14/10/2014

Leçon 904 Langages rationnels - Exemples et Applications

(I) Formalisme syntaxique

1.1) Définition - Langage rationnel

définition 1 (Alphabet, mot, langage) [CAR], p 13

- Un alphabet Σ est un ensemble fini dont les éléments sont appelés lettres ou symboles
- Un mot est une suite finie d'éléments de Σ. Le mot vide est noté ε. Σ^* est l'ensemble des mots, appelé monoïde libre sur Σ
- Un langage L est une partie de Σ^*

exemple: Mots dont la longueur est un nombre premier.

Définition 2 (Operations rationnelles) [CAR], p 15

Les opérations rationnelles sur les langages sont :

- L'union de deux langages rationnels L et L' est noté $L+L'$.
- Le produit de deux langages rationnels L et L' est $LL' = \{uv / u \in L, v \in L'\}$
- Soit $L \subset A^*$. on définit l'étoile de L par: $L^0 = \{\varepsilon\}$, $L^{i+1} = LL^i$, $L^* = \bigcup_{i \geq 0} L^i$.

Définition 3 (Langages rationnels) [CAR], p 33

La classe $\mathcal{R}$ des langages rationnels sur Σ est la plus petite famille de langages tels que

(i) $\emptyset \in \mathcal{R}$ et $\{a\} \in \mathcal{R}$ pour tout $a \in \Sigma$

(ii) $\mathcal{R}$ est close par les opérations rationnelles

Exemples : $\{\varepsilon\} = \emptyset^*$, $\Sigma^* \in \mathcal{R}$

1.2) Expressions régulières

Définition 4 (Expression regulière) [WOL], p 10

La classe $\mathcal{E}$ des expressions rationnelles sur Σ est la plus petite famille d'expressions telles que

(i) $\emptyset \in \mathcal{E}$ et $a \in \mathcal{E}$ pour tout $a \in \Sigma$

(ii) Si $(E, E') \in \mathcal{E}^2$, alors $((E+E'), (EE'), E^*) \in \mathcal{E}^3$

Remarque : Par abus de notation, s'il n'y a pas ambiguïté, on omettra les parenthèses, et si $A = \{a_1, \dots a_n\}$, on notera A pour $a_1 + \dots + a_n$.

Exemples : $(a+b^*)$ noté $a+b^*$, $(ab)^*a$, A^*, a^*, $(a+aa)^*$ sont des expressions rationnelles

Propriété 5 Les expressions rationnelles sont non ambiguës. On peut donc définir des fonctions inductives sur les expressions rationnelles

Définition 6 Le langage $L(E)$ associé à une expression rationnelle est défini par :

(1) $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$ pour tout $a \in \Sigma$

(2) $L((E_1 + E_2)) = L(E_1) + L(E_2)$

(3) $L((E_1 E_2)) = L(E_1) L(E_2)$

(4) $L(E_1^*) = L(E_1)^*$

Propriété 7 L'application $L : \mathcal{E} \longrightarrow \mathcal{R}$, $E \longmapsto L(E)$ est surjective.

Contre exemple pour l'injectivité : $L((a+aa)^*) = L(a^*)$

1.3) Équation sur les langages [CAR], 40

On sera amenés à resoudre des equations linéaires sur les langages. Pour cela, on introduit des outils

Lemme 8 : Lemme d'Arden

Soient K, L deux langages et l'equation $X = KX + L$, X désignant un langage

1) Si $\varepsilon \notin K$, l'unique solution est $X = K^* L$

2) Si $\varepsilon \in K$, les solutions sont de la forme $X = K^*(L+Y)$, $Y \subseteq A^*$

Elimination de Gauss

Pour résoudre un système d'équations $\{X_p = \sum_{q \in Q} A X_q + Y, p \in Q\}$ on procède par élimination successive des langages X_p en utilisant le lemme d'Arden.

II) Automates finis

2.1) Définitions des automates

[CAR], p35-36

Définition 9 (Automate fini)
Un automate fini est un quintuplet $A=(Q,\Sigma,\delta,I,F)$ où Q est fini, $I\subset Q$, $F\subset Q$, $\delta\subset Q\times\Sigma\times Q$.
Les éléments de Q sont les états, ceux de I sont les état finaux, ceux de F les état finaux, ceux de δ les transitions.

