

1. *Algorithmes utilisant des automates*
  - Recherche de motifs.
  - Recherche de régularités.
  - Compression.
  
2. *Algorithmes pour l'étude des automates*
  - Complexité d'états :  
coût du passage d'un modèle à un autre.
  - Automates et variétés de langages :  
tests de propriétés du langage reconnu.

## Recherche de motifs

On cherche toutes les occurrences

d'un mot  $x$  (le **motif**) de longueur  $m$   
dans un mot  $t$  (le **texte**) de longueur  $n$ .

### Recherche naïve

pour  $j$  de 1 à  $n-m+1$

comparer  $x[1..m]$  à  $t[j..j+m-1]$

**Proposition** Algorithme naïf:

- Espace supplémentaire  $O(1)$ .
- Temps:
  - maximal  $m(n - m + 1)$ .
  - en moyenne  $\leq 2n$  si  $|A| \geq 2$ .

Traitement préalable du motif ou du texte  
 $\implies$  accélération de la recherche.

## Périodes et bords

Une période de  $u \in A^+$  : un entier  $p$  tq.

$$\begin{cases} 0 < p \leq |u| \\ u_i = u_{i+p} & \text{si } i, i+p \in [1, |u|] \end{cases}$$

La période  $\text{Per}(u) =$  la plus petite période.

Un bord de  $u \in A^+$  : un mot  $v \in A^*$  tq.

$$\begin{cases} v \neq u \\ \exists z, t \in A^*, u = vz = tv \end{cases}$$

Le bord  $\text{Bord}(u)$  de  $u =$  le plus grand bord.

**Proposition**  $u \in A^+, a \in A$  :

– Soit  $k = \min\{l \mid \text{Bord}^l(u) = \varepsilon\}$ . Les bords de  $u$  sont  $\text{Bord}(u), \text{Bord}^2(u), \dots, \text{Bord}^k(u)$ .

–  $\text{Bord}(ua) = \begin{cases} \text{Bord}(u)a & \text{si } \text{Bord}(u)a \leq_p u \\ \text{Bord}(\text{Bord}(u)a) & \text{sinon} \end{cases}$

**Proposition** Si  $u \in A^+$  et  $0 < p \leq |u|$ , tfae :

- (i)  $p$  est une période.
- (ii)  $\exists v \in A^+, k > 0$  tq  $|v| = p$  et  $u \leq_p v^k$ .
- (iii)  $\exists v \in A^+, k > 0$  tq  $|v| = p$  et  $u \leq_f v^k$ .
- (iv)  $u = (xy)^k x$  où  $|xy| = p$ ,  $y \neq \varepsilon$  et  $k > 0$ .
- (v)  $\exists x, y, z, |x| = |y| = p$  et  $u = xz = zy$ .

## L'automate de $A^*x$

$f_x(u) =$  plus long  $v$  tq  $v \leq_s u$  et  $v \leq_p x$ .

**Proposition** L'automate minimal de  $A^*x$  est  $\mathcal{A}(x) = (\text{Pref}(x), \varepsilon, x, \delta : (p, a) \mapsto f_x(pa))$

**Proposition** La fonction  $\delta$  se calcule par

$$p \cdot a = \begin{cases} pa & \text{si } pa \leq_p x \\ \text{Bord}(pa) = \text{Bord}(p) \cdot a & \text{sinon} \end{cases}$$

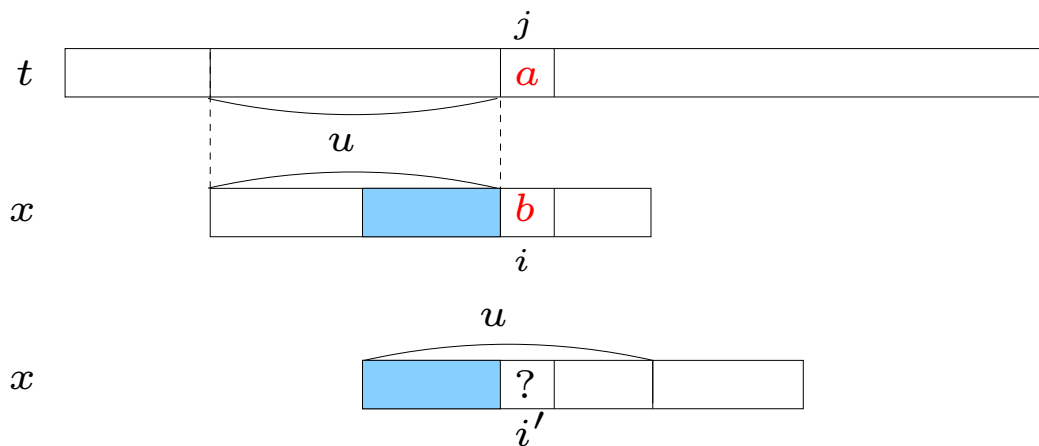
Construction de  $\mathcal{A}(x)$  :

