Automate des occurrences

Julie Parreaux

2018-2019

Référence du développement : Cormen [2, p.915]

Leçons où on présente le développement : 907 (Texte) ; 909 (Langages rationnels) ; 921 (Recherche).

1 Introduction

L'automate des occurrences, aussi appelé l'automate des suffixes permet d'accepter les mots dont le suffixe est un certain motif : il reconnaît des mots du type $\Sigma^* P$ où P est fixé à l'avance. De plus, sa construction nous donne un automate minimal ce qui est toujours intéressant.

Cet automate peut être utilisé pour la recherche d'un motif dans un texte puisque rechercher un motif c'est vérifier si un des préfixes de ce texte n'a pas comme suffixe le motif. Autrement dit, on cherche si un préfixe du texte n'est pas accepté par l'automate des occurrences. Cette remarque, sans le prétraitement, nous donne une recherche linéaire en la taille du texte puisqu'une seule lecture du texte dans l'automate nous donne la présence ou non du motif dans le texte.

Remarques sur le développement

Ce développement peut se spécialiser en fonction de la leçon.

1. Définition et propriétés de la fonction suffixe.
2. Définition de l'automate des occurrences (leçons 907 et 921 : présentation d'un exemple).
3. Correction de cet automate.
4. Minimalité de cet automate (leçon 909).
5. Application à la recherche de motif (leçons 907 et 921).

2 Propriétés de l'automate des occurrences et applications

Soit Σ un alphabet fini, $T \in \Sigma^*$ un texte et $P \in \Sigma^*$ un mot que l'on cherche dans T. On suppose P fixé.

Définition et propriétés de la fonction suffixe Nous commençons par définir la fonction suffixe qui est une manière de calculer le décalage en cas d'échec de la lecture.

Définition. On définie la fonction suffixe associée à P, σ, telle que $\forall u \in \Sigma^*$

$$\sigma(u) = \max_{k \in [\![0,p]\!]} \{P_k \text{ est suffixe de } u\}$$

où p est la taille de P et P_k est le préfixe de P de longueur k. (Cette fonction calcule la taille du plus long suffixe de u qui est aussi préfixe de P.)

Proposition. *La fonction suffixe vérifie les propriétés suivantes.*

1. *Si u est un suffixe de v alors $\sigma(u) \leq \sigma(v)$ (remarque après la définition)*

2. *Si $u \in \Sigma^*$ et $a \in \Sigma$, alors $\sigma(ua) \leq \sigma(u) + 1$ (lemme 32.2)*
3. *Si $w \in \Sigma^*$ et $a \in \Sigma$, alors $\sigma(wa) = \sigma(P_{\sigma(w)}a)$ (lemme 32.3)*

Démonstration. 1. Si u est suffixe de v alors tout suffixe w de u est aussi suffixe de v. En particulier si w est le plus long suffixe de u qui est aussi préfixe de P, alors w est suffixe de v. Par définition de la fonction suffixe, on $\sigma(u) \leq \sigma(v)$ (w n'est pas nécessairement le plus long pour v).

2. On pose $r = \sigma(ua)$. Si $r = 0$, alors $r = 0 \leq \sigma(u) + 1$ par positivité de la fonction σ. Sinon, $r > 0$ et P_r est un suffixe de ua. En particulier P_{r-1} est un suffixe de u (on supprime le a à la fin de P_r et de ua). D'où $r - 1 \leq \sigma(u)$ (par définition de $\sigma(u)$ qui est la longueur maximale d'un suffixe de u qui est préfixe de P). Donc, $r \leq \sigma(u) + 1$.

3. $\geq$ Par définition de σ, $P_{\sigma(w)}$ est un suffixe de w. En particulier $P_{\sigma(w)}a$ est un suffixe de wa. Donc, (par le même raisonnement que précédemment) $\sigma(wa) \geq \sigma(P_{\sigma(w)}a)$.

   $\leq$ Réciproquement, $P_{\sigma(w)}a$ et $P_{\sigma(wa)}$ sont suffixes de wa. Comme, $|P_{\sigma(wa)}| = \sigma(wa) \leq \sigma(w) + 1 = |P_{\sigma(w)}|$ (item précédent), $P_{\sigma(wa)}$ est un suffixe de $P_{\sigma(w)}a$. D'où, $\sigma(wa) \leq \sigma(P_{\sigma(w)}a)$.

□

Définition de l'automate des occurrences On continue en énonçant la définition de l'automate des occurrences ainsi que ces propriétés.

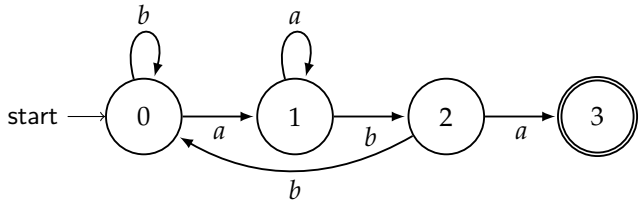

FIGURE 1 – Un automate des occurrences pour le motif *aba*. (On ne fait un automate qui ne reconnaît que $\Sigma^* w$, cela simplifie la preuve de la minimalité (sinon on contredit la co-déterminisation).)

