Notes de révision : Automates et langages

Sylvain SCHMITZ
LSV, ENS Cachan & CNRS
Version du 26 octobre 2009 (r32)

Automates et langages

Ces notes de révisions sont destinées aux candidats à l'agrégation de mathématiques qui ont choisi l'option D d'informatique à l'oral. Les notes couvrent (pour l'instant) l'essentiel du programme de l'option en matière d'automates et de langages rationnels, et seront complétées un jour...

Les notes de révision ne se substituent pas à la lecture des ouvrages classiques sur le sujet, mais offrent des renvois à ces ouvrages et les complètent avec des exemples et des applications plus variées. En particulier, on ne trouvera pas en ces pages une seule démonstration complète, mais seulement des justifications rapides, qui sont à proscrire lors d'un développement oral.

Programme officiel

Le programme comporte d'après le document du site de l'agrégation `http://agreg.org/indexmaths2010.html` :

1. Automates finis. Langages reconnaissables. Lemme d'itération. Existence de langages non reconnaissables. Automates complets. Automates déterministes. Algorithme de déterminisation. Propriétés de clôture des langages reconnaissables.
2. Expressions rationnelles. Langages rationnels. Théorème de KLEENE.
3. Automate minimal. Résiduel d'un langage par un mot. Algorithme de minimisation.
4. Utilisation des automates finis : recherche de motifs, analyse lexicale.
5. Langages algébriques. Lemme d'OGDEN. Existence de langages non algébriques. Grammaires algébriques. Propriétés de clôture des langages algébriques.
6. Automates à pile. Langages reconnaissables par automates à pile.
7. Utilisation des automates à pile : analyse syntaxique. Grammaires LL(1).

Les leçons correspondantes sont :

Leçon 907 Algorithmique du texte : exemples et applications.

Leçon 908 Automates finis. Exemples et applications.

Leçon 909 Langages rationnels. Exemples et applications.

Leçon 910 Langages algébriques. Exemples et applications.

Leçon 911 Automates à pile. Exemples et applications.

Leçon 923 Analyses lexicale et syntaxique : applications.

Si le programme de l'option D porte globalement sur l'informatique fondamentale, on notera tout de même l'insistance des jurys à demander des exemples et applications pour les résultats assez théoriques du thème. C'est que ces résultats, souvent sous la forme de preuves *constructives*, ne se réduisent pas à des théorèmes sur des objets mathématiques, mais sont réellement appliqués dans de nombreux domaines. Deux questions sont à se poser systématiquement face à un résultat théorique en informatique :

1. *« À quoi cela sert-il ? »*, quelles sont les applications de ce résultat ?
2. *« Quel est le coût ? »*, quelle est la complexité théorique de cette construction ?

Bibliographie recommandée

Tout n'est pas forcément disponible dans toutes les bibliothèques universitaires, mais au moins une partie de ces ouvrages font partie de la bibliothèque de l'agrégation, et sont donc disponibles pendant les trois heures de préparation des candidats. Des renvois vers ces ouvrages sont systématiques dans les notes de révision.

Il est peut-être utile de rappeler que les seuls documents autorisés lors de la préparation de l'oral sont des *livres* largement diffusés, de manière à préserver l'égalité des chances entre les candidats. En particulier, ces notes de révision ne peuvent pas être utilisées lors de la préparation elle-même, mais ont vocation à l'être en amont lors du travail de révision à proprement parler.

[ASU86] Alfred AHO, Ravi SETHI et Jeffrey ULLMAN. *Compilateurs : Principes, techniques et outils*. Dunod, 2000. Traduit de *Compilers*, Addison Wesley, 1986.

[Aut94] Jean-Michel AUTEBERT. *Théorie des langages et des automates*. Masson, 1994.

[Car08] Olivier CARTON. *Langages formels, calculabilité et complexité*. Vuibert, 2008.

[GBJL00] Dick GRUNE, Henri E. BALS, Ceriel J.H. JACOBS et Koen G. LANGENDOEN. *Compilateurs*. Dunod, 2002. Traduit de *Modern Compiler Design*, John Wiley & Sons, 2000.

[Har78] Michael A. HARRISON. *Introduction to Formal Language Theory*. Addison Wesley, 1978.

[HU79] John E. HOPCROFT et Jeffrey D. ULLMAN. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[Sak03] Jacques SAKAROVITCH. *Éléments de théorie des automates*. Vuibert, 2003.

[SSS88] Seppo SIPPU et Eljas SOISALON-SOININEN. *Parsing Theory*, Vol. I: *Languages and Parsing*. EATCS Monographs on Theoretical Computer Science volume 15, Springer, 1988.

[SSS90] Seppo SIPPU et Eljas SOISALON-SOININEN. *Parsing Theory*, Vol. II: *LR(k) and LL(k) Parsing*. EATCS Monographs on Theoretical Computer Science volume 20, Springer, 1990.

Contenu des notes de révision

1 Automates et langages rationnels **5**
1.1 Langages reconnaissables 5
1.1.1 Propriétés élémentaires 6
1.1.2 Automates déterministes 7
1.1.3 Propriétés de clôture 9
1.1.4 Lemmes d'itération 12
1.2 Langages rationnels 13
1.2.1 Des expressions aux automates 14
Construction de THOMPSON 14
Automate standard de GLUSHKOV 15
Expressions dérivées partielles d'ANTIMIROV 18
1.2.2 Des automates aux expressions 22
Construction de MCNAUGHTON et YAMADA 22
Construction de BRZOZOWSKI et MCCLUSKEY 23
Lemme d'ARDEN et élimination de GAUSS 25
Complexité des expressions rationnelles 26
1.3 Minimisation 28
1.3.1 Automate des quotients 29
1.3.2 Algorithme par renversement de BRZOZOWSKI 30
1.3.3 Congruence de NERODE 31
1.3.4 Algorithme de MOORE 32
1.4 Applications 33
1.4.1 Localisation 33
1.4.2 Analyse lexicale 37
1.4.3 Linguistique 41

2 Références supplémentaires **43**

Chapitre 1

Automates et langages rationnels

Le programme de l'agrégation pour les automates et langages rationnels tourne essentiellement autour du théorème de KLEENE :

Théorème 1.1 (KLEENE [18])**.** *Soit Σ un alphabet fini. Un langage de Σ^* est rationnel si et seulement si il est reconnu par un automate fini sur Σ :*

$$Rat(\Sigma^*) = Rec(\Sigma^*)$$

C.f. [Car08, p. 36], [Sak03, p. 94], [Aut94, p. 52], [SSS88, p. 82], [HU79, p. 30], [Har78, p. 57]. Le théorème n'est pas vérifié pour tout monoïde [Sak03, p. 261].