- + en ligne,
- + temps de recherche  $O(n)$ ,
- temps de construction  $O((m+1)|A|)$ ,
- espace supplémentaire  $O((m+1)|A|)$ .

## Algorithme de Morris et Pratt

Si  $x[i - l] = t[j - l]$  ( $l \geq 1$ ) et  $x[i] \neq t[j]$   
 décaler le motif de  $i - 1 - |\text{Bord}[i - 1]|$ .

Représentation compacte de  $\mathcal{A}(x)$  : fonction de  
 suppléance (cf. fig 1).



procédure **MP**(x,t)

$i \leftarrow 1, j \leftarrow 1;$

tant que  $j \leq n$

    tant que  $i > m$  ou ( $i \geq 1$  et  $x[i] \neq t[j]$ )

$i \leftarrow \text{SuppMP}[i];$

$i \leftarrow i + 1;$

$j \leftarrow j + 1;$

    si  $i = m + 1$

        occurrence de x en  $j - m$

## Théorème

Le nombre de comparaisons  $x[i] \neq t[j]$  de la procédure MP est au plus  $2n - 1$ .

Calcul des bords basé sur

$$\text{Bord}(ua) = \begin{cases} \text{Bord}(u)a & \text{si } \text{Bord}(u)a \leq_p u \\ \text{Bord}(\text{Bord}(u)a) & \text{sinon} \end{cases}$$

procédure **Bord**(x, m)

Bord[0] ← -1;

pour j de 1 à m

    i ← Bord[j-1];

    tant que i ≥ 0 et x[i+1] ≠ x[j]

        i ← Bord[i];

    Bord[j] ← i+1;

## Algorithme de Morris et Pratt

Calcul de la fonction de suppléance SuppMP :

procédure **SuppMP**( $x, m$ )

SuppMP[1]  $\leftarrow$  0;  $i \leftarrow$  0;

pour  $j$  de 1 à  $m-1$

$i \leftarrow$  SuppMP[ $j$ ]; // en fait inutile

tant que  $i > 0$  et  $x[i] \neq x[j]$

$i \leftarrow$  SuppMP[ $i$ ];

$i \leftarrow i + 1$ ;

SuppMP[ $j+1$ ]  $\leftarrow$   $i$ ;

C'est l'algorithme **MP** ( $t = x, x = x[2..m]$ )

### Théorème

Le calcul de la fonction de suppléance fait au plus  $2(m-1) - 1 = 2m - 3$  comparaisons.