Dans le cadre de la recherche, on l'applique à $T = aabbabab$.

Définition. On construit l'automate des occurrences comme suit :

— $Q = [\![0, m]\!]$ si $m = |P|$
— $I = 0$
— $F = \{m\}$
— $\delta(q, a) = \sigma(P_q a)$

Définition. On définit $\phi : \Sigma^* \to Q$ la fonction calculant l'état dans lequel on arrive dans l'automate en lisant un mot donné (en entrée). On la définie par induction :

— $\phi(\epsilon) = 0$;
— $\forall u \in \Sigma^*\ a \in \Sigma, \phi(ua) = \delta(\phi(u), a)$.

Théorème. *Pour tout mot $w \in \Sigma^*$, on a $\phi(w) = \sigma(w)$.*

Démonstration. On raisonne par récurrence sur la taille de w. On note, pour tout $i \in \mathbb{N}$, $\mathcal{P}_i$: "$\forall w \in \Sigma^i, \phi(w) = \sigma(w)$" l'hypothèse de récurrence pour tous les mots de longueur i.

Initialisation Pour $i = 0$, $|w| = 0$ implique que $w = \epsilon$. On a par définition, $\phi(\epsilon) = 0$ et $\sigma(\epsilon) = 0$.

Hérédité Soit $i \in \mathbb{N}$ tel que $\mathcal{P}_i$ soit vraie. Soit $w \in \Sigma^{i+1}$. Montrons que $\phi(w) = \sigma(w)$. On note $w = w'a$ avec $|w'| = 1$. On a :

$$\begin{array}{rcll} \phi(w) & = & \phi(w'a) & \text{(définition de } w\text{)} \\ & = & \delta(\phi(w'), a) & \text{(définition de } \phi\text{)} \\ & = & \delta(\sigma(w'), a) & \text{(hypothèse de récurrence)} \\ & = & \sigma(P_{\sigma(w')}a) & \text{(définition de } \delta\text{)} \\ & = & \sigma(w'a) & \text{(item 3 de la proposition 2)} \\ & = & \sigma(w) & \text{(définition de } w\text{)} \end{array}$$

□

Corollaire. *Soit P un motif de taille m et $\mathcal{A}$ son automate des occurrences. Alors, on a $\mathcal{L}(\mathcal{A}) = \Sigma^* P$.*

Démonstration. On raisonne par double inclusion.

$\subseteq$ Si $w \in \mathcal{L}(\mathcal{A})$, alors $\delta(0, w) = m = \sigma(m)$. Donc P est suffixe de w et $w \in \Sigma^* P$.

$\supseteq$ Si $w \in \Sigma^* P$, alors $\sigma(w) = m = \delta(0, w)$. Donc w est accepté par l'automate $\mathcal{A}$, soit $w \in \mathcal{L}(\mathcal{A})$.

□

Théorème. *Soit $\mathcal{A}$ l'automate des occurrences d'un motif P. $\mathcal{A}$ est l'automate minimal pour $\Sigma^* P$.*

Démonstration. Montrons que l'automate des occurrences est minimal par l'algorithme des renversements. Pour cela, on va montrer qu'il est déterministe, co-accessible et co-déterministe.

Montrons que l'automate des occurrences $\mathcal{A}$ est déterministe. Comme δ est définie par la fonction suffixe σ, $\mathcal{A}$ est déterministe (sa fonction transition est donnée par une fonction).

Montrons que l'automate des occurrences $\mathcal{A}$ est co-accessible ($\forall q \in Q, \exists f \in F$ tel que $\exists w \in \Sigma^+, \delta(q, w) = f$). Soit $q \in Q$. Comme $F = \{m\}$, f est nécessairement m. On pose w qui est le suffixe de taille $m - q$ de P. Par définition de δ, on a $\delta(q, w) = \sigma(P_q w) = \sigma(P) = m$. Donc $\mathcal{A}$ est co-accessible.

Montrons que l'automate des occurrences $\mathcal{A}$ est co-déterministe (le miroir de celui-ci est déterministe). Par définition $F = \{m\}$, donc $\mathcal{A}$ ne possède qu'un état acceptant ? Montrons que pour tout état $i \in Q$ et toute lettre $a \in \Sigma$, il n'y a au plus qu'une transition de a qui arrive dans i. Raisonnons par l'absurde et supposons qu'il existe $i \in Q$ et $a \in \Sigma$ tels que $\delta(q, a) = \delta(p, a) = i$ avec $p \neq q$. Par définition de δ, $\sigma(P_q a) = \sigma(P_p a) = i$. Par la construction de $\mathcal{A}$, il existe $v \in \Sigma^*$ tel que $P_q av = P = P_q av$ D'ou $|P_q av| = |P_p av|$. On en déduit que $|P_q| = |P_q|$. On obtient une contradiction car $p \neq q$.