On s'attachera donc particulièrement à connaître les preuves des deux inclusions

- **Rat**$(\Sigma^*) \subseteq$ **Rec**(Σ^*) par la construction de THOMPSON, celle de GLUSHKOV, ou celle d'ANTIMIROV (section 1.2.1),
- **Rec**$(\Sigma^*) \subseteq$ **Rat**(Σ^*) par la construction de MCNAUGHTON et YAMADA, ou celle de BRZOZOWSKI et MCCLUSKEY, ou par résolution d'un système d'équations et le lemme d'ARDEN (section 1.2.2).

Il n'est pas nécessaire de maîtriser toutes ces constructions, mais il faut en connaître une dans chaque direction du théorème !

Au résultat central qu'est le théorème de KLEENE s'ajoutent des résultats périphériques de

1. clôture par diverses opérations (section 1.1.3),
2. lemmes d'itération (section 1.1.4), et
3. minimisation (section 1.3).

Ces résultats sont à connaître puisqu'ils sont au programme et sont donc susceptibles de faire l'objet de questions à l'oral. L'introduction d'un ou plusieurs de ces points dans un plan d'une leçon **908** ou **909** est utile, en veillant à ne pas oublier pour autant les exemples et applications.

1.1 Langages reconnaissables

Définition 1.2 (Automate fini)**.** Un *automate fini* sur un alphabet Σ est un tuple $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ où Q est un ensemble fini d'états, $I \subseteq Q$ l'ensemble des états initiaux, $F \subseteq Q$ l'ensemble des états finals, et $\delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times Q$ la relation de transition. Alternativement, on peut voir δ comme une fonction de $Q \times \Sigma \cup \{\varepsilon\}$ dans 2^Q l'ensemble des parties de Q.

On étend naturellement la définition de δ sur $Q \times \Sigma^* \times Q$ par $\delta^*(q, \varepsilon) = q$ et $\delta^*(q, aw) = \bigcup_{q' \in \delta(q,a)} \delta(q', w)$. Par un abus de notation bien pratique, on peut aussi

C.f. [Car08, sec. 1.5.2][Sak03, sec. I.1], [Aut94, ch. 4], [SSS88, sec. 3.2], [HU79, sec. 2.2–2.4], [Har78, ch. 2].

l'étendre à des ensembles d'états ou de mots par l'union $\delta^*(E, L) = \bigcup_{q\in E, u\in L} \delta^*(q, u)$. Enfin, on peut définir similairement la relation inverse $\delta^{-1} = \{(q, a, p) \mid (p, a, q) \in \delta\}$.

Définition 1.3 (Langage reconnaissable)**.** Le langage reconnu par un automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta\rangle$ est

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \exists i \in I, f \in F, f \in \delta^*(i, w)\}$$

Deux automates finis sur Σ sont *équivalents* s'ils reconnaissent le même langage de Σ^*.

L'autre définition de Rec(Σ^*) utilise la notion de reconnaissance par monoïde, voir [Car08, sec. 1.11], [Sak03, p. 256], [Aut94, p. 47].

Un langage L sur Σ^* est *reconnaissable* s'il existe un automate fini sur Σ tel que $L = L(\mathcal{A})$. L'ensemble des langages reconnaissables sur Σ^* est noté Rec(Σ^*).

1.1.1 Propriétés élémentaires

On rappelle ici quelques propriétés élémentaires sur les automates.

Définition 1.4 (Automate accessible)**.** L'ensemble des états accessibles d'un automate fini est $\delta^*(I, \Sigma^*)$. Un automate fini est *accessible* si tous ses états sont accessibles, c'est-à-dire si, pour tout état q de Q, il existe un mot u de Σ^* et un état initial i de I tels que $q \in \delta^*(i, u)$.

Proposition 1.5. *Pour tout automate fini sur Σ, il existe un automate fini accessible équivalent.*

Démonstration. Étant donné un automate fini $\mathcal{A} = \langle \Sigma, Q, I, F, \delta\rangle$, on définit l'automate $\mathcal{A}' = \langle \Sigma, Q', I, F, \delta'\rangle$ avec

$$\begin{aligned} Q' &= \delta^*(I, \Sigma^*) \\ \delta' &= \delta \cap (Q' \times \Sigma \cup \{\varepsilon\} \times Q') \end{aligned}$$

□

Un calcul d'accessibilité dans un automate se réduit à un calcul d'accessibilité dans un graphe orienté, qui n'est lui-même qu'un calcul de l'image d'un ensemble par la fermeture réflexive transitive de la relation R d'adjacence du graphe. En pratique, on utilise pour ce type de problème soit un simple parcours en profondeur depuis les états initiaux en $O(|I| \cdot |\delta|)$, soit des algorithmes plus avancés utilisant les composantes fortement connexes du graphe (algorithme de TARJAN [27] en $O(t \cdot \max(|S|, |R|))$ où S est l'ensemble des sommets du graphe, R la relation d'adjacence, et t le coût algorithmique d'une opération d'union entre deux parties de S, ce que l'on tend à approximer par $O(|\mathcal{A}|)$ en posant $|\mathcal{A}| = \max(|Q|, |\delta|)$).

C.f. [SSS88, ch. 2], [Sak03, exercice I.1.20]. On rappelle aussi que le problème de l'accessibilité dans un graphe orienté, c'est-à-dire de déterminer pour deux sommets quelconques p et q s'il existe un chemin orienté qui relie p à q, est l'exemple classique avec 2SAT d'un problème NLOGSPACE*-complet [Car08, théorème 4.46], [HU79, sec. 13.5], [21, ch. 16].*

On définit de manière similaire un automate *co-accessible* comme un automate où tout état peut accéder à au moins un état final. On observe aisément qu'il suffit de vérifier l'accessibilité de l'automate *transposé* qui échange I et F et utilise δ^{-1} comme relation de transition, et donc que pour tout automate fini on peut construire un automate co-accessible équivalent. En combinant accessibilité et co-accessibilité, on définit les automates émondés.