### Théorème

En ajoutant dans **MP** le test  $j - i + m \leq n$ ,

$$\text{Coût}(\text{MP}) = 2n - m$$

$$\text{Coût}(\text{MP} + \text{SuppMP}) = 2n + m - 3$$

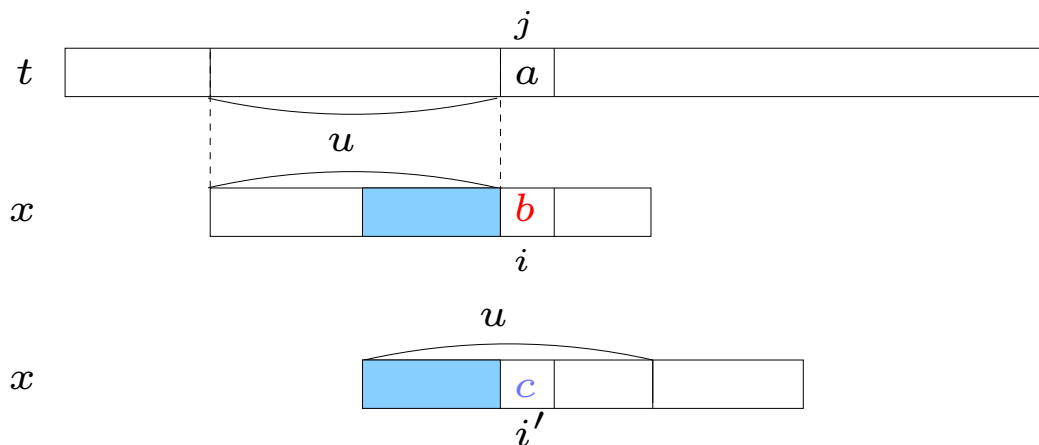
## Algorithme de Knuth, Morris et Pratt

Idée : supprimer les itérations pour lesquelles

$$x[i] = x[\text{SuppMP}(i)]$$

Nouvelle fonction de suppléance SuppKMP.

BordDisjoint[ $i - 1$ ] :  $b \neq c$ .



**Lemme** Si  $k = \text{SuppMP}[i]$ ,

$$\text{SuppKMP}[i] = \begin{cases} k & \text{si } x[i] \neq x[k] \text{ ou } i = m \\ \text{SuppKMP}[k] & \text{sinon} \end{cases}$$

## Algorithme de Knuth, Morris et Pratt

**Délai** : nombre maximal de comparaisons sur 1 caractère de  $t$ .

**Théorème** Le délai de l'algorithme de Knuth, Morris et Pratt est  $\lfloor \log_{\varphi}(m + 1) \rfloor$  où  $\varphi = \frac{1 + \sqrt{5}}{2}$  et cette borne est optimale.

**Lemme** (Fine & Wilf, 65)  $p, q \geq 1$  et  $d = \gcd(p, q)$ . Si  $u$  admet  $p$  et  $q$  pour périodes et si  $p + q - d \leq |u|$ , alors  $u$  admet  $d$  pour période.

**Corollaire** Si  $w$  est le bord strict de  $v$  et  
 $v$  est le bord strict de  $u$ ,  
alors  $|u| > |v| + |w| + 1$ .