Par la propriété justifiant la correction de l'algorithme par renversement, $\mathcal{A}$ est minimal. □

Autre preuve à privilégier. Soient $i \neq j$ deux états distincts q_i et q_j de l'automate des occurrences. Montrons que ces deux états sont séparables par la congruence de Nérode. Soit w un mot tel $q_i \overrightarrow{w} q_f$. Alors $q_j \overrightarrow{w} q_k$ avec $k \neq f$ car sinon w est suffixe des préfixes de taille i et de taille j. □

Remarque. Un argument de cardinalité sur les états peut être également évoquer.

Application à la recherche de motif [2, p.916] Une application de cet automate est son utilisation pour la recherche d'un motif. En effet, si P est dans un texte T alors $T = \Sigma^* P \Sigma^*$. L'automate que nous venons de construire est donc tout adapté à cette recherche. Les algorithmes 3 et 4 permettent d'effectuer cette recherche. Ils sont correct par le théorème précédent (pas celui sur la minimalité). On obtient une recherche en $O(|T|)$ avec un prétraitement de $O(m|\Sigma|)$. L'algorithme KMP améliore la complexité du prétraitement.

3 Algorithme de Knuth–Morris–Pratt

Une amélioration de l'application à la recherche de motif est l'algorithme de Knuth–Morris–Pratt qui va se passer de l'automate pour calculer le saut à effectuer. Les algorithmes Morris–Pratt et Knuth–Morris–Pratt sont des algorithmes qui utilisent la notion de bord. On utilise le bord des préfixes du motif pour être plus astucieux et rapide quand une comparaison échoue : il nous donne le décalage que l'on doit réaliser.

Hypothèse : Le motif est fixe et connu à l'avance.

Quelques notions sur le bord [1, p.340] Le bord a un rôle essentiel dans ces algorithmes : il permet de calculer le décalage que l'on va effectuer lors d'un échec de comparaison. Bien définir cette notion est donc primordiale.

Définition. Un bord de x est un mot distinct de x qui est à la fois préfixe et suffixe de x. On note $Bord(x)$ le bord maximal d'un mot non vide.

Exemple Le mot *ababa* possède les trois bords ϵ, a et aba. De plus $Bord(ababa) = aba$.

Proposition. *Soit x un mot non vide et $k \in \mathbb{N}$, le plus petit, tel que $Bord^k(x) = \epsilon$.*

1. *Les bords de x sont les mots $Bord^i(x)$ pour tout $i \in \{1, \ldots, k\}$.*

2. *Soit a une lettre. Alors, $Bord(xa)$ est le plus long préfixe de x qui est dans l'ensemble $\{Bord(x)a, \dots Bord(x)^k a, \epsilon\}$.*

Arguments de la preuve. 1. z est un bord de $Bord(x)$ ssi z est un bord de x (+ récurrence).

2. z est un bord de za ssi $z = \epsilon$ ou $z = z'a$ avec z' un bord de x.

□

Corollaire. *Soit x un mot non vide et soit a une lettre. Alors,*

$$Bord(xa) = \begin{cases} Bord(x)a & \text{si } Bord(x)a \text{ est préfixe de } x \\ Bord(Bord(x)a) & \text{sinon} \end{cases}$$

Arguments de la preuve. — Si $Bord(x)a$ est préfixe de x alors $Bord(x)a = Bord(xa)$.

— Sinon, $Bord(xa)$ est préfixe de $Bord(x) = y$ et le plus long préfixe de x dans $\{Bord(y)a, \dots Bord(y)^k a, \epsilon\}$.

□

Définition. Soit x un mot de longueur m. On pose $\beta : \{0, \dots, m\} \to \{-1, \dots, m-1\}$ la fonction qui est définie comme suit : $\beta(0) = -1$ et pour tout $i > 0$, $\beta(i) = |Bord(x_1, \dots, x_i)|$. On pose $s : \{1, \dots, m-1\} \to \{0, \dots, m\}$ la fonction suppléance de x définie comme suit : $s(i) = 1 + \beta(i-1), \forall i \in \{1, \dots, m\}$.

Remarque : on a bien entendu $\beta(i) \leq i - 1$.

