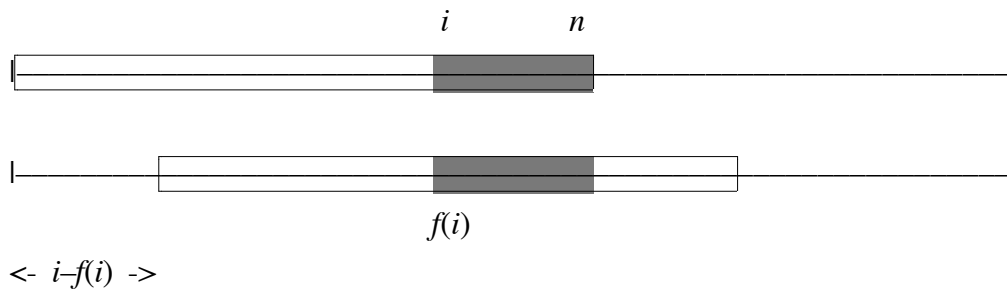
### 3.2 Variante : algorithme de Boyer & Moore

Dans la méthode précédente avec l'AFD des suffixes du miroir  $x\sim$ , en cas d'échec dans la lecture droite-gauche, l'AFD permet de trouver un préfixe du mot qu'on a lu qui est suffixe de  $x\sim$ , c'est-à-dire dont le miroir est préfixe de  $x$ . On se décale pour recommencer la lecture à partir de ce préfixe de  $x$ , qui peut être un début possible pour une nouvelle occurrence.

Dans l'algorithme de Boyer & Moore, qui est une variante, mais sans construire l'AFD précédent, on ne repart pas d'un préfixe plus court que le mot lu dans la fenêtre droite-gauche (donné par l'AFD), mais on utilise ce mot complet (on note  $w$  son miroir, dans l'ordre normal gauche-droite) :

- 1) on numérote les positions dans la fenêtre de 1 à  $n$  (où  $n$  est la longueur du motif  $x$ ) dans l'ordre normal gauche-droite,
- 2) on construit une fonction  $f(i)$  qui donne la position la plus à droite où le suffixe de  $x$  qu'on a lu dans  $t$ , noté  $w = x(i)\dots x(n)$  entre les positions  $i$  et  $n$  de la fenêtre, réapparaît à gauche dans le motif  $x$  (s'il réapparaît en plusieurs endroits, c'est le premier en partant de la droite qui est pris en compte),
- 3) on se décale dans le texte  $t$  pour faire coïncider la fenêtre avec cette occurrence de  $w$ , soit  $i-f(i)$  lettres.

S'il y a eu échec sans lire un suffixe de  $x$  (dès la première lettre), on se décale d'une lettre dans  $t$ .



Exemple : recherche de  $x = baacaa$ , dans  $t = aaabaacaa$

On calcule la fonction  $f$ , en attribuant la valeur 0 aux suffixes qui ne réapparaissent pas plus à gauche dans le mot.

Position $i$	1	2	3	4	5	6
$f(i)$	0	0	0	0	2	5

$a a a b a a c a a$

$c a a$

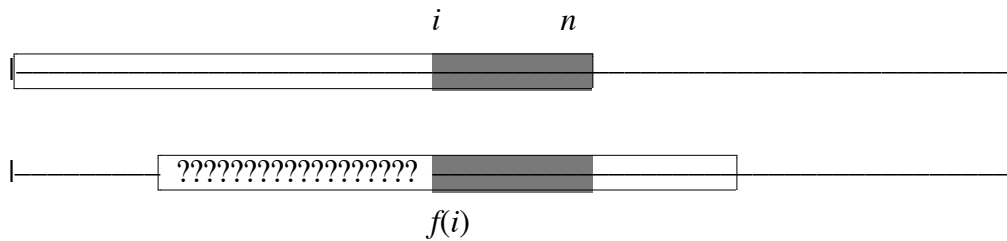
5 = échec de la lecture du  $c$  attendu au lieu de  $b$ , suffixe lu =  $aa$

on décale la fenêtre pour la faire coïncider avec le  $aa$  trouvé, en utilisant  $f(5) = 2$

$b a a c a a$  = OK occurrence trouvée

2

La différence avec l'AFD des suffixes de  $x$  est que le décalage ne garantit pas qu'on repart d'un début de  $x$ . Avant l'occurrence du suffixe mise en coïncidence avec une portion du motif, il peut y avoir des lettres divergentes qui font échec.



Exemple : Avec le même motif  $x = baacaa$ , si le texte avait été  $t = aaadaacaa$  au lieu de  $aaabaacaa$  (avec  $d$  à la place de  $b$ ), le décalage aurait donné un échec.

En fait, dans l'algorithme de Boyer & Moore, on utilise aussi le caractère qui fait l'échec (ici le  $b$  trouvé à la place du  $c$ ) pour améliorer le décalage. Cela ne change pas le calcul ci-dessus dans le premier exemple. Mais dans le deuxième avec  $d$  à la place de  $b$ , cela aurait évité le décalage donnant un échec.

L'algorithme tel qu'il est décrit ici est inopérant dans les cas où l'échec se produit dès la première lettre, sans lecture dans la fenêtre droite-gauche : dans ce cas, on se décale d'une seule lettre. Dans le véritable algorithme de Boyer & Moore, l'utilisation de la lettre d'échec permet d'améliorer cette situation (voir TD).