Définition 10 (chemin, chemin acceptant, mot accepté, Langage accepté)
- Un chemin est une suite finie $q_0 \xrightarrow{a_1} q_1 \cdots \xrightarrow{a_n} q_n$ telle que $\forall i\in[\![1,n]\!]$, $(q_{i-1},a_i,q_i)\in\delta$. On note $q_0 \xrightarrow{a_1\cdots a_n} q_n$
- Un chemin est acceptant si $q_0\in I$, $q_n\in F$
- Un mot est accepté par A s'il est l'étiquette d'un chemin acceptant.
- Le langage accepté par A est l'ensemble des mots acceptés par A
- Un langage est reconnaissable s'il existe un état qui l'accepte.

Exemple : (cf Annexe) : recherche de mot dans un texte : abaa

Définition 11 (Automate émondé) Un automate est émondé si par tout état passe au moins un chemin acceptant.

Exemple : cf Annexe

Définition 12 (Automate normalisé) A est normalisé si $I=\{i\}$, $F=\{f\}$ et $\forall a\in\Sigma, q\in Q$, $(q,a,i)\notin\delta$ et $(f,a,q)\notin\delta$

Propriété 13 Pour tout automate A, il existe un automate normalisé A' acceptant le même langage. (A et A' sont équivalents)

Exemple : cf Annexe

Définition 14 (Automate déterministe) A est déterministe si $I=\{i\}$ et $(p,a,q)\in\delta$, $(p,a,q')\in\delta \Rightarrow q=q'$

Exemple : cf Annexe

2.2) Les langages reconnaissables sont rationnels

Définition 15 (Grammaires linéaires à gauche) [CAR], p75
- Une grammaire linéaire à gauche est un quadruplet (Σ,V,P,S) où Σ et V sont des alphabets finis et disjoints, appelés terminaux et variables, et P est une partie finie de $V\times(\Sigma V\cup\{\varepsilon\})$ appelé règles. $S\in V$ est le symbole de départ.
- (Dérivation) u se dérive en v si il existe $\alpha\in\Sigma^*$, $X\in V$ et $w\in\Sigma V\cup\{\varepsilon\}$ tels que
$$u=\alpha X,\quad v=\alpha w,\quad (X,w)\in P.$$
On note $u\to v$.
On note $u\to^* v$ s'il existe une suite finie de dérivations passant de u à v.
- (Langage engendré) $\hat{L}_{\mathcal{G}}=\{v\in\Sigma^*(V\cup\{\varepsilon\}),\ S\to^* v\}$
$$L_{\mathcal{G}}=\hat{L}_{\mathcal{G}}\cap\Sigma^*.$$

Définition 16 Soit $A=(Q,\Sigma,\delta,I,F)$ [SAK], p107
$p\in Q$. $L_p=\{f\in\Sigma^*\mid\exists t\in T,\ p\xrightarrow[A]{f}t\}$

- On a $L_p=\bigcup_{(x,q)\mid(p,x,q)\in\delta} xL_q+\delta_{p,F}$, $\delta_{p,F}=\varepsilon$ si $p\in F$, $\emptyset$ sinon
et $L(A)=\bigcup_{p\in Q} L_p$

Proposition 17 Les langages reconnaissables sont engendrés par une grammaire linéaire à gauche, et sont rationnels

2.3) Les langages rationnels sont reconnaissables

Lemme 18 Un langage L est reconnaissable si et seulement si $L\setminus\{\varepsilon\}$ est reconnaissable

- On peut construire par induction un automate normalisé reconnaissant $L\setminus\{\varepsilon\}$ défini par une expression rationnelle (cf Annexe)

Théorème de Kleene : Les langages rationnels et les langages reconnaissables sont les mêmes

III) Application du Théorème de Kleene

Le théorème de Kleene donne des conditions de rationalité des langages.

3.1) Lemme de l'étoile

[SAK], p78

Lemme 19 (Lemme de l'étoile)

Si L est rationnel, $\exists N \in \mathbb{N}$ tel que pour tout $f \in L$, si $f = g_1 h g_2$ et $|h| \geq N$, alors il existe une factorisation $h = uvw$, $v \neq \varepsilon$ et $g_1 u v^* w g_2 \subset L$.