Définition 1.6 (Automate émondé)**.** Un automate fini est *émondé* s'il est accessible et co-accessible.

Théorème 1.7 (Vide)**.** *On peut décider si un langage reconnaissable est vide en temps polynomial.*

Démonstration. Soit $L = L(\mathcal{A})$ un langage reconnaissable. L est l'ensemble vide si et seulement si l'automate émondé de $\mathcal{A}$ a un ensemble vide d'états. □

Une autre construction qui utilise les résultats de complexité sur les graphes permet d'éliminer les ε-transitions, ce que l'on supposera ensuite chaque fois que cela sera utile.

À l'inverse, l'universalité d'une expression rationnelle (et donc d'un automate non déterministe) est un problème PSPACE*-complet (*STOCKMEYER *et* MEYER *[26], [SSS90, p. 392]).*

Proposition 1.8. *Pour tout automate fini sur Σ, il existe un automate fini sans ε-transition équivalent.*

Démonstration. Étant donné un automate fini $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$, on définit un automate $\mathcal{A}' = \langle \Sigma, Q, I, F', \delta' \rangle$ tel que

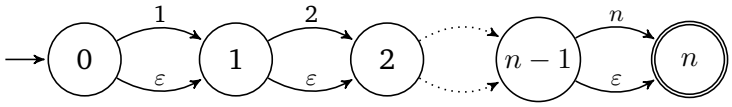
$$F' = \{q \in Q \mid \delta^*(q, \varepsilon) \cap F \neq \emptyset\}$$
$$\delta' = \{(p, a, q) \mid \exists q' \in Q, \delta^*(p, \varepsilon, q') \wedge (q', a, q) \in \delta\}$$

□

En utilisant une construction inspirée par celle de GLUSHKOV*, on peut construire un automate fini sans ε-transitions de taille $O(n(\log n)^2)$ pour toute expression rationnelle de taille n, et donc pour l'expression $(1+\varepsilon)(2+\varepsilon)\cdots(n+\varepsilon)$ correspondant à l'automate de l'exemple 1.9 [16].*

Comme précédemment, on peut réduire ce calcul à celui de la clôture réflexive transitive d'une relation, ici $\{(p, q) \mid (p, \varepsilon, q) \in \delta\}$. La complexité de la construction est alors en $O(|Q|^2)$.

Exemple 1.9. On observe réellement cette complexité quadratique sur l'automate fini sur l'alphabet $\{1, \ldots, n\}$ suivant



1.1.2 Automates déterministes

Les définitions suivantes supposent des automates sans ε-transitions.

Définition 1.10 (Automate complet)**.** Un automate fini est *complet* si, pour tout état q de Q et pour tout symbole a de Σ, $\delta(q, a) \neq \emptyset$.

Proposition 1.11. *Pour tout automate fini sur Σ, il existe un automate fini complet équivalent.*

Démonstration. Étant donné un automate fini $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$, on définit un automate $\mathcal{A}' = \langle \Sigma, Q \uplus \{p\}, I, F, \delta' \rangle$ avec

$$\delta' = \delta \cup \{(p, a, p) \mid a \in \Sigma\} \cup \{(q, a, p) \mid q \in Q, a \in \Sigma, \delta(q, a) = \emptyset\}$$

□

Le calcul d'un automate complet est manifestement en $O(|Q| \cdot |\Sigma|)$. L'état p est appelé un état *puit*.

On doit en fait la notion d'automates non déterministes et leur conversion en automates déterministes à RABIN *et* SCOTT *[22].*

Définition 1.12 (Automate déterministe)**.** Un automate fini est *déterministe* si $|I| \leq 1$ et pour tout état q de Q et pour tout symbole a de Σ, $|\delta(q, a)| \leq 1$.

Proposition 1.13. *Pour tout automate fini sur Σ, il existe un automate fini déterministe équivalent.*

Démonstration. Étant donné un automate fini $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$, on construit $\mathcal{A}' = \langle \Sigma, 2^Q, I, \{E \subseteq Q \mid E \cap F \neq \emptyset\}, \delta' \rangle$ avec

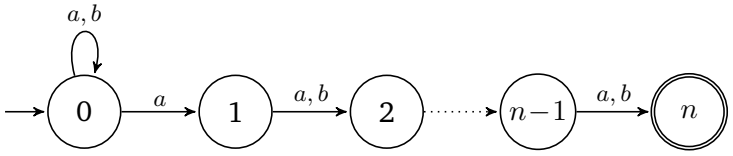
$$\delta' = \{(E, a, E') \mid E' = \delta^*(E, a)\}$$

□

Cette construction par la méthode des sous-ensembles (ou des parties) a un coût de $O(2^{|Q|})$ dans le pire des cas, et il existe des automates à n états dont le déterminisé complet a 2^n états.

Exemple 1.14. Un exemple classique d'explosion lors de la déterminisation est l'automate suivant, avec un passage de $n+1$ à 2^n+1 états.

C.f. [Car08, sec. 1.6], [Sak03, sec. I.3.2], [Aut94, p. 45], [SSS88, sec. 3.4], [HU79, théorème 3.2]. Voir en particulier [Sak03, exercice I.3.6] pour plusieurs exemples d'explosion lors de la déterminisation, dont un exemple qui atteint la borne.



Démonstration. On montre par récurrence sur $k = |P|$ pour toute partie P de $\{1, \ldots, n\}$ que l'état $\{0\} \cup P$ est accessible dans le déterminisé. La propriété est vraie pour $k = 0$ puisque $\{0\}$ est l'état initial du déterminisé.

Soit P un sous-ensemble de $\{1, \ldots, n\}$ de cardinal $k+1$. On définit l'ensemble

$$P-1 = \{i - 1 \mid i \neq 1 \in P\}$$

Si P contient 1, alors soit $P-1$ est de cardinal k et $\{0\} \cup P-1$ est accessible par hypothèse de récurrence ; on a bien $\delta'(\{0\} \cup P-1, a) = \{0\} \cup P$ donc P est accessible.