Corollaire. *oit x un mot non vide de longueur m. Pour $j \in \{0, \dots, m-1\}$, on a $\beta(1+j) = 1 + \beta^k(j)$ où $k \geq 1$ est le plus petit entier vérifiant l'une des deux conditions suivantes :*

1. $1 + \beta^k(j) = 0$
2. $1 + \beta^k(j) \neq 0$ *et* $x_{1+\beta^k(j)} = x_{j+1}$

Arguments de la preuve. Algorithme 1 □

Algorithm 1 Calcul de la fonction β donnant la taille des bords maximaux d'un mot x.

```
1:  function BORD-MAXIMAUX(x, β)        ▷ x mot de taille m
2:      β[0] ← −1
3:      for j = 1 à m do
4:          i ← β[j − 1]
5:          while i ≥ 0 et x[j] ≠ x[i + 1] do
6:              i ← β[i]
7:          end while
8:          β[j] ← i + 1
9:      end for
10: end function
```

Algorithm 2 Calcul de la fonction suppléance s du mot x.

```
1:  function SUPPLÉANCE(x, s)           ▷ x mot de taille m
2:      s[1] ← 0
3:      for j = 1 à m − 1 do
4:          i ← s[j]
5:          while i ≥ 0 et x[j] ≠ x[i] do
6:              i ← s[i]
7:          end while
8:          s[j + 1] ← i + 1
9:      end for
10: end function
```

L'algorithme de Morris–Pratt [2, p.916] L'algorithme de Morris–Pratt utilise un automate des occurrences pour calculer les bords du motif. On définit l'automate des occurrences associé à $P[1 \dots m]$ comme suit : $\mathcal{Q} = \{1, \dots m\}$; $q_0 = 0$; $F = \{m\}$; $\delta(q, a) = \sigma(P_q a)$ pour tout $a \in \Sigma$ et $q \in \mathcal{Q}$.

Définition. La fonction suffixe associé au motif P $\sigma : \Sigma^* \mapsto \{0, 1, \dots, m\}$ donne la longueur de $Bord(x)$ pour $x \in \Sigma^*$.

Algorithm 3 Calcul de la fonction de transition δ de l'automate des occurrences.

```
1: function CALCUL-δ(P, Σ)        ▷ P motif ; Σ alphabet
2:     p ← P.longueur
3:     for q = 0 à p do
4:         for a ∈ Σ do
5:             k ← min(p + 1, q + 2)
6:             repeat
7:                 k ← k + 1
8:             until P_k préfixe de P_q a   ▷ P_q est le préfixe de P de longueur q.
9:             δ(q, a) ← k
10:        end for
11:    end for
12:    Retourner δ
13: end function
```

Complexité temporelle du prétraitement dans le pire cas : $O(p|\Sigma|)$ Ce fait une fois par motif.

Algorithm 4 Calcul de la recherche dans l'automate des occurrences.

```
1: function MORRIS–PRATT(T, δ, p)   ▷ T texte ; δ fonction transition ; p la longueur du motif
2:     t ← T.longueur
3:     for i = 1 à t do
4:         q ← δ(q, T[i])
5:         if q = p then
6:             Retourner i − m
7:         end if
8:     end for
9:     Retourner 0
10: end function
```

Complexité temporelle dans le pire cas : $\Theta(t)$

Remarque : La validité de cet algorithme provient de la validité de la construction de l'automate des occurrences.

L'algorithme de Knuth–Morris–Pratt [1, p.343] Cet algorithme utilise une méthode plus astucieuse afin de calculer les bords du préfixes. On s'épargne ainsi un calcul de l'automate des occurrences et on calcul la fonction δ puisse à la volée. On exploite les propriétés de la fonction de bord.

Définition. La fonction préfixe associé au motif P $\pi : \{1, 2, \ldots, p\} \mapsto \{0, 1, \ldots, p-1\}$ donne la longueur du plus long préfixe de P qui est suffixe propre de P_q où $q \in \{1, 2, \ldots, p\}$.

Si on ne fait pas l'hypothèse d'un préfixe propre alors le plus long suffixe d'un mot qui est aussi son préfixe est le mot en question. De plus, sans cette hypothèse l'algorithme de Knuth-Morris-Pratt serait soit incorrect soit non terminal.

Algorithm 5 Algorithme de Knuth–Morris–Pratt.

```
1: function KMP(T, P)        ▷ T texte ; P motif
2:     t ← T.longueur
3:     p ← P.longueur
4:     i ← 1
5:     j ← 1
6:     s ← SUPPLÉANCE(x, P)
7:     while i ≤ p et j ≤ t do
8:         if i ≥ 1 et t[j] ≠ x[i] then
9:             i ← s[i]
10:        else
11:            i ← i + 1
12:            j ← j + 1
13:        end if
14:    end while
15:    if then i > m
16:        Renvoie j − m
17:    else
18:        Renvoie 0
19:    end if
20: end function
```

Remarque : La validité de l'algorithme de Knuth–Morris–Pratt (Algorithme 5) provient des propriétés sur le bord.
Complexité du prétraitement Calcul de la fonction suppléance s : $\Theta(p)$ On applique une méthode de l'agrégat : dans la boucle pour la variable ne peut augmenter plus de p fois et comme elle est toujours strictement positive, la boucle tant que ne peut pas s'exécuter plus de p fois.
Complexité : $\Theta(t)$ On applique une méthode de l'agrégat (exactement le même raisonnement).
Remarque : Il existe encore une version améliorer de cet algorithme qui consiste à ne pas vouloir se retrouver dans la même situation qu'au début (on teste toujours une situation différente). Cette amélioration donne les même complexité asymptotique mais semblerait plus rapide en pratique.