Contre exemple : Ce lemme ne donne pas une CNS : [SAK], p80

$L = \{(aab)^n (abb)^n \mid n \in \mathbb{N}\} \cup A^* (aaa + bbb) A^*$

Lemme 20 (Lemme de l'étoile par blocs)

Si L est rationnel, $\exists N \in \mathbb{N}$ tel que pour tout $f \in L$, pour toute factorisation $f = u v_1 v_2 \dots v_N w$, $|v_i| \geq 1$, alors il existe $0 \leq j < k \leq N$ tel que

$u v_1 \dots v_j (v_{j+1} \dots v_k)^* v_{k+1} \dots v_N w \subset L$

Contre exemple Ce lemme ne donne pas une CNS

particulier $L = \{a^{k_1} b a^{k_2} b \dots a^{k_n} b, \exists i, i \neq k_i\}$

3.2) Quotients à gauche et minimisation

[CAR], 45, 46

Définition 21 (Quotient à gauche)

$L \subset A^*$, le quotient à gauche de L par $u \in A^*$ est $u^{-1}L = \{v \in A^*, uv \in L\}$

Proposition 22 Un langage est rationnel si et seulement si il a un nombre fini de quotients à gauche

Définition 23 (Automate minimal)

Soit L un langage rationnel. L'automate minimal de L est $\mathcal{A}_L = (Q, \Sigma, E, \{i\}, F)$, où
- $Q = \{u^{-1}L, u \in \Sigma^*\}$
- $i = L$
- $E = \{u^{-1}L \xrightarrow{a} (ua)^{-1}L, u \in \Sigma^*, a \in \Sigma\}$
- $F = \{u^{-1}L, u \in L\}$

Définition 24 Congruence de Nerode

$\mathcal{A} = (Q, \Sigma, \delta, \{i\}, F)$ un automate déterministe complet.

Alors $q \sim q' \Leftrightarrow (\forall w \in \Sigma^*, (q.w \in F \Leftrightarrow q'.w \in F))$

Propriété 25 Si $\mathcal{A}$ reconnaît L, l'automate minimal de L est égal à $\mathcal{A}/_\sim = (Q/\sim, \Sigma, \delta', \{[i]\}, \{[f], f \in F\})$

$\delta' = \{[q] \xrightarrow{a} [q.a], q \in Q, a \in \Sigma\}$.

IV) Applications des langages rationnels

4.1) Recherche de mot clé un texte

[BBC]

Algorithme de Knuth-Morris-Pratt (DVP)

4.2) Problème de séparation par Automate

[FB]

Le problème de séparation par automates est NP-complet (DVP)

Références

[CAR] Olivier Carton, Langages formels.

[WOL] Pierre Wolper, Introduction à la Calculabilité

[SAK] Jacques Sakarovitch, Éléments de théorie des automates

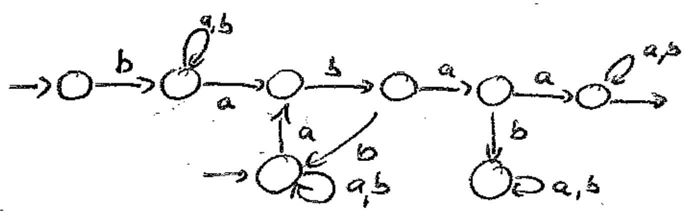
[BBC] Beauquier, Berstel, Chrétienne, Éléments d'Algorithmique..

[FB] Floyd, Begel, The language of machines.

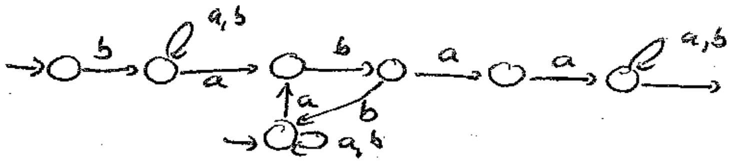
[HU] Hopcroft Ullmann, Introduction to automata theory.