Si P ne contient pas 1, alors on procède par récurrence sur j la plus petite valeur dans P. Pour $j = 2$, $P-1$ est de cardinal $k+1$ et contient 1, donc est accessible au vu du cas précédent, et $\delta'(\{0\} \cup P-1, b) = \{0\} \cup P$. Puis, pour une plus petite valeur $j+1$ de P, $P-1$ a pour plus petite valeur j donc est accessible par hypothèse de récurrence, et $\delta'(\{0\} \cup P-1, b) = \{0\} \cup P$. □

Malgré son coût potentiellement explosif, la déterminisation sert avant tout à améliorer la complexité des problèmes sur les automates ! Pour peu que l'automate déterministe reste de taille raisonnable, on bénéficie alors de complexités polynomiales pour les problèmes d'équivalence et d'inclusion de langages, et linéaire pour le problème d'appartenance.

C.f. [SSS88, pp. 92–95].

Théorème 1.15 (Appartenance)**.** *Soit w un mot de Σ^* et $\mathcal{A}$ un automate fini sur Σ.*

1. *L'appartenance de w à $L(\mathcal{A})$ est décidable en temps déterministe $O(|\mathcal{A}| \cdot |w|)$ et en espace $O(|\mathcal{A}|)$.*
2. *Si $\mathcal{A}$ est déterministe, alors l'appartenance de w à $L(\mathcal{A})$ est décidable en temps déterministe $O(|\mathcal{A}| + |w|)$ et en espace $O(|\mathcal{A}|)$.*

Des complexités polynomiales pour les problèmes d'inclusion et d'équivalence peuvent aussi être obtenues en considérant des automates non ambigus *tout en diminuant les risques d'explosion [25]. L'automate non déterministe de l'exemple 1.14 était justement non ambigu.*

Démonstration. Soit $w = a_1 \cdots a_n$ et $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$.

On démontre le point 1 à l'aide d'un algorithme qui calcule initialement les états accessibles par $S_0 := \delta^*(I, \varepsilon)$ en temps $O(|\mathcal{A}|)$ puis les ensembles $S_i := \delta^*(S_{i-1}, a_i)$ en temps $O(|\mathcal{A}|)$ pour i de 1 à n. Le mot w appartient à $L(\mathcal{A})$ si et seulement si $S_n \cap F \neq \emptyset$. On n'a besoin de conserver qu'un seul ensemble S_i à la fois dans ce calcul, donc on reste bien en espace $O(|\mathcal{A}|)$.

Pour le cas du point 2, on observe que l'algorithme précédent voit ses ensembles S_i réduits à des singletons calculables en temps $O(1)$. La complexité doit de plus tenir compte du temps et de l'espace pris par le chargement de $\mathcal{A}$. □

1.1.3 Propriétés de clôture

C.f. [Car08, sec. 1.8], [Aut94, pp. 43–48], [HU79, sec. 3.2], [Har78, sec. 2.3].

Les langages reconnaissables sont clos par union, concaténation et étoile de KLEENE comme nous le verrons dans le cadre de la construction de THOMPSON en section 1.2.1. Cette section est plutôt l'occasion de parler d'une autre propriété de clôture, par *relation rationnelle* ou *transduction rationnelle*, qui généralise plusieurs autres propriétés de clôture.

C.f. [Sak03, sec. IV.1], [4, ch. III]. L'article fondateur en la matière est dû à ELGOT *et* MEZEI *[13].*

Sa démonstration est souvent donnée en décomposant la transduction en morphisme, morphisme inverse et intersection et en utilisant les preuves de clôture par ces opérations. Nous prenons ici le cheminement inverse et nous déduirons les clôtures par morphisme, morphisme inverse et intersection de celle par transduction.

Le principal avantage de cette introduction des transducteurs dans une leçon est qu'ils donnent lieu à de nombreuses applications : au lieu de simplement définir un langage, un transducteur définit une relation ou une fonction, et permet donc de calculer effectivement.

Définition 1.16 (Transducteur)**.** Un *transducteur* sur $\Sigma \times \Delta$ est un automate fini sur $\Sigma^* \times \Delta^*$. Un transducteur est sous *forme normale* si sa relation de transition est incluse dans $Q \times (\Sigma \times \{\varepsilon\} \cup \{\varepsilon\} \times \Delta) \times Q$. La *relation rationnelle* sur $\Sigma^* \times \Delta^*$ définie par $\mathcal{T}$ est

$$R(\mathcal{T}) = \{(u, v) \in \Sigma^* \times \Delta^* \mid \delta^*(I, (u, v)) \cap F \neq \emptyset\}$$

C.f. [Sak03, corollaire IV.1.3], [HU79, théorème 11.1], [Har78, théorème 6.4.3].

Proposition 1.17 (Clôture par transduction)**.** *L'ensemble des langages reconnaissables est clôt par transduction rationnelle.*

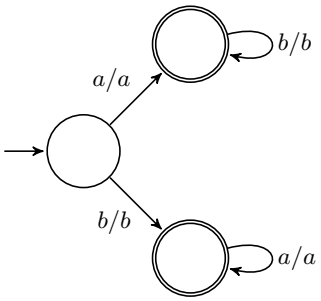
Démonstration. Soient $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ un automate fini sur Σ et $\mathcal{T} = \langle \Sigma \times \Delta, Q', I', F', \delta' \rangle$ un transducteur sur $\Sigma \times \Delta$. On construit un automate fini $\mathcal{A}' = \langle \Delta, Q \times Q', I \times I', F \times F', \delta'' \rangle$ sur Δ tel que $R(\mathcal{T})(L(\mathcal{A})) = L(\mathcal{A}')$. Sans perte de généralité, on suppose $\mathcal{A}$ sans ε-transition et $\mathcal{T}$ sous forme normale. On définit alors avec a dans $\Delta \cup \{\varepsilon\}$ et b dans Δ :

$$\begin{aligned}\delta'' = &\{((q_1, q_2), a, (q_1', q_2')) \mid \exists c \in \Sigma, (q_1, c, q_1') \in \delta \wedge (q_2, (c, a), q_2') \in \delta'\} \\ &\cup \{((q_1, q_2), b, (q_1, q_2')) \mid (q_2, (\varepsilon, b), q_2') \in \delta'\}\end{aligned}$$

