Leçon 909 : Langages rationnels et automates finis. Exemples et applications.

Julie Parreaux

2018 - 2019

Références pour la leçon

[1] Beauquier, Berstel et Chretienne, *Éléments d'algorithmique.*
[2] Carton, *Langages formels, calculabilité et complexité.*
[3] Floyd et Biegel, *Le langage des machines.*
[4] Sakarovitch, *Éléments de la théorie des automates.*
[5] Wolper, *Introduction à la calculabilité.*

Développements de la leçon

Le problème PSA est NP-complet

L'automate minimal de la recherche de motif

Plan de la leçon

Motivation

Défense

La théorie des langages reconnaissable apparaît dans les années 1940 avec une première modélisation de neurone grâce à des automates finis. Ensuite, dès 1956, la théorie des langages rationnels et reconnaissables sont uniformiser grâce aux théorème de Kleene (en 1956, on a également leur minimisation et en 1958, on a leur caractérisation). Il faudra attendre les années 1959 pour qu'un premier article posant les bases de la théorie des automates telles que nous la connaissons soit posé (cela à valu à leur auteur un prix Turing).

Les automates finis (qui sont les premières machines) ont été trouvé leurs essor dans les années 1980 pour commencer l'automatisation de certaine tâches comme l'analyse syntaxique lors de la compilation. Cependant leurs applications ne se limitent pas à l'informatique (avec la compilation ou le traitement de texte), ils apparaissent également dans la théorie linguiste en décrivant la morphologie d'une langue. En informatique théorique, ils sont très facile à manipuler. Les langages rationnels sont les premiers langages de la hiérarchie de Chomsky. Ils sont les langages les moins expressifs mais ils nous permettent de répondre à de nombreux problèmes de décisions. Il est difficile de trouver des problèmes de décision sur ces langages qui ne soit pas décidable.

Ce qu'en dit le jury

Pour cette leçon très classique, il importe de ne pas oublier de donner exemples et applications, ainsi que le demande l'intitulé.

Une approche algorithmique doit être privilégiée dans la présentation des résultats classiques (déterminisation, théorème de Kleene, etc.) qui pourra utilement être illustrée par des exemples. Le jury pourra naturellement poser des questions telles que : connaissez-vous un algorithme pour décider de l'égalité des langages reconnus par deux automates ? quelle est sa complexité ?

Des applications dans le domaine de l'analyse lexicale et de la compilation entrent naturellement dans le cadre de cette leçon.

Introduction

Motivation : Lien avec l'étude des langages (les comprendre) ; Automatiser des processus simples.
Cadre : Σ alphabet fini ; L un langage sur cet alphabet.

1 Langages rationnels [2, p.38]

— *Définition* [2, p.17] : opérations rationnelles (union, concaténation, étoile) + *Exemple* : $\{a, b\}^*$.

— *Définition* : langages rationnels + *Exemples* (ne pas oublier le langage fini)

— *Définition* : Expression rationnelles + *Exemples* + *Remarque* : parenthésage et ambiguïté (On amène le monde de la syntaxe ici (on réintroduit un nouveau niveau). Les expressions rationnelles sont aux opérateurs rationnelles ce que sont les expressions arithmétiques pour leurs opérateurs.)

— *Proposition* : les langages rationnels sont ceux induits par les expressions rationnelles.

2 Langages reconnaissables [2, p.39]

2.1 Automate fini

— *Définition* : automate fini + *Exemple* (Les automates sont des machines de Turing particulière : sans mémoire, le ruban est en lecture seul en une seule passe)

— *Définition* : langage accepté + *Exemple*

— *Définition* : langage reconnaissable + *Exemple*

— *Application* : Décidabilité de Presburger

2.2 Quelques automates particuliers

— *Définitions* : automate déterministe [2, p.46] [4, p.114]/ complet / émondé [4, p.74]

— *Proposition* [2, p.46] : Équivalence avec ces différentes formes d'automates.

— *Remarque* [1, p.299] : complexité exponentielle (nombre d'états) de la déterminisation + *Exemple* : Automate des parties (optimal) + Contre-exemple

— *Définition* [3, p.555] : Problème de la séparation de deux automates

— *Proposition* : Problème de la séparation est NP-complet. DEV

2.3 Équivalence de ces langages [2, p.40]

— *Théorème* : de Kleene

— Passage des automates aux expressions ; McNaughton–Yamada ; Brzozowski–McCluskey ; élimination de Gauss

— *Lemme* : d'Arden

— Passage des expressions aux automates : Thomson

— *Appli* : analyse lexicale Première étape de la compilation.

— *Appli* : recherche d'un motif par une expression régulière (on construit son automate par Thomson, déterminiser, applique) ; Analyse lexicale

3 La classe des rationnels

Cette famille de langage est très agréable à manipuler.On souhaite connaître quelques unes de ces propriétés. On souhaite également alors savoir quels langages appartiennent à cette famille.