Annexe

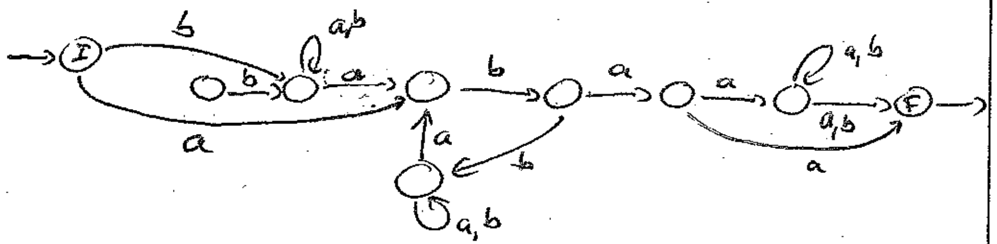
recherche de mot : abaa



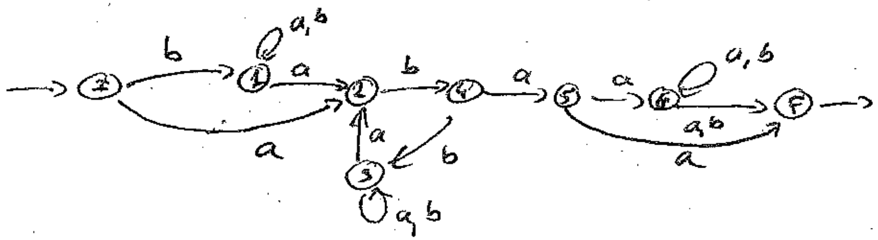
automate émondé



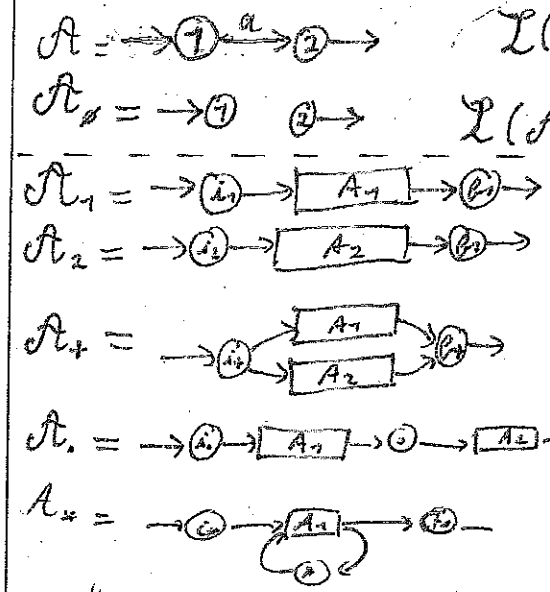
automate normalisé



émondé



théorème de Kleene



$\mathcal{L}(\mathcal{A}) = \{a\}$

$\mathcal{L}(\mathcal{A}_\emptyset) = \emptyset$

$L_1 = \mathcal{L}(\mathcal{A}_1)$, $\varepsilon \notin L_1$

$L_2 = \mathcal{L}(\mathcal{A}_2)$, $\varepsilon \notin L_2$

$\mathcal{L}(\mathcal{A}_+) = L_1 + L_2$

$\mathcal{L}(\mathcal{A}_\cdot) = L_1 L_2$

$\mathcal{L}(\mathcal{A}_*) = L_1^* \setminus \{\varepsilon\}$

$\Downarrow$

Algorithme de Thompson.

Autre possibilité : (Glushkov). (très coûteux.

$$\begin{array}{ccccccc} (a & + & b &)^* & + & c & \cdot d \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ 1 & & 2 & 3 & 4 & 5 & 6 \end{array}$$

	a	
1	{1,3}	

Le problème de séparation par automates est NP-complet

Énoncé du problème :

> Soit $(S,T) \in \Sigma^* \times \Sigma^*$ deux ensembles finis et disjoints et soit $k \in \mathbb{N}$.
> On cherche un automate déterministe à k états, $\mathcal{A} = (Q, \Sigma, \delta, \{i\}, F)$ tel que $S \subset \mathcal{L}(\mathcal{A})$ et $T \cap \mathcal{L}(\mathcal{A}) = \emptyset$.