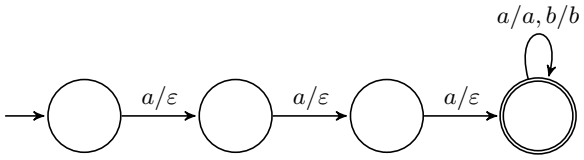
□

Cette construction est essentiellement un raffinement du calcul de l'intersection de deux automates à états finis. La complexité de ce produit synchrone est en $O(|\mathcal{A}| \cdot |\mathcal{A}'|)$ dans le pire des cas. Grâce à cette construction similaire à celle de l'intersection de deux automates finis, on obtient les propriétés de fermeture suivantes :

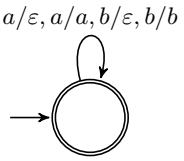
intersection de deux langages reconnaissables L_1 et L_2 sur Σ : en appliquant le transducteur $L_2 \times L_2$ à L_1. Par exemple, le transducteur suivant calcule le résultat de l'intersection avec le langage sur $\{a, b\}^*$ $\{ab^n \mid n \geq 0\} \cup \{ba^m \mid m \geq 0\}$:



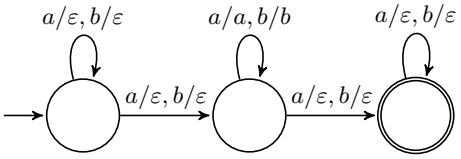
quotient gauche de L par un mot u de Σ^* : le transducteur reconnaît (u, ε) puis l'identité sur Σ^*. Par exemple, pour le quotient par $u = aaa$ sur l'alphabet $\{a, b\}$:



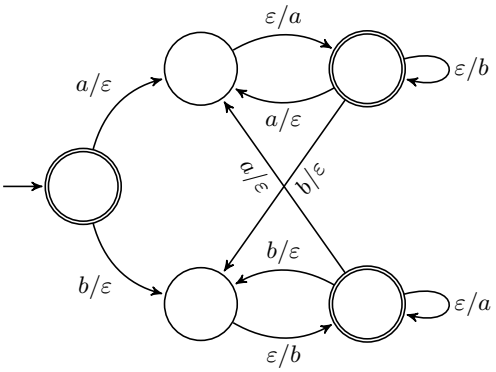
sous-mot : le transducteur choisit non déterministiquement entre recopier (identité) et effacer ses symboles d'entrée ($\Sigma^* \times \varepsilon$). Par exemple, sur $\{a, b\}$:



facteur (et similairement préfixe ou suffixe) : le transducteur est initialement dans un état d'effacement des symboles, choisit non déterministiquement de passer en mode copie, puis non déterministiquement de passer en mode d'effacement jusqu'à la fin. Par exemple, sur $\{a, b\}$:



substitution rationnelle σ de Σ^* dans $\mathrm{Rec}(\Delta^*)$: par le transducteur défini par $(\bigcup_{a\in\Sigma}\{a\} \times \sigma(a))^*$. Cela implique les clôtures par substitution inverse, morphisme et morphisme inverse. Par exemple, pour la substitution σ définie par $a \mapsto \{ab^n \mid n \geq 0\}$ et $b \mapsto \{ba^m \mid m \geq 0\}$:



On pourrait aussi exprimer les opérations d'union et de concaténation de langages reconnaissables, mais les constructions sont des adaptations de celles pour les automates finis et n'exploitent pas réellement les transducteurs.

Il nous reste à mentionner une opération de clôture importante : celle par complément.

Proposition 1.18 (Clôture par complément)**.** *L'ensemble des langages reconnaissables est clôt par complément.*

Démonstration. Soit $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ un automate déterministe complet. L'automate $\langle \Sigma, Q, I, Q \backslash F, \delta \rangle$ reconnaît $\overline{L(\mathcal{A})}$, le complémentaire de $L(\mathcal{A})$. □

C.f. [Sak03, p. 118].

On en déduit immédiatement la décidabilité de l'inclusion et de l'équivalence de deux langages reconnaissables :

Théorème 1.19 (Inclusion)**.** *Soient L_1 et L_2 deux langages reconnaissables. Décider si $L_1 \subseteq L_2$ est* PSPACE*-complet.*

C.f. [SSS88, théorème 3.46].

Démonstration. En effet, les calculs du complément et de l'intersection de deux langages reconnaissables sont effectifs, et

$$L_1 \subseteq L_2 \text{ ssi } L_1 \cap \overline{L_2} = \emptyset \ .$$

En composant « au vol » ces calculs avec un calcul du vide (qui est en espace logarithmique), cela fournit bien un algorithme en espace polynomial. La complétude pour PSPACE vient du problème de l'universalité : $\Sigma^* \subseteq L$ est déjà PSPACE-complet. □

On verra en section 1.3 une autre technique pour vérifier l'équivalence de deux automates déterministes : calculer l'automate minimal (qui est canonique) pour chacun et vérifier s'ils sont isomorphes.

La construction du complément d'un langage reconnu par un automate avec n états passe ici par le calcul d'un automate déterministe, avec une complexité en $O(2^n)$. On pourrait espérer trouver des automates non déterministes plus compacts pour le complémentaire d'un langage, mais cette borne supérieure est atteinte, comme le montre l'exemple suivant :

Exemple 1.20. On considère l'automate $\mathcal{A}_n = \langle \{a, b\}, Q, Q, Q, \delta \rangle$ avec l'ensemble d'états $Q = \{0, \ldots, n-1\}$ et les transitions $\delta(i, a) = i + 1 \bmod n$ pour tout i de Q, et $\delta(i, b) = i$ si $i \neq 0$. Tout automate reconnaissant $\overline{L(\mathcal{A}_n)}$ possède au moins $2^n - 2$ états.