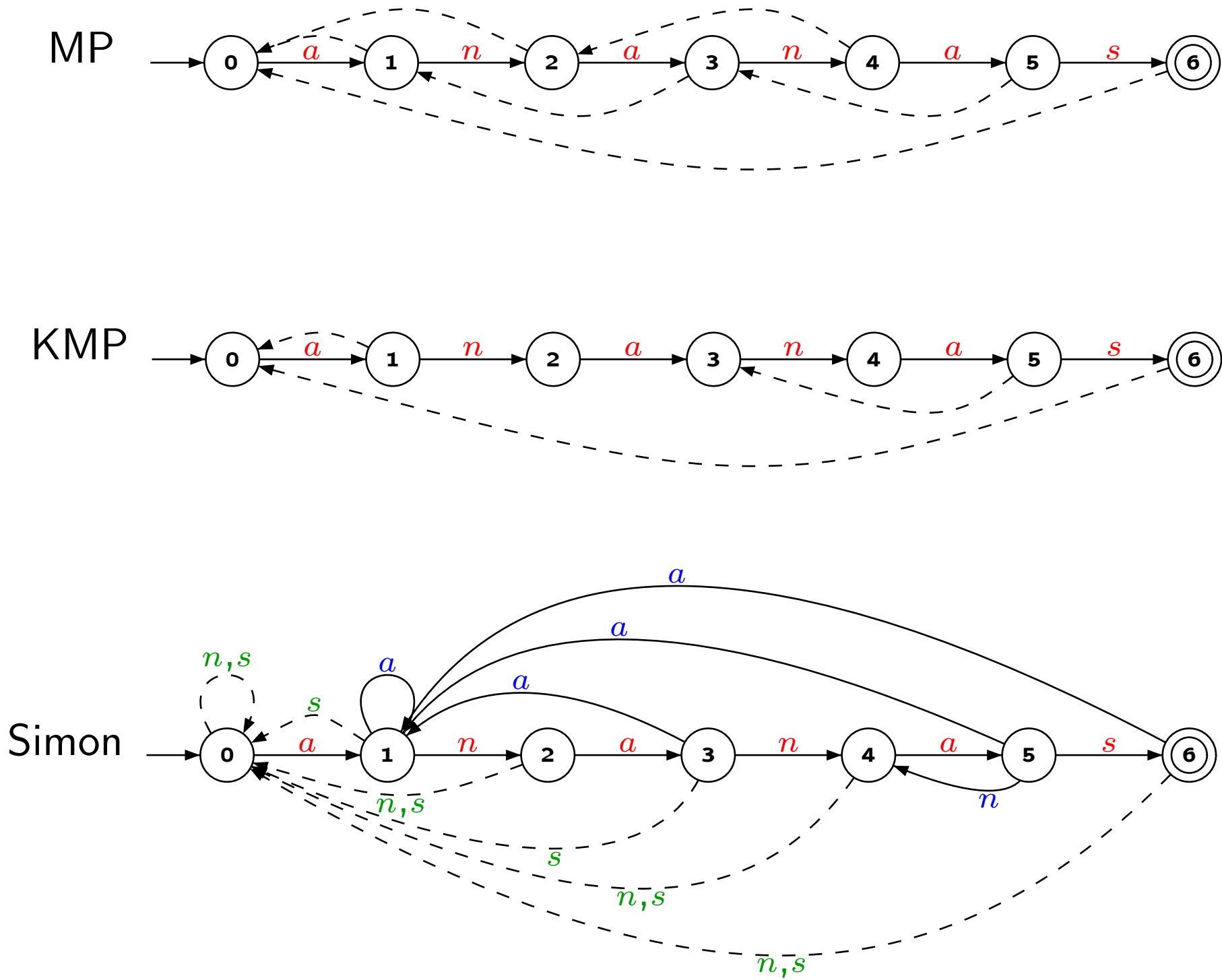
## Délai de KMP

Suite de Fibonacci:  $f_0 = \varepsilon$ ,  $f_1 = b$ ,  $f_2 = a$ ,  
 $f_{k+2} = f_{k+1}f_k$ .

### Lemme

- a.  $f_k \leq_p f_{k+1}$ ,
- b.  $|f_k| \wedge |f_{k+1}| = 1$ ,
- c. Si  $k \geq 3$ ,  $f_k = g_k h_k$ ,  $|h_k| = 2$ , alors  $h_n = ab$  si  $n \equiv 1 \pmod{2}$  et  $h_n = ba$  sinon.
- d. Si  $k \geq 6$ ,  $g_k = g_{k-1} h_{k-1} g_{k-2}$  et  $g_k = f_{k-2} g_{k-1}$ .  
En particulier,  $g_{k-1}$  est un bord disjoint de  $g_k$ .
- e. Si  $k \geq 6$ ,  $g_k = f_{k-2}^2 g_{k-2}$ . Le mot  $g_k \leq_p f_{k-2}^3$  et  $g_k \leq_p f_{k-1}^2$ .
- f.  $g_{k-1}$  est le bord (disjoint) de  $g_k$ .

Fig. 1: comparaison des 3 algorithmes.



## Algorithme de Simon

Idée : il y a peu de transitions significatives.

Flèche **avant** : mène de  $p$  à  $pa$ .

Flèche **arrière** : les autres ne menant pas sur  $\varepsilon$ .

**Proposition** Dans  $\mathcal{A}(x)$ , il y a au plus  $|x|$  flèches arrière.

On peut donc représenter l'automate de façon **compacte**, en oubliant toutes les transitions ramenant sur l'état initial.

On mémorise ainsi au plus  $2|x|$  transitions.