4 Automate minimal

Dans ce développement, on présente une preuve de minimalité d'un automate fini car l'automate des occurrences est un automate fini. On va donc présenter ici quelques caractéristiques de ces automates minimaux. Déterminer un automate déterministe contenant un nombre minimal d'états pour un langage rationnel donné est très intéressant en pratique. On définit ainsi un représentant canonique pour reconnaître un langage. Le résultat nous assurant de leur existence est un résultat très fort (qui reste vrai même si l'automate n'est pas déterministe). Il y a deux manières de définir un automate minimal (selon le point de vu que l'on considère suite au théorème de Kleene) : intrinsèquement au langage avec la notion de quotient ou en utilisant les états inséparables qui est une méthode plus calculatoire.

Morphismes d'automates La relation d'ordre permettant de définir un automate minimal est formellement donnée par la notion de morphisme d'automate.

Définition (Morphisme d'automate [3, p.120])**.** Soient $\mathcal{A}_1 = (Q_1, \delta_1, I_1, F_1)$ et $\mathcal{A}_2 = (Q_2, \delta_2, I_2, F_2)$ deux automates finis. Un morphisme d'automate $\phi : \mathcal{A}_1 \to \mathcal{A}_2$ est une application de Q_1 dans Q_2 telle que : $\phi(I_1) \subset I_2$; $\phi(F_1) \subset F_2$ et $\forall p, q \in Q_1, q \in \delta_1(p, a) \Rightarrow \phi(q) \in \delta_2(\phi(p), a)$.

Remarque. S'il existe $\phi : \mathcal{A}_1 \to \mathcal{A}_2$ un morphisme entre deux automates alors $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$. En effet, soit $w \in \mathcal{L}(\mathcal{A}_1)$. Par définition, il existe $i \in I_1$ tel que $\delta_1(i, w) \in F_1$. En appliquant le morphisme, on a $\phi(i) \in I_2$ (première condition sur le morphisme) et $\delta_2(\phi(i), w) \in F_2$ (deux dernières conditions sur le morphisme). Donc $w \in mathcal L(\mathcal{A}_2)$.

Proposition. *Si $\phi : \mathcal{A}_1 \to \mathcal{A}_2$ est un morphisme d'automate surjectif entre deux automates déterministes complets, alors $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.*

Idée de la démonstration.
— $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$: remarque
— $\mathcal{L}(\mathcal{A}_1) \supseteq \mathcal{L}(\mathcal{A}_2)$: surjectif qui assure que la fonction de transition sur la fonction inverse est bien définie et le complet assure qu'une transition inverse existe pour toute lettre.

□

Définition. On définit la relation d'ordre $\preceq$ sur l'ensemble des automates déterministes complets par : $\mathcal{A}_1 \preceq \mathcal{A}_2$ s'il existe un morphisme ϕ surjectif.

Définition. Un automate est dit minimal s'il est minimal pour $\preceq$.

Caractérisation de l'automate minimal La notion de morphisme, même si elle donne un cadre formel à notre étude n'est pas facile à manipuler. Nous allons donc donner d'autres caractérisations de la minimalité d'un automate qui sont basées sur les points de vus que nous avons de cette minimalité.

Automate des résiduels Une première manière de caractériser l'automate minimal est d'utiliser le point de vu intrinsèque au langage : la notion de quotient qui est la notion de langage et d'automate résiduel.

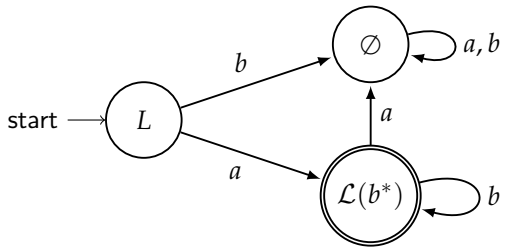

FIGURE 2 – Un automate des résiduels pour $L = \mathcal{L}(ab^*)$.

Définition. Soit $L \subseteq \Sigma^*$, $u \in \Sigma^*$. On définit le résiduel de L par u comme le langage $u^{-1}L = \{v \in \Sigma^* | uv \in L\}$.