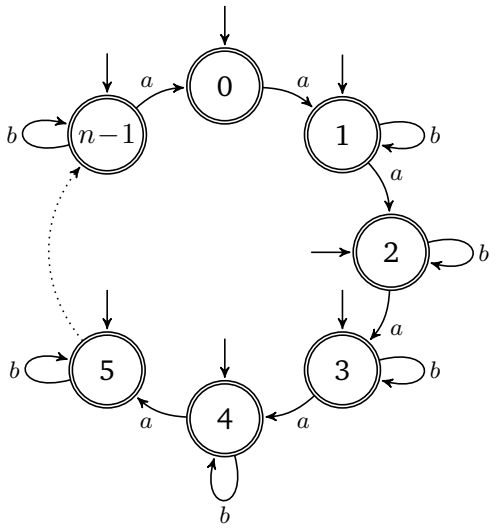
Exemple adapté de YAN *[29].*

Démonstration. On peut déjà noter que la déterminisation de cet automate produit un automate avec 2^n états. Mais on cherche ici à montrer que même un automate *non déterministe* pour le langage $\overline{L(\mathcal{A}_n)}$ doit être de taille exponentielle.

Soit K un sous-ensemble strict non vide de Q ; il existe $2^n - 2$ ensembles K différents. On définit $w_K = x_0 \cdots x_{n-1}$ avec pour tout i de Q $x_i = a^i b a^{n-i}$ si $i \in K$ et $x_i = a^n$ sinon.

On observe que le mot $w_K w_{Q \backslash K}$ n'est pas un mot de $L(\mathcal{A}_n)$: pour chaque i de Q, un facteur $a^i b a^{n-i}$ apparaît dans $w_K w_{Q \backslash K}$, et donc quel que soit le choix de l'état initial pour reconnaître le mot, on devra tenter de lire un b dans l'état 0.

En revanche, si $K \neq K'$, sans perte de généralité on peut supposer $K' \not\subseteq K$ et alors le mot $w_K w_{Q \backslash K'}$ est dans $L(\mathcal{A}_n)$: il existe alors un état i tel que $i \notin K$ et



$i \notin Q \backslash K'$, et il suffit de faire démarrer la reconnaissance de $w_K w_{Q\backslash K'}$ dans l'état $n - i$.

Dès lors, pour chacun des $2^n - 2$ ensembles K, les ensembles d'états atteints en lisant w_K dans un automate pour $\overline{L(\mathcal{A}_n)}$ doivent être disjoints, sous peine de permettre la reconnaissance d'un mot $w_K w_{Q\backslash K'}$ dans $L(\mathcal{A}_n)$. □

1.1.4 Lemmes d'itération

C.f. [Car08, sec. 1.9], [Sak03, pp. 77–80], [Aut94, p. 54], [HU79, sec. 3.1], [Har78, théorème 2.2.2].

Les lemmes d'itération et les propriétés de clôture permettent de démontrer que certains langages ne sont pas reconnaissables. Il est en général plus simple de procéder par clôtures successives jusqu'à un langage manifestement non reconnaissable, comme $\{a^n b^n \mid n \geq 0\}$, sur lequel il est aisé d'appliquer un lemme d'itération.

Lemme 1.21 (Lemme de l'étoile)**.** *Si L est un langage reconnaissable de Σ^*, alors il existe un entier n tel que*

1. *pour tout mot w de L avec $|w| \geq n$, il existe une factorisation $w = u_1 u_2 u_3$ avec u_2 non vide, telle que $u_1 u_2^* u_3 \subseteq L$,*
2. *pour tout mot w de L et pour toute factorisation de $w = v_1 v_2 v_3$ avec $|v_2| \geq n$, il existe une factorisation $v_2 = u_1 u_2 u_3$ avec u_2 non vide, telle que $v_1 u_1 u_2^* u_3 v_3 \subseteq L$,*
3. *pour tout mot w de L et pour toute factorisation $w = u_0 u_1 \cdots u_n u_{n+1}$ avec u_i non vide pour i de 1 à n, il existe deux positions $0 \leq j < k \leq n$ telles que $u_0 u_1 \cdots u_j (u_{j+1} \cdots u_k)^* u_{k+1} \cdots u_n u_{n+1} \subseteq L$.*

Même dans sa troisième version, le lemme de l'étoile n'est qu'une condition nécessaire *pour qu'un langage soit rationnel. Il est possible de renforcer les deux dernières version du lemme pour obtenir des caractérisations des langages rationnels. Voir l'article d'*EHRENFEUCHT *et al. [12], [Car08, théorème 1.106] et [Sak03, sec. I.3.4].*

Démonstration. On démontre le troisième cas, les deux autres en sont déductibles. Soit L un langage reconnaissable, et $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ un automate fini pour L. On pose $n = |Q|$ le nombre d'états de $\mathcal{A}$, et pour tout mot de L admettant une factorisation $w = u_0 u_1 \cdots u_n u_{n+1}$ avec u_i non vide pour i de 1 à n, on appelle q_i pour i de 0 à n l'état atteint après avoir lu $u_0 u_1 \cdots u_i$ lors de la reconnaissance de w. Parmi ces $n + 1$ états q_i, il en existe au moins deux, q_j et q_k avec $0 \leq j < k \leq n$ tels que $q_j = q_k$. Mais alors $u_0 u_1 \cdots u_j (u_{j+1} \cdots u_k)^* u_{k+1} \cdots u_n u_{n+1} \subseteq L$. □

Les exemples et les exercices sur le lemme d'itération abondent dans les livres sur les langages formels. Voici deux exemples plus originaux.

Exemple 1.22. Soit L un langage *quelconque* sur Σ^*, et $\#$ un symbole qui n'est pas dans Σ. Alors le langage

$$L_\# = (\#^+ L) \cup \Sigma^*$$

satisfait les conditions de la première version du lemme de l'étoile.

En effet, on pose $n = 1$ et pour tout mot w de $\#^+ L$ on peut choisir une factorisation avec $u_1 = \varepsilon$ et $u_2 = \#$, et pour tout i $u_1 u_2^i u_3 = \#^i u_3$ sera bien dans $L_\#$. Pour tout mot de Σ^*, le lemme est bien vérifié puisque Σ^* est reconnaissable.

Pourtant $L = \mu(L_\# \cap \#^+\Sigma^*)$ pour le morphisme μ défini par $a \mapsto a$ pour tout a de Σ et par $\# \mapsto \varepsilon$, et donc si L n'est pas reconnaissable, alors $L_\#$ ne l'est pas non plus.