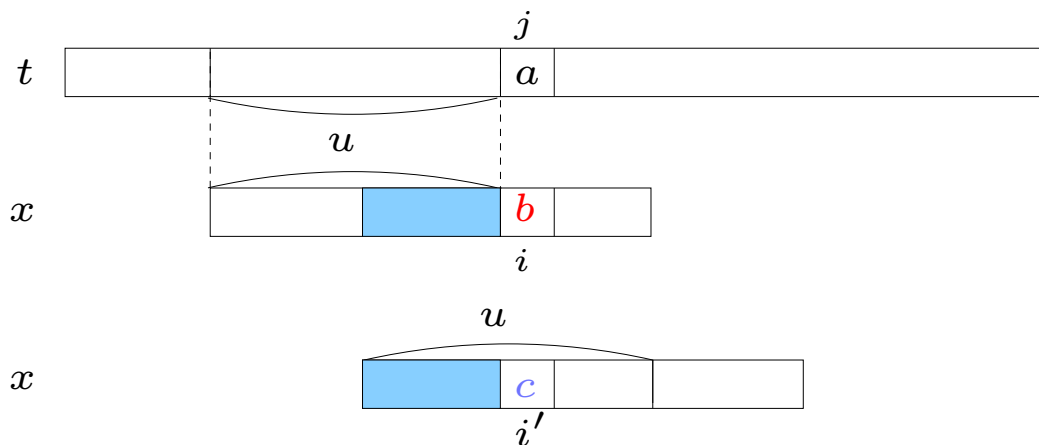
## Algorithme de Knuth, Morris et Pratt

Idée : supprimer les itérations pour lesquelles

$$x[i] = x[\text{SuppMP}(i)]$$

Nouvelle fonction de suppléance SuppKMP.

BordDisjoint[ $i - 1$ ] :  $b \neq c$ .



**Lemme** Si  $k = \text{SuppMP}[i]$ ,

$$\text{SuppKMP}[i] = \begin{cases} k & \text{si } x[i] \neq x[k] \text{ ou } i = m \\ \text{SuppKMP}[k] & \text{sinon} \end{cases}$$

## Algorithme de Knuth, Morris et Pratt

**Délai** : nombre maximal de comparaisons sur 1 caractère de  $t$ .

**Théorème** Le délai de l'algorithme KMP est  $\leq \lfloor \log_{\varphi}(m + 1) \rfloor$  où  $\varphi = \frac{1 + \sqrt{5}}{2}$ . Cette borne est optimale.

**Lemme** (Fine & Wilf, 65)  $p, q \geq 1$  et  $d = \gcd(p, q)$ . Si  $u$  admet  $p$  et  $q$  pour périodes et si  $p + q - d \leq |u|$ , alors  $u$  admet  $d$  pour période.

**Corollaire** Si  $w$  est le bord strict de  $v$  et  $v$  est le bord strict de  $u$ , alors  $|u| > |v| + |w| + 1$ .

## Délai de KMP

Suite de Fibonacci:  $f_0 = \varepsilon$ ,  $f_1 = b$ ,  $f_2 = a$ ,  
 $f_{k+2} = f_{k+1}f_k$ .

### Lemme

- $f_k \leq_p f_{k+1}$ ,
- $|f_k| \wedge |f_{k+1}| = 1$ ,
- Si  $k \geq 3$ ,  $f_k = g_k h_k$ ,  $|h_k| = 2$ , alors
$$\begin{cases} h_n = ab & \text{si } n \equiv 1 \pmod{2} \\ h_n = ba & \text{sinon.} \end{cases}$$
- Si  $k \geq 6$ ,  $g_k = g_{k-1} h_{k-1} g_{k-2}$  et
$$g_k = f_{k-2} g_{k-1}.$$
 $\Rightarrow g_{k-1}$  est *un* bord disjoint de  $g_k$ .
- Si  $k \geq 6$ ,  $g_k = f_{k-2}^2 g_{k-2}$ .  
Donc  $g_k \leq_p f_{k-2}^3$  et  $g_k \leq_p f_{k-1}^2$ .
- $g_{k-1}$  est *le* bord (disjoint) de  $g_k$ .

## Algorithme de Simon

Idée : peu de transitions significatives dans  $\mathcal{A}(x)$ .

Flèche **avant** : mène de  $p$  à  $pa$ .

Flèche **arrière** : les autres ne menant pas sur  $\varepsilon$ .

Flèche **significative** : avant ou arrière.