Théorème : Ce problème est NP-complet

Preuve : • Ce problème est NP : étant donné un automate $\mathcal{A}$ ~~déterministe~~ construit de manière non déterministe, on vérifie en temps linéaire que $S \subset \mathcal{L}(\mathcal{A})$ et $T \cap \mathcal{L}(\mathcal{A}) = \emptyset$

• Ce problème est NP-dur : On va réduire polynomialement le problème SAT au problème de séparation par automates ; on construit les ensembles S et T.

Soit $F = \bigwedge_{i=0}^{m-1} C_i$ une CNF sur les littéraux $x_0, \dots, x_{v-1}$.

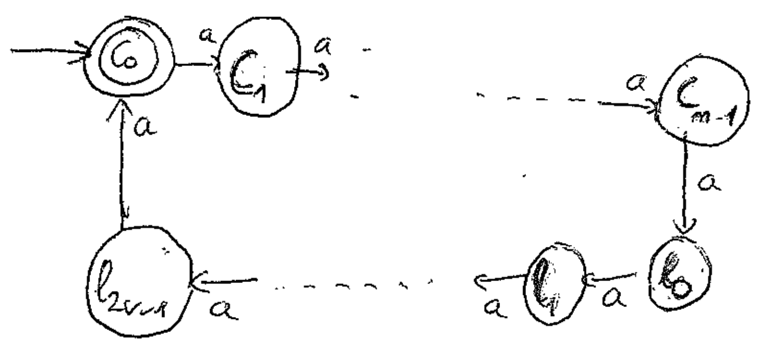
on choisit $\boxed{k = m + 2v}$ et l'alphabet $\underline{\Sigma = \{a, b\}}$

On note $l_0, \dots l_{2v-1} = x_0, \dots x_{v-1}, \overline{x_0}, \dots \overline{x_{v-1}}$.

Soit $\boxed{S_0 = \{\varepsilon, a^k\}}$, $\boxed{T_0 = \{a^i, 0 < i < k\}}$

Si $\mathcal{A}$ accepte S_0 et refuse T_0, et $\mathcal{A}$ a k états, alors $\mathcal{A}$ est une boucle de transitions étiquetées par a, avec $F = \{i\}$.

On note $c_0 = i$ et $Q = \{c_0, \dots c_{m-1}, l_0, \dots l_{2v-1}\}$ dans l'ordre des transitions



va utiliser le codage suivant par des transitions étiquettées par b :

- $C_i \xrightarrow{b} l_j \Rightarrow l_j$ est un litteral de C_i
- $l_j \xrightarrow{b} C_0 \Rightarrow l_j$ est un litteral vrai.

On veut même : $l_j \xrightarrow{b} q$, $q \in Q \Rightarrow q = C_0$ et l_j est vrai.

$$\boxed{T_1 = \{a^i b a^r,\ i<m,\ r<k\} \setminus \{a^i b a^{2v-j},\ l_j \text{ litteral de } C_i\}}$$

$$\boxed{T_2 = \{a^{m+j} b a^i,\ j<2v,\ 0<i<k\}}$$

- Il faut éviter le cas où x_i et $\overline{x_i}$ sont codés vrais

$$\boxed{T_3 = \{a^{m+j} b a^{m+j+v} b,\ j<v\}}$$

Finalement, pour résoudre le problème SAT, il faut que chaque clause C_i ait un litteral vrai :

$$\boxed{S_1 = \{a^i bb,\ i<m\}}$$

$\boxed{S = S_0 \cup S_1}$ et $\boxed{T = T_0 \cup T_1 \cup T_2 \cup T_3}$. sont des ensembles finis

Un algorithme qui résout le problème de séparation de S et T avec k états donne une solution de SAT. Il ne peut donc être polynomial

$\square$

Recherche de motifs : algorithme de Knuth, Morris et Pratt

Recherche de motifs :

C'est un problème utile notamment en traitement de texte et en biologie (recherche d'une séquence d'acides aminés sur un brin d'ADN). On veut déterminer efficacement, étant donné un mot $x \in \Sigma^*$, l'appartenance à $\Sigma^* x \Sigma^*$ d'un mot t, et trouver, le cas échéant, la première occurrence de x dans t

1) Un algorithme naïf :

Soit $m = |x|$ et $n = |t|$. On pourrait penser à comparer à x tous les sous-mots de t de longueur m successivement, et s'arrêter en cas d'égalité.