Enfin, si L ne satisfait pas les conditions de la seconde version du lemme de l'étoile, alors $L_\#$ ne les satisfait pas non plus : il suffit d'utiliser le même facteur que pour L.

Les exemples 1.22 et 1.23 sont adaptés de Yu *[30].*

Exemple 1.23. Soit L un langage *quelconque* sur Σ^* et $\$$ un symbole qui n'est pas dans Σ. On définit

$$L_1 = \{\$^+ a_1 \$^+ a_2 \$^+ \cdots \$^+ a_n \$^+ \mid n \geq 0, \forall i\ a_i \in \Sigma, \text{ et } w = a_1 \cdots a_n \in L\}$$
$$L_2 = \{\$^+ v_1 \$^+ v_2 \$^+ \cdots \$^+ v_n \$^+ \mid n \geq 0, \forall i\ v_i \in \Sigma^*, \text{ et } \exists j\ |v_j| \geq 2\}$$
$$L_\$ = L_1 \cup L_2$$

Le langage $L_\$$ satisfait les conditions de la deuxième version du lemme de l'étoile.

En effet, on pose $n = 3$ et on considère un mot xyz de $L_\$$ avec $|y| \geq 3$. Si $\$$ apparaît dans y, alors on peut choisir une décomposition $y = u_1 \$ u_3$ qui vérifie bien le lemme. Si $\$$ n'apparaît pas dans y, alors xyz est dans L_2, et on peut choisir un symbole a de Σ comme facteur itérant : si on l'efface, on aura encore au moins deux symboles consécutifs de Σ et on restera donc dans L_2, et si on l'itère on reste clairement dans L_2.

Pourtant, comme dans l'exemple 1.22, on vérifie que $L = \mu(L_\$ \cap (\$^+\Sigma)^*\$^+)$ pour le morphisme μ défini par $a \mapsto a$ pour tout a de Σ et par $\$ \mapsto \varepsilon$, et donc si L n'est pas reconnaissable, alors $L_\$$ ne l'est pas non plus.

Enfin, si L ne vérifie pas les conditions de la troisième version du lemme de l'étoile, alors $L_\$$ ne les vérifie pas non plus. Si un mot $a_0 \cdots a_m$ a servi à démontrer que L n'est pas reconnaissable en utilisant des positions sur les lettres a_{i_0} à a_{i_n}, alors on considère le mot de L_1 de la forme $\$a_0\$ \cdots \$a_m\$$, et les positions correspondantes modulo l'ajout des symboles $\$$. On utilise alors un mot $w_l = a_0 \cdots (a_{i_{j+1}} \cdots a_{i_k})^l \cdots a_m$ qui n'est pas dans L : l'itération du facteur $\$a_{i_{j+1}}\$ \cdots \$a_{i_k}$ correspondant préserve la propriété qu'il n'y a qu'une seule lettre entre deux symboles $\$$, et donc le mot obtenu n'est pas dans L_2, et il n'est pas non plus dans L_1 puisque w_l n'est pas dans L.

1.2 Langages rationnels

C.f. [Car08, sec. 1.5.1], [Sak03, sec. I.4], [Aut94, ch. 5], [SSS88, sec. 3.1], [HU79, sec. 2.5].

Définition 1.24. (Langage rationnel) $\mathrm{Rat}(\Sigma^*)$ est le plus petit ensemble de parties de Σ^* qui contient les parties finies de Σ^* et qui est fermé par union, concaténation et étoile. Un *langage rationnel* sur Σ^* est un élément de $\mathrm{Rat}(\Sigma^*)$.

Comme chaque partie finie de Σ^* est soit l'ensemble vide, soit une union finie de mots de Σ^*, et que ces mots sont soit le mot vide, soit la concaténation finie de lettres de Σ, on obtient une définition équivalente sous la forme d'expressions rationnelles.

Définition 1.25. (Expression rationnelle) Une *expression rationnelle* E sur Σ est un terme défini inductivement par

$$E ::= \emptyset \mid \varepsilon \mid a \mid E_1^* \mid E_1 + E_2 \mid E_1 E_2$$

où a est un symbole de Σ, et E_1 et E_2 sont des expressions rationnelles.

On interprète une expression rationnelle E comme un langage $L(E)$ sur Σ^* défini inductivement par

$$\begin{aligned} L(\emptyset) &= \emptyset \\ L(\varepsilon) &= \{\varepsilon\} \\ L(a) &= \{a\} \\ L(E^*) &= L(E)^* \\ L(E_1 + E_2) &= L(E_1) \cup L(E_2) \\ L(E_1 E_2) &= L(E_1) L(E_2) \end{aligned}$$

Le langage d'une expression rationnelle est bien un langage rationnel.

Deux mesures sont couramment utilisées pour la *taille* d'une expression rationnelle E : $|E|$ la taille du terme E, définie par

$$\begin{aligned} |\emptyset| = |\varepsilon| = |a| &= 1 \\ |E^*| &= |E| + 1 \\ |E_1 + E_2| = |E_1 E_2| &= |E_1| + |E_2| + 1 \end{aligned}$$

L'autre mesure est le nombre d'occurences de symboles alphabétiques dans E, défini par

$$\begin{aligned} \|\emptyset\| = \|\varepsilon\| &= 0 \\ \|a\| &= 1 \\ \|E^*\| &= \|E\| \\ \|E_1 + E_2\| = \|E_1 E_2\| &= \|E_1\| + \|E_2\| \end{aligned}$$

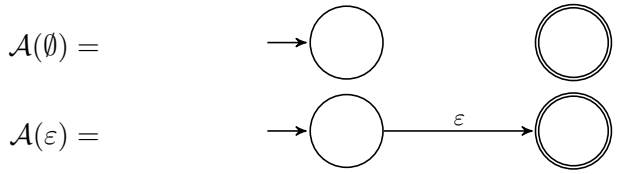
Comme annoncé au début du chapitre 1, le résultat central sur les langages rationnels est qu'ils coïncident avec les langages reconnaissables, sur le monoïde libre Σ^*.