Définition. Soit $L \subseteq \Sigma^*$. On définit l'automate des résiduels de L par $\mathcal{R}(L) = \{Q_L, \delta_L, I_L, F_L\}$ avec $Q_L = \{u^{-1}L | u \in \Sigma^*\}$, $\delta_L(u^{-1}L, a) = a^{-1}u^{-1}L = (ua)^{-1}L$, $I_L = L = \epsilon^{-1}L$ et $F_L = \{u^{-1}L | u \in L\} = \{u^{-1}L | \epsilon \in u^{-1}L\}$.

Proposition (Caractérisation de la minimalité par les résiduels)**.** *L est reconnaissable si et seulement si L possède un nombre fini de résiduels. De plus, dans ce cas, $\mathcal{R}(L)$ est minimal.*

Équivalence de Nérode Le deuxième point de vu qui est plus calculatoire est la congruence de Nérode. Cette congruence se fait sur une relation d'équivalence qui permet de déterminer si deux états sont inséparables ou non.

Définition. Soit $\mathcal{A}$ un automate fini déterministe. Une relation d'équivalence $\sim$ sur Q est une congruence si :

— $\forall p, q,\ p \sim q \Rightarrow \forall a \in \Sigma, \delta(p,a) \sim \delta(q,a)$ (comptabilité avec δ)

— $\forall p, q,\ p \sim q \Rightarrow (p \in F \Leftrightarrow q \in F)$ (saturation de F)

Remarque. La congruence de Nérode est la plus grossière respectant la définition de la congruence.

Définition. Si $\mathcal{A}$ un automate fini déterministe et $\sim$ est une relation d'équivalence sur Q, on définit le quotient de $\mathcal{A}$ par $\sim$ comme : $\mathcal{A}/\sim = (Q/\sim, \delta_\sim, \{[i]\}, \{[f] | f \in F\})$ où $\delta_\sim([p], a) = [\delta(p), a]$.

Remarque. Cette définition est valide pour toute relation d'équivalence.

Proposition. *On a* $\mathcal{L}(\mathcal{A}/\sim) = \mathcal{L}(\mathcal{A})$.

Définition. Soit $\mathcal{A}$ un automate fini. L'équivalence sur Q définie par $p \equiv q$ si et seulement si $\forall w \in \Sigma^* \delta(p,w) \in F \Leftrightarrow \delta(p,w) \in F$ est appelée congruence de Nérode.

Proposition. *L'automate quotient obtenu par la congruence de Nérode* $\mathcal{A}/\equiv$ *est isomorphe à l'automate des résiduels* $\mathcal{R}(L)$.

Calcul de l'automate minimal Nous avons énoncé quelques caractérisation de cet automate minimal. Il nous reste à étudier comment le calculer à partir d'un automate fini qui n'est pas nécessairement optimal. Cette construction peut se faire à partir de la construction de Moore que nous allons explicité. L'implémentation naïve de cette construction donne l'algorithme de Moore [3, p.124]. Puis nous étudierons l'algorithme de Hopcroft qui est une implémentation élaborée de cette méthode [1, p.318].

Construction de Moore Soit $\mathcal{A}$ un automate déterministe. Pour calculer son automate minimal, il suffit de calculer l'équivalence de Nérode. Pour cela, nous allons l'approcher successivement par l'équivalence $\sim_k$ dont la limite sera celle de Nérode.

Définition. Soit $k \in \mathbb{N}$, on définit l'équivalence suivante sur Q d'un automate déterministe $\mathcal{A}$:

$$p \sim_k q \Leftrightarrow L_p^{(k)} = L_q^{(k)}$$

avec $L_p^{(k)} = \{w \in L_p \mid |w| \leq k\}$.

Proposition. *Pour tout entier* $k \geq 1$, *on a*

$$p \sim_k q \Leftrightarrow p \sim_{k-1} q \text{ et } (\forall a \in A, pa \sim_{k-1} qa)$$

Démonstration. On a

$$\begin{array}{rcll} L_p^{(k)} & = & \{p.w \in T \mid |w| \leq k\} & \text{Définition} \\ & = & \{p.w \in T \mid |w| \leq k-1\} \cup \bigcup_{a \in A} a\{v \mid (pa)v \in T \text{ et } |v| \leq k-1\} & \text{Cas} \\ & = & L_p^{(k-1)} \cup \bigcup_{a \in A} aL_{pa}^{(k-1)} & \text{Traduction} \end{array}$$

On conclut en traduisant par leur définition ces égalités. □

Corollaire. *Si les équivalences* $\sim_k$ *et* $\sim_{k+1}$ *coïncident, alors les équivalence* $\sim_{k+l}$ *avec* $l \geq 0$ *sont toutes égales et égales à l'équivalence de Nérode.*

Démonstration. Par récurrence (en appliquant la proposition précédente), on a l'égalité des équivalence. La deuxième assertion provient de la congruence de Nérode : $p \sim q \Leftrightarrow p \sim_k q\ \forall k \in \mathbb{N}$. □

Proposition. *Si* $\mathcal{A}$ *est un automate à n états, l'équivalence de Nérode de* $\mathcal{A}$ *est égale à* $\sim_{n-2}$

Démonstration. Si pour $k \geq 0$, les équivalence $\sim_{k-1}$ et $\sim_k$ sont distinctes, le nombre de classe d'équivalence $\sim_k$ est inférieure où égale à $k+2$. □

Algorithme de Hopcroft L'algorithme de Hofcroft (algorithme 6) permet d'implémenter de manière astucieuse cette équivalence inductive.