On obtiendrait :

```
RECHERCHE(x,t) :
m := |x| ; l := 0 ; trouvé := faux, parcouru := faux ; n := |t|
tant_que (non trouvé) et (non parcouru)
      si l + m > n alors parcouru := vrai fin
      si  x = t_{l+1} ... t_{l+m} alors trouvé := vrai
      sinon l := l + 1 fin
fin
si trouvé retourne l
sinon retourne 0
fin
```

On effectue dans le pire cas $m(n-m+1)$ comparaisons (c'est le cas pour $x = a^{m-1}b$; $t = a^{n-1}b$)

Certaines sont redondantes : ~~par~~ exemple, si $x = abb$ et que le test échoue au rang l après avoir testé avec succès $t_{l+1} = x_1$ et $t_{l+2} = x_2$, il est inutile de relancer un test au rang $l+1$, puisque $x_2 \neq x_1$, i.e. $t_{l+2} \neq x_1$

...gorithme de Morris et Pratt :

Plus précisément, si l'échec se produit lors de la comparaison de x_i à t_{k+i}, on a $x_1 \dots x_{i-1} = t_{k+1} \dots t_{k+i-1}$ et $x_i \neq t_{k+i}$, donc $t_{k+1} \dots t_{k+i-1}$ est préfixe de x.

Si une recherche est lancée à un rang $l \in [\![k+1, k+i-1]\!]$, alors $t_{l+1} t_{l+2} \dots t_{k+i-1}$ est un suffixe de $t_{k+1} \dots t_{k+i-1} = x_1 \dots x_{i-1}$. On compare donc x à un sous-mot de x, comparaison que l'on peut effectuer indépendamment de t et garder en mémoire.

Pour formaliser cette idée, on introduit la notion de bord : un bord de $u \in \Sigma^*$ est un sous-mot strict de u qui en est à la fois préfixe et suffixe.

On note $\mathrm{Bord}(x)$ le bord maximal de x.

On définit la fonction $\beta : [\![0, m]\!] \to [\![-1 ; m-1]\!]$ (où $m = |x|$) par $\beta(0) = -1$ et $\forall i \in [\![1, m]\!]$ $\beta(i) = |\mathrm{Bord}(x_1 \dots x_i)|$

On a toujours $\beta(i) < i$

Revenons à notre recherche de motif : si $x_1 \dots x_{i-1} = t_{k+1} \dots t_{k+i-1}$ et $x_i \neq t_{k+i}$, et si $p \in [\![k+1, k+i-1]\!]$, alors $t_{k+p+1} \dots t_{k+i-1}$ ne peut être un préfixe de x (et donc une recherche en p ne peut avoir une chance de succès) que s'il est un bord de $x_1 \dots x_{i-1}$

On va donc vouloir relancer une comparaison à partir du rang l tel que $t_{l+1} \dots t_{k+i-1}$ est le plus grand bord de $x_1 \dots x_{i-1}$, i.e. $l = k + m - \beta(i-1)$

Concrètement, lors du calcul, en cas d'échec au rang k et à la lettre i ($x_1 \dots x_{i-1} = t_{k+1} \dots t_{k+i-1}$ et $x_i \neq t_{k+i}$), il suffit donc de "continuer" la comparaison en remplaçant i par $s(i) = 1 + \beta(i-1)$

On appelle s fonction de suppléance de x

Dans l'algorithme de Morris et Pratt, i et j représentent les lettres courantes dans x et t respectivement. On a donc pour invariant de boucle $x_1 \dots x_{i-1} = t_{j-i+1} \dots t_{j-1}$

```
RECHERCHE_MP(x, t):
i:=1; j:=1
tant-que (i≤m) et (j≤n) fais
    si (i ≥ 1) et (t.[j] ≠ x.[i]) alors i:=s(i)
    sinon i:=i+1; j:=j+1 fin
fin
si i > m alors retourne j-m sinon retourne 0 fin
```

L'algorithme effectue au plus $2n-1$ comparaisons si on a déjà calculé la fonction s. En effet, une comparaison réussie incrémente strictement j, et un échec incrémente strictement $j-i$.