Exemple : reprenons l'exemple étudié au début du chapitre avec  $x = bacbb$  et  $t = abacbacbbba$

Il correspond à un cas défavorable de cette version simplifiée de Boyer & Moore, car on ne peut jamais obtenir un décalage de plus d'une seule lettre.

Position $i$	1	2	3	4	5
$f(i)$	0	0	0	0	4

$a \ b \ a \ c \ b \ a \ c \ b \ b \ a$

b b

5 = échec de la lecture du  $b$  attendu au lieu de  $c$ , suffixe lu =  $b$

on décale la fenêtre pour la faire coïncider avec le  $b$  trouvé, en utilisant  $f(5) = 4$

mais cela revient à ne faire un décalage que d'une seule lettre

(b) b

4

échec dès la première lettre, pas de suffixe lu, donc décalage d'une seule lettre

b

b b

5 = échec de la lecture du  $b$  attendu au lieu de  $c$ , suffixe lu =  $b$

on décale la fenêtre pour la faire coïncider avec le  $b$  trouvé, en utilisant  $f(5) = 4$  mais cela revient à ne faire un décalage que d'une seule lettre

$$\boxed{b \ a \ c \ b \ b} = \text{OK occurrence trouvée}$$

4

#### 4. Remarques complémentaires

##### 4.1 Recherche d'un ensemble fini $X$ de motifs

On a un ensemble de motifs  $X = \{x_1, \dots, x_k\}$ .

Les deux principales méthodes ci-dessus se généralisent à ce cas :

- 1) Construction de l'AFD pour  $\Sigma^* X$
- 2) Construction de l'AFD pour l'ensemble des suffixes des miroirs  $X^\sim$

##### 4.2 Recherche d'une expression régulière $r$

On construit un AFN qui reconnaît  $\Sigma^* r$  (voir l'algorithme de Thompson au chapitre suivant).

##### 4.3 Comparaison des deux méthodes : Morris & Pratt, suffixes du miroir

La méthode de l'AFD de  $\Sigma^* x$  avec une représentation de l'AFD par une table de transition, a un coût de construction en  $O(|x||\Sigma|)$ , et un coût pour la lecture de  $t$  en  $O(|t|)$ .

Si l'on ne construit pas effectivement l'AFD, mais qu'on construit seulement la fonction d'échec, le coût de la construction est en  $O(|x|)$ , et celui de l'analyse en  $O(|t|\log|\Sigma|)$ .

Dans l'exemple traité ci-dessus, la méthode de Morris & Pratt (AFD de  $\Sigma^* x$ ) donnait les mêmes sauts que celle de l'AFD des suffixes de  $x^\sim$  (trois positions de la fenêtre dans les deux cas) Mais ce n'est pas vrai dans le cas général. Voici un exemple qui montre les différences entre les deux méthodes.

Exemple : recherche de  $x = bbabbabb$ , dans  $t = abaabbabbabb$

##### 1) méthode de Morris & Pratt

Position	1	2	3	4	5	6	7	8
$f(i)$	0	1	0	1	2	3	4	5

$\boxed{a \ b \ b \ b \ b \ a \ b \ b} \ a \ b \ b$

0 échec dans la lecture :  $t$  contient  $a$ , alors qu'il faut un  $b$   
on se décale d'une lettre

$a$   $\boxed{b\ b\ b\ b\ a\ b\ b}$   $a\ b\ b$

1 2 échec dans la lecture :  $t$  contient  $b$ , alors qu'il faut un  $a$

1 on reprend à l'état  $1 = f(2)$

$a\ b$   $\boxed{b\ b\ b\ a\ b\ b\ a\ b}$   $b$

1 2 échec dans la lecture :  $t$  contient  $b$ , alors qu'il faut un  $a$

1 on reprend à l'état  $1 = f(2)$

$a\ b\ b$   $\boxed{b\ b\ a\ b\ b\ a\ b\ b}$

1 2 3 4 5 6 7 8 occurrence de  $x$  trouvée

Dans cet exemple, la fonction de Morris & Pratt ne permet d'éviter aucun saut : on essaie toutes les positions de la fenêtre jusqu'à l'occurrence de  $x$ .

2) méthode de l'AFD de  $x\sim$

Dans ce cas,  $x$  est identique à son miroir, on a :  $x\sim = x = bbabbabb$ .

$\boxed{a\ b\ b\ b\ b\ a\ b\ b}$   $a\ b\ b$

$b\ b\ a\ b\ b$  échec dans la lecture droite-gauche

Le principe est de repartir du plus long préfixe  $w$  de  $x$  lu dans la fenêtre droite-gauche (c'est-à-dire tel que  $w\sim$  est suffixe de  $x\sim$ ). Ici  $w = bbabb$  est le plus long préfixe trouvé, donc on décale la fenêtre pour la faire coïncider avec  $bbabb$ .

$a\ b\ b$   $\boxed{b\ b\ a\ b\ b\ a\ b\ b}$

occurrence de  $x$  trouvée

Dans ce cas, on ne fait qu'un seul saut, et le calcul est bien meilleur qu'avec Morris & Pratt.