**Théorème**  $\mathcal{A}(x)$  a au plus  $|x|$  flèches arrière.

$p \rightarrow q, q \leq_p p$ .  $N_u(q \cdots p)$  : nb de flèches significatives dans  $\mathcal{A}(u)$  partant de  $r, q \leq_p r \leq_p p$

**Lemme**

$$N_{ua}(v) = \begin{cases} N_u(v) + 1(\text{Bord}(ua) = \varepsilon) & \text{si } v = u \\ N_u(\text{Bord}(v)) & \text{si } v = ua \\ N_u(v) & \text{sinon} \end{cases}$$

$$N_u(v) = \begin{cases} 1 & \text{si } v = \varepsilon \\ N_u(\text{Bord}(v)) & \text{si } v = u \\ N_u(\text{Bord}(v)) + \\ \quad 1(\text{Bord}(vu_{|v|+1}) = \varepsilon) & \text{sinon} \end{cases}$$

## Algorithme de Simon

Analyse du nombre de flèches significatives.  
Ch. Hancart, Inf. Proc. Letters **47**, 2 (1993)

### Théorème

$$N_u(q \cdots p) \leq 2|p| - 2|q| + 1(q \neq \varepsilon) + 1(p \neq u)$$

### Corollaire

$\mathcal{A}(x)$  a au plus  $2|x|$  flèches significatives. On peut les calculer en temps  $O(|x|)$ .

**Proposition** L'algorithme de Simon fait moins de comparaisons que l'algorithme KMP.

## Calcul du délai

**Lemme** ( $\varepsilon \neq v \leq_p u$  et  $2|\text{Bord}(v)| \geq |v|$ )

$\Downarrow$

$$N_u(\text{Bord}(v)) = N_u(\mathbf{B}^2(v))$$

**Proposition**  $v \leq_p u$

$\Downarrow$

$$N_u(v) \leq 1 + \log_2(\min(|v| + 1, |u|))$$

**Corollaire** Le délai de l'algorithme de Simon est au plus  $\min(1 + \lfloor \log_2(|x|) \rfloor, |A|)$ .

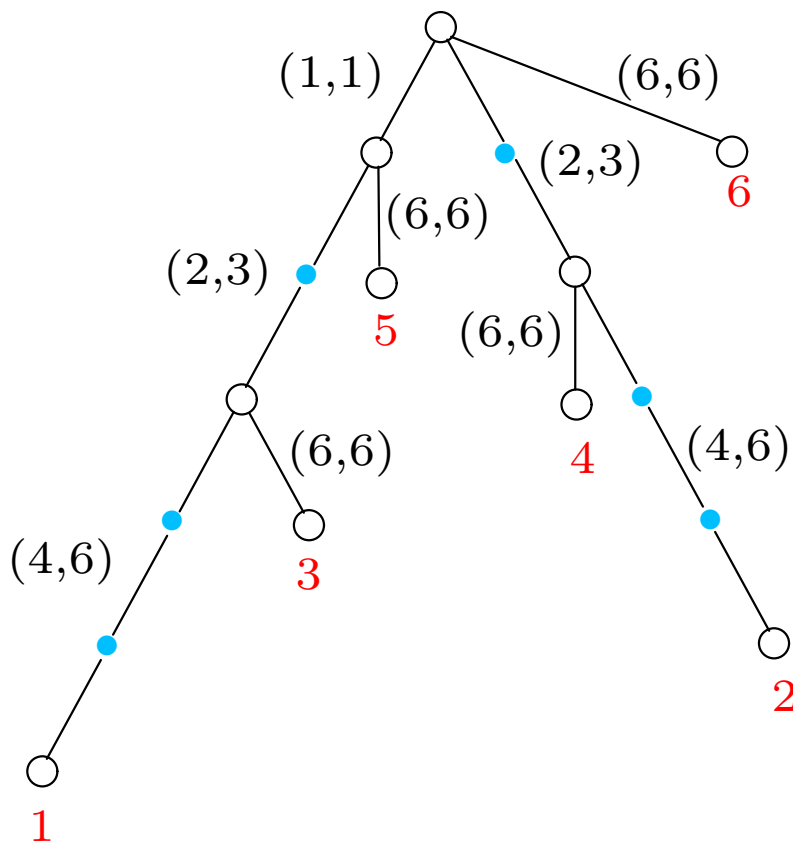


## Arbre des suffixes compressé

Compression de l'arbre précédent  $\rightarrow$  taille  $O(n)$

Une arête est étiquetée par un facteur de  $t$ .

$t_i \cdots t_j$  est représenté par le couple  $(i, j)$ .



Nœuds explicites = nœuds de l'arbre.

Nœuds implicites : positions bleues des arêtes.