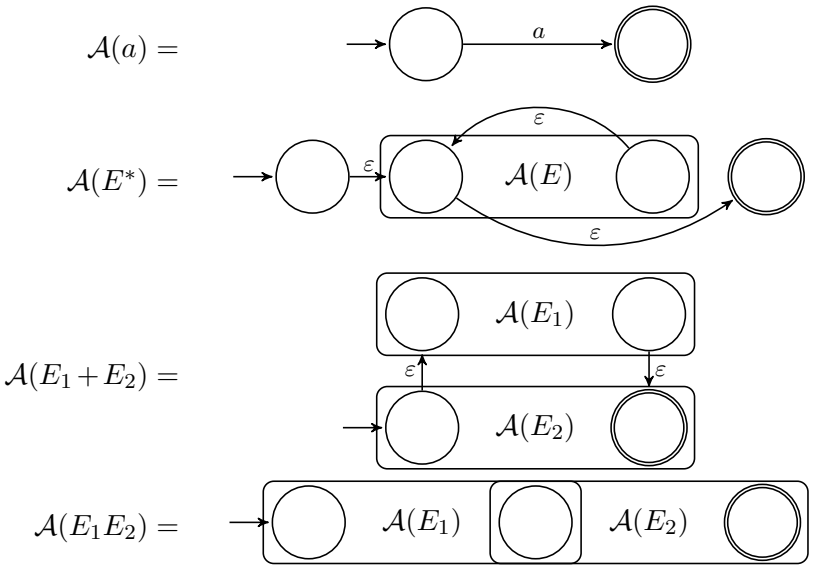
1.2.1 Des expressions aux automates

C.f. THOMPSON *[28] et [Sak03, pp. 157–158], [SSS88, théorème 3.16], [HU79, théorème 2.3].*

Construction de THOMPSON

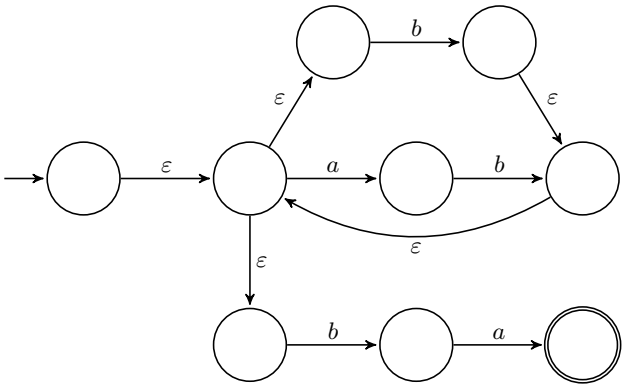
La construction la plus simple et la plus intuitive : on construit un automate $\mathcal{A}(E)$ par induction sur l'expression rationnelle E. Les automates générés vérifient comme invariant qu'ils ont un unique état initial, qui n'a pas de transition entrante, et un unique état final, qui n'a pas de transition sortante. Cet invariant permet d'assurer la validité de la construction pour la concaténation et l'étoile.





Cette construction inductive travaille en $O(|E|)$ sur la taille de l'expression, et résulte en un automate avec ε-transitions avec au plus $2|E|$ états et $3|E|$ transitions, soit au final $O(|E|^2)$ une fois les ε-transitions éliminées.

Exemple 1.26. Par exemple, si on considère l'expression $E = (ab + b)^*ba$, on obtient par cette construction l'automate



Automate standard de GLUSHKOV

C.f. GLUSHKOV *[15] et* MCNAUGHTON *et* YAMADA *[20], et aussi* BERSTEL *et* PIN *[5].*

Cette construction d'un automate équivalent à une expression rationnelle E est appréciable, d'une part parce qu'elle résulte en des automates de petite taille, et d'autre part parce qu'elle reflète fidèlement les propriétés de l'expression rationnelle de départ. Ainsi, la notion d'*ambiguïté* d'une expression rationnelle est définie comme l'ambiguïté de son automate de GLUSHKOV [14]. En effet, cette construction résulte en un automate ayant un état pour chaque occurence d'un symbole de Σ dans l'expression E, et distingue ces différentes occurrences – ce qui revient à *linéariser* E.

Définition 1.27 (Expression linéaire)**.** Une expression rationnelle E sur Σ est *linéaire* si chaque symbole de Σ apparaît au plus une fois dans E.

Proposition 1.28. *Tout langage rationnel sur Σ est le résultat de l'application d'un morphisme alphabétique $\Delta \to \Sigma$ au langage d'une expression rationnelle linéaire sur Δ.*

Démonstration. Soit $L(E)$ un langage rationnel sur Σ décrit par une expression rationnelle E sur Σ. On indice chaque apparition d'une lettre a de Σ par sa position dans la lecture linéraire de l'expression, par exemple $(a+b)^*a$ devient $(a_1+b_2)^*a_3$. L'expression E' obtenue est bien linéaire sur Δ l'alphabet indicé, et l'image de $L(E')$ par le morphisme alphabétique $a_i \mapsto a$ est bien $L(E)$. □

La présentation suit fidèlement BERSTEL *et* PIN *[5] ; voir aussi [Sak03, exercice II.1.8].*

Il nous faut faire un petit détour par les automates et langages locaux.

Définition 1.29 (Langage local)**.** Un langage L sur Σ est *local* s'il existe trois ensembles $P \subseteq \Sigma$, $S \subseteq \Sigma$ et $N \subseteq \Sigma^2$ tels que

$$L\backslash\{\varepsilon\} = (P\Sigma^* \cap \Sigma^* S)\backslash(\Sigma^* N \Sigma^*)$$

c.-à-d. tels que tout mot de L commence par un symbole dans P, finisse par un symbole dans S, et n'ait aucun facteur de longueur deux dans N. On peut définir ces ensembles P, S et N par

$$\begin{aligned} P &= \{a \in \Sigma \mid a\Sigma^* \cap L \neq \emptyset\} \\ S &= \{a \in \Sigma \mid \Sigma^* a \cap L \neq \emptyset\} \\ N &= \{ab \in \Sigma^2 \mid \Sigma^* ab \Sigma^* \cap L = \emptyset\}. \end{aligned}$$

Définition 1.30 (Automate local)**.** Un automate fini déterministe est *local