Algorithm 6 Algorithme de Hopcroft permettant de calculer un automate minimal.

Entrée : $\mathcal{A}$ est un automate déterministe complet et émondé

```
1:  function HOPCROFT(A)
2:      P ← (F, Q \ F)
3:      S ← {(min(F, Q \ F), a) | a ∈ A}
4:      while S ≠ ∅ do
5:          Choisir (C, a) ∈ S
6:          S ← S \ (C, a)
7:          for B ∈ P do
8:              Couper B par (C, a) en B1, B2
9:              Remplacer B par B1, B2 dans P
10:             for b ∈ Σ do
11:                 if (B, b) ∈ S then
12:                     Remplacer (B, b) par (B1, b), (B2, b) dans S
13:                 else
14:                     Ajouter (min(B1, B2), b) à S
15:                 end if
16:             end for
17:         end for
18:     end while
19: end function
```

L'algorithme naïf qui applique la méthode de Moore s'exécute en $O(mn^2)$ où m est la taille de l'alphabet et n le nombre d'étapes de calcul. L'algorithme de Hopcroft nous permet d'obtenir une exécution en temps en $O(mn \log n)$.

Principe : Calcul de la congruence de Nérode par raffinement successifs utilisant le paradigme diviser pour régner.

Définition (Partie stable [1, p.320])**.** Soit $\mathcal{P}$ une partition de Q, A, B deux éléments de $\mathcal{P}$ et a un élément de Σ.
On dit que A est stable pour (B, a) si $Aa \subset B$ ou $Aa \cap B = \emptyset$ avec $Aa = \{qa \mid q \in A\}$. Sinon, la paire (B, a) coupe A en deux parties $A_1 = \{q \in A \mid qa \in B\}$ et $A_2 = \{q \in A \mid qa \notin B\}$.

Théorème. *Soit Σ un alphabet à m lettres et $\mathcal{A}$ un automate à n états sur cet alphabet. L'algorithme* HOPCROFT *(Algorithme 6) calcule, dans le pire cas, en temps $O(mn \log n)$ l'automate minimal de $\mathcal{A}$.*

Arguments de la démonstration. Terminaison :

- On munit l'ensemble $\{\text{Partition de } Q\} \times (\mathcal{P}(\mathcal{P}(Q))) \times \Sigma$ de l'ordre bien fondé $\leq$ définie telle que $(\mathcal{P}, S) \leq (\mathcal{P}', S')$ si $|\mathcal{P}| > |\mathcal{P}'|$ et $|S| \leq |S'|$.
- $\leq$ ordre bien fondé : si $\mathcal{P}$ reste stable, S diminue et il ne peut pas augmenter.

Correction : La partition donnée par Nérode est toujours plus fine que $\mathcal{P}$. On montre que, lorsque l'algorithme termine, $\mathcal{P}$ est stable pour (P, a) avec $P \in \mathcal{P}$ et $a \in \Sigma$.

Notations : $\mathcal{P}_i$ la valeur de $\mathcal{P}$ avant la $i^{\text{ième}}$ itération et $\mathcal{P}_N$ la dernière valeur de $\mathcal{P}$ et

$$B \lhd_a C = \begin{cases} \{B\} & \text{si } B \text{ est stable par} (C, a) \\ \{B_1, B_2\} & \text{sinon} \end{cases}$$

Lemme. *Soit $C_1 \sqcup C_2$, $a \in \Sigma$.*

1. *B stable par (C_1, a) et (C_2, a) implique que B est stable par (C, a).*
2. *B stable par (C_1, a) et (C, a) implique que B est stable par (C_2, a).*
3. *$(B \lhd_a C) \lhd_a T = (B \lhd_a T) \lhd_a C$.*

Arguments de la démonstration.

1. Distinction de cas
2. Distinction de cas
3. $(B \lhd_a C) \lhd_a T$ et $(B \lhd_a T) \lhd_a C$ sont composés d'éléments non vide de $B \cap a^{-1}C \cap b^{-1}T$, $B \cap a^{-1}\overline{C} \cap b^{-1}T$, $B \cap a^{-1}\overline{C} \cap b^{-1}\overline{T}$ et $B \cap a^{-1}C \cap b^{-1}\overline{T}$.