$\rightarrow$  Ne sont pas des nœuds de l'arbre compressé.

$\rightarrow$  Correspondent à des nœuds de l'arbre non compressé.

## Construction directe de l'arbre compressé

$s_i$  :  $i^{\text{ème}}$  suffixe de  $t$  :  $s_i = t_i \cdots t_n$ .

**Principe** : insérer successivement  $s_1, s_2, \dots, s_n$

$T_i$  : arbre obtenu après insertion de  $s_i$ .

$\bar{x}$  : nœud implicite ou explicite associé à  $x \in A^*$

### Insertion de $s_{i+1}$ dans $T_i$

1. Recherche de  $\bar{h}_{i+1}$  tel que

$h_{i+1} =$  plus long préfixe de  $s_{i+1}$  dans  $T_i$ .

2. Si  $s_{i+1} = h_{i+1}q_{i+1}$ , création d'une nouvelle arête  $\bar{h}_{i+1} \xrightarrow{q_{i+1}} \bar{s}_{i+1}$ .

Chacune des étapes 2 prend un temps constant.

## Algorithme de McCreight

**But** : recherche de  $\bar{h}_2, \dots, \bar{h}_n$  en temps  $O(n)$

$\text{EI}(T) = \{\text{nœuds explicites internes de } T\}$

**Invariants (liens suffixes)**.  $a \in A$  dans la suite

(1) Tout nœud de  $\text{EI}(T_i) \setminus \{\bar{\varepsilon}\}$  est branchant.

(2) Si  $\bar{h}_i \neq \bar{ax} \in \text{EI}(T_i)$ , on a  $\text{Suf}(\bar{ax}) = \bar{x}$

Remarque :  $\bar{h}_i \neq \bar{ax} \in \text{EI}(T_i) \implies \bar{x} \in \text{EI}(T_i)$ .

(preuve : construction de  $T_{i+1}$  à partir de  $T_i$ )

### Lemme

1. Si  $h_i = ay_i$ ,  $a \in A$ , alors  $h_{i+1} = y_i z_{i+1}$

2. Si  $h_i = \varepsilon$ , soit  $z_{i+1} = h_{i+1}$ . On a alors

$$\sum_j |z_j| \leq n$$

**Corollaire** Si on peut trouver tous les  $\bar{y}_i$  en temps linéaire, on peut aussi trouver tous les  $\bar{h}_i$  en temps linéaire.

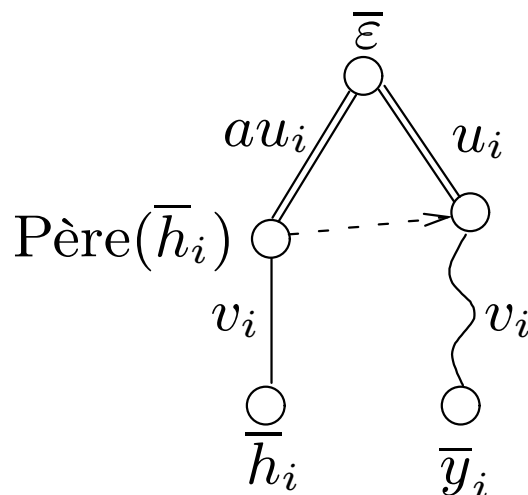
## Algorithme de McCreight

**But** : recherche de  $\bar{y}_2, \dots, \bar{y}_n$  en temps  $O(n)$ .

Si on avait un lien suffixe partant de  $\bar{h}_i$ , on accéderait à  $\bar{y}_i$  à partir de  $\bar{h}_i$  en temps constant.

Mais ce lien n'est pas garanti par l'invariant.

**Idée** : utiliser  $\text{Suf}(\text{Père}(\bar{h}_i))$  [avec  $\text{Suf}(\bar{\varepsilon}) = \bar{\varepsilon}$ ]



Si  $\overline{u_i v_i}$  est implicite dans  $T_i$ , il faut le rendre explicite (en coupant une arête) pour mettre à jour le lien suffixe de  $\bar{h}_i$  à  $\overline{u_i v_i}$ .

### Lemme

Si  $\overline{u_i v_i}$  était implicite dans  $T_i$ , alors  $\bar{h}_i = \overline{u_i v_i}$ .