Le calcul de s est polynomial en m (en général $m \ll n$)

Algorithme de Knuth, Morris et Pratt

Il existe encore un type de comparaisons redondantes qu'on peut facilement éliminer ; si $x_i = x_{1+\beta(i-1)}$, alors il est inutile, dans le cas où $x_1 \ldots x_{i-1} = t_{\ell+1} \ldots t_{\ell+i-1}$ et $t_{\ell+i} \neq x_i$, de relancer un test au rang $p = i-1-\beta(i-1)$, comme le fait l'algorithme de Morris et Pratt : ce test va systématiquement échouer puisque $t_{\ell+i} = t_{p+1-\beta(i-1)}$ est distinct de x_i, donc de $x_{1+\beta(i-1)}$

On adapte donc notre fonction de suppléance, pour interdire le cas $x_{i-p} = x_i$

Plus formellement, on dira que deux préfixes u et v de x sont disjoints si l'un est x ou si $x_{1+|u|} \neq x_{1+|v|}$
soit v un préfixe de x, u est appelé bord disjoint de v si u est un bord de v et u et v sont des préfixes disjoints
On note DBord(v) le plus long bord disjoint de v, s'il existe.
Les bords "intéressants" au sens de la recherche de motif sont les bords disjoints : les autres ne mènent nulle part à cause du problème énoncé en début de partie
On définit donc $\gamma : [\![0, m]\!] \to [\![-1, m-1]\!]$

$$0 \mapsto 1$$
$$j \mapsto \begin{cases} |\mathrm{DBord}(x_1 \ldots x_j)| & \text{si } x_1 \ldots x_j \text{ a un bord disjoint dans } x \\ -1 & \text{sinon} \end{cases}$$

En découle une nouvelle fonction de suppléance :
$r(i) = 1+\gamma(i-1)$

L'algorithme est alors le même en remplaçant s par r

```
KMP (x, t)
  i := 1, j := 1,
  tant que i ≤ m et j ≤ n faire
    tant que i > 0 et
      t[j] ≠ x[i] faire :
        i := r[i]
    fin
    i := i + 1
    j := j + 1
  fin
  si i > m alors
    texte j à la position j - m.
```

délai

$\mathcal{A}(x)$: automate des occurrences. [BE]

Fonctions de suppléance

On utilise le lien entre β et γ :

$\beta(1+j) = 1+\gamma^{l}(\beta(j))$ où l est le plus petit entier l tel que :
$(1+\gamma^{l}(\beta(j)) = 0)$ ou $(1+\gamma^{l}(\beta(j)) \neq 0$ et $x_{1+\gamma^{l}(\beta(j))} = x_{1+j})$

$$\gamma(j) = \begin{cases} \beta(j) \text{ si } j = m \text{ ou } x_{1+j} \neq x_{1+\beta(j)} \\ \gamma(\beta(j)) \text{ sinon} \end{cases}$$

On en déduit un algorithme calculant r :

```
SUPPLEANCE (x) :
r[1] := 0 , i := 0
pour j = 1 à m-1 faire
    tant que i > 0 et x[j] ≠ x[i] faire i := r[i] fin
    i := i+1
    si x[1+j] ≠ x[i] alors r[1+j] := i, sinon r[1+j] := r[i]
fin
```

C'est la traduction de la deuxième propriété ci-dessus : la boucle tant que calcule en fait $s[j] = 1+\beta(j-1)$

Le calcul est linéaire en $m = |x|$.

En termes d'automates, cette procédure revient à implémenter l'automate minimal de $\Sigma^* x$, i.e. un automate à $m+1$ états (les préfixes de x) dont les transitions sont : $i \xrightarrow{a} j$ lorsque $x_1 \ldots x_j$ est le plus long préfixe de $x_1 \ldots x_i a$ qui est suffixe de $x_1 \ldots x_j$
(on confond un préfixe et sa longueur en étiquetant les états)

pour $x = ababc$, $\mathcal{A} =$