3.1 Stabilité

La classe des rationnels est très stable.

— *Proposition* [4, p.66,p.97,p.118] : stabilité par intersection, passage au complémentaire, et opérations rationnelles

— *Proposition* : stabilité par morphisme [2, p.58]

— *Remarque* : L'inclusion n'est pas stable Σ^* est rationnel mais pas $a^n b^n$ ne l'est pas.

3.2 Grammaire régulière (linéaire à droite) [5]

Idée : Les automates finis (langages rationnels) sont la première marche de la hiérarchie de Chomsky.

— *Définition* : Grammaire / Langage engendré + *Exemple*
— *Définition* : Grammaire régulière
— *Proposition* : Caractérisation des langages rationnels.

3.3 Caractérisation

Mais qui y ait ?

— *Lemmes* [4, p78] : de l'étoile
— *Contre-exemple* : $a^n b^n$
— *Théorème* [4, p.128] : Ehrenfeucht, Parikh, Rozenberg
— *Lemme* : étoile par bloc
— *Proposition* : caractérisation des langages reconnaissables + *Application* : on montre qu'un langage n'est pas reconnaissable exemple

3.4 Décidabilité et langages réguliers

Idée : il est très difficile de trouver des problèmes indécidables sur les langages rationnels.

— *Problèmes décidables* : mot, vide, universalité, ...
— *Remarque* : problème universelle est PSPACE-complet

4 Minimisation [4, p.120]

Existe-t-il un représentant canonique pour reconnaître un langage ?

4.1 Morphismes d'automates

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

— *Définition* [4, p.120] : Morphisme d'automate + *Exemple*
— *Proposition* : Morphisme surjectif sur automates déterministes complets alors ils reconnaissent le même langage. Une bijection par morphisme d'automate définie un isomorphisme entre ces automates à numérotation près.
— *Définitions* : relation d'ordre et minimisation

4.2 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.

Automate des résiduels [1, p.312] Quelques fois appelé quotient : point de vu intrinsèque au langage.

— *Définition* : automate des résidus + *Exemple*
— *Proposition* : caractérisation de l'automate minimal par les résidus

Équivalence de Nérode Autre point de vu : plus calculatoire.

— *Définitions* : Équivalence de Nérode et automate quotient

— *Proposition* : équivalence des langages

— *Définition* : congruence de Nérode + *Proposition* : caractérisation de l'automate minimale

— *Exemple* : Automate des occurrences est l'automate minimal de $\Sigma^* P$ + *Application* : recherche de motif dans un texte (MP-KMP DEV)

4.3 Calcul de l'automate minimal

Comment calculer cet automate minimal ?

— Construction de Moore et algorithme de Hopcroft (Algorithme 1) [1, p.325]

— Algorithme par renversement [4, p..125]

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

Ouverture

Fermeture transitive ?

Quelques notions importantes

Automate minimal

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 [4, 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.

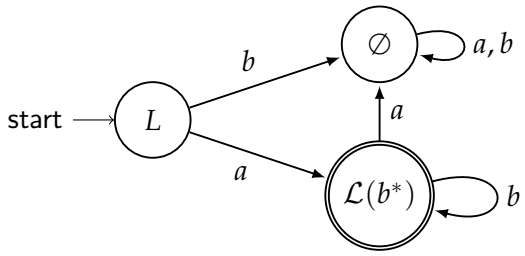

FIGURE 1 – Un automate des résiduels pour $L = \mathcal{L}(ab^*)$.

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 [4, 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

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

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 1) permet d'implémenter de manière astucieuse cette équivalence inductive.

Algorithm 1 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 1) 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

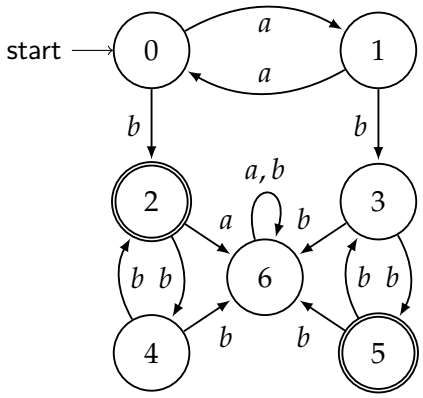

FIGURE 2 – Automate à minimiser

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 3 – Exemple de calcul par l'algorithme de Hopcroft (Algorithme 1). En vert, on donne la partition dans laquelle on arrive si on lit un a et en marron celle si on lit un b.

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).

□

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 [4, 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] O. Carton. *Langages formels, calculabilité et complexité*. Vuibert, 2008.

[3] R. Floyd and R. Biegel. *Le langage des machines*. International Thomson Publishing, 1994.

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

[5] P. Wolper. *Introduction à la calculabilité*. Dunod, 2006.