$\Rightarrow$  tout nœud de  $\text{El}(T_{i+1})$  reste branchant.

## Algorithme de McCreight

La recherche de  $\bar{y}_i$  à partir de  $\bar{u}_i$  ne se fait pas en lisant  $v_i$  lettre à lettre, mais en progressant **arête par arête** en se basant, à chaque nœud explicite, sur la lettre suivante à lire.

Temps d'une étape de recherche de  $\bar{y}_i$  :  $O(n_i)$ ,  
 $n_i$  : nb. de nœuds explicites vus en lisant  $v_i$

Soit  $w_i$  le suffixe de  $t$  lu après  $u_i$ .

### Lemme

On a :  $n_i \leq |w_i| - |w_{i+1}|$ , d'où  $\sum_i n_i \leq n$ .

### Corollaire

L'algorithme de McCreight calcule l'arbre des suffixes en temps linéaire.

## Algorithme d'Ukkonen

Noeud implicite  $(\beta, u)$  normal  $\stackrel{def}{\iff} |u|$  minimal  
suf $[\gamma]$  calculé seulement pour  $\gamma$  explicite interne

```
procedure Inserer  $((\beta, u), t_{i+1})$   
   $\{(\beta, u)$  est le point actif normal $\}$   
   $\alpha := \text{root};$   
   $(\text{stop}, \delta) := \text{test\&split}((\beta, u), t_{i+1})$   
  tant que  $(!\text{stop})$   
    créer  $\delta_1$  et transition  $\delta \xrightarrow{t_{i+1}} \delta_1$   
    si  $\alpha \neq \text{root}$ ,  $\text{suf}[\alpha] := \delta$   
     $\alpha := \delta$   
     $(\beta, u) := \text{normaliser}(\text{suf}[\beta], u, t_{i+1})$   
     $(\text{stop}, \delta) := \text{test\&split}((\beta, u), t_{i+1})$   
  si  $\alpha \neq \text{root}$ ,  $\text{suf}[\alpha] := \delta$ 
```

Un **sous-mot** d'un mot  $u$  est un mot dont les lettres forment une suite extraite de la suite des lettres de  $u$ .

**Définition**  $u \sim_n v$  si et seulement si  $u$  et  $v$  ont même sous-mots de longueur  $\leq n$ .

**Lemme**  $\sim_n$  est d'indice fini.

**Proposition** Soit  $L \subseteq A^*$ . Alors

$L$  est union de  $\sim_n$ -classes

$\iff L$  est dans l'algèbre de Boole engendrée par les langages de la forme  $A^*a_1A^*a_2\cdots a_kA^*$ .

Dans ce cas,  $L$  est **testable par morceaux**.

## Théorème de Simon

$M(L)$ : monoïde syntaxique du langage  $L$ .

### Théorème

$L$  est testable par morceaux

$\iff M(L)$  satisfait  $(xy)^n x = (xy)^n = y(xy)^n$   
pour un  $n \leq (|M(L)|)!$ .

$\iff M(L)$  satisfait  $(xy)^\omega x = (xy)^\omega = y(xy)^\omega$

### Corollaire

On peut décider si  $L$  est testable par morceaux.

## Théorème de Simon

**Lemme 1**  $x, y \in A^*$ ,  $n > 0$

$$x \sim_n yx \iff x = x_1 \cdots x_n$$

$$\text{avec } c(y) \subseteq c(x_1) \subseteq \cdots \subseteq c(x_n).$$

Soit  $\longrightarrow$  définie par

$$x \longrightarrow y \iff (x = uav \text{ et } y = uv \text{ et } x \sim_n y)$$

**Lemme 2**  $x, y \in A^*$ ,  $n > 0$

Si  $x \sim_n y$ , il existe  $z$  tel que  $x \xleftarrow{*} z \xrightarrow{*} y$

**Lemme 3** Soit  $a \in A$ . Alors

$$xay \sim_{2n-1} xy \implies xa \sim_n x \text{ ou } ay \sim_n y$$

**Lemme 4**

Si  $M$  satisfait  $(xy)^\omega x = (xy)^\omega = y(xy)^\omega$ , la relation  $\leq$  définie par

$$x \leq y \iff MxM \subseteq MyM$$

est un ordre (partiel) sur  $M$ .