□

Lemme. *Soit $P \in \mathcal{P}$, $a \in \Sigma$, $(P, a) \notin S$, alors $\mathcal{P}_N$ est stable pour (P, a).*

Arguments de la démonstration. On raisonne par induction

Initialisation Si $(P, a) \in S_1$, $(\overline{P}, a) \notin S_1$ et $\mathcal{P}_N$ sera stable pour $(\overline{P}, a)$ donc pour (P, a) (par 1 et 3 du lemme).

Hérédité Soit $P \in \mathcal{P}_{k+1}$ tel que $(P, a) \notin S_{k+1}$.

- Si $P \in \mathcal{P}_k$, alors si $(P, a) \notin S_k$, ok. Sinon $(P, a) \in S_k$, on a alors retirer (P, a) de S_k. Donc $\mathcal{P}_N$ est stable pour (P, a).
- Si $P \notin \mathcal{P}_k$, alors il existe C, b tels que $R \lhd_b C = (P, P')$. Si $(R, a) \notin S_k$, alors $(P', a) \in S_{k+1}$ et $\mathcal{P}_N$ est stable pour (R, a) et (P', a), donc pour (P, a). Sinon, on a retiré (R, a) et S_{k+1} est stable pour (R, a). Donc $(P, a) \in S_{k+1}$ et $\mathcal{P}_N$ est stable pour (P, a).

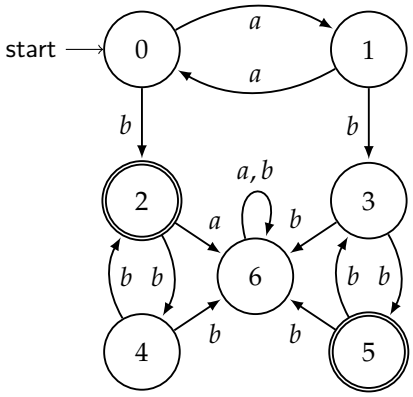

FIGURE 3 – Automate à minimiser

r_0 $\{ \underset{AB}{0}, \underset{AA}{1}, \underset{AB}{3}, \underset{AB}{4}, \underset{AA}{6} \}_A \{ \underset{AA}{2}, \underset{AA}{5} \}_B$

r_1 $\{ \underset{BA}{1}, \underset{AA}{6} \}_A \{ \underset{AC}{0}, \underset{AC}{3}, \underset{AC}{4} \}_B \{ \underset{AB}{2}, \underset{AB}{5} \}_C$

r_2 $\{1\}_A \{6\}_B \{ \underset{AD}{0}, \underset{AD}{3}, \underset{BD}{4} \}_C \{ \underset{BC}{2}, \underset{BC}{5} \}_D$

r_3 $\{1\}_A \{6\}_B \{0\}_C \{ \underset{BE}{3}, \underset{BE}{4} \}_D \{ \underset{BD}{2}, \underset{BD}{5} \}_E$

r_4 $\{1\}_A \{6\}_B \{0\}_C \{3,4\}_D \{2,5\}_E$

FIGURE 4 – Exemple de calcul par l'algorithme de Hopcroft (Algorithme 6). En vert, on donne la partition dans laquelle on arrive si on lit un a et en marron celle si on lit un b.

□

Complexité temporelle : $O(mn \log n)$

— La boucle `tant que` s'effectue $O(nm)$ fois (méthode de l'agrégat).

— Le nombre de fois que l'on supprime un élément de S est au plus $\log n$.

□

Algorithme par renversement Il existe un autre algorithme pour calculer un automate minimal qui n'utilise pas la congruence de Nérode [3, p.125]. On rappelle la définition d'un automate co-accessible : $\exists f \in F, \exists w \in \Sigma^*$ avec $f \in \delta(q, w)$ et co-déterministe est un automate déterministe si on inverse les transitions.

Proposition. *Soit $L \in Rec(\Sigma^*)$. Le déterminisé d'un automate co-déterministe co-accessible qui reconnaît L est minimal.*

Définition. Soit $\mathcal{A} = (Q, \delta, I, F)$. Le miroir de $\mathcal{A}$ est $mir(\mathcal{A}) = (Q, \delta^t, F, I)$ où $\delta^t(p, a) = \{q \mid p \in \delta(q, a)\}$.

Proposition (Algorithme de Brzozowski)**.**

$$R(\mathcal{L}(\mathcal{A})) = det(mir(det(mir(\mathcal{A}))))$$

Applications

— Algorithmes de Hopcroft et de Moore

— *Appli* : Équivalence de langage + Bi-simulation.

Références

[1] D. Beauquier, J. Berstel, and P. Chrétienne. *Eléments d'algorithmique*. Manuels informatiques Masson. Masson, 1992.

[2] Rivest R. Stein C. Cormen T., Leiserson C. *Algorithmique, 3ème édition*. Dunod, 2010.

[3] J. Sakarovitch. *Éléments de la théorie des automates*. Vuibert informatique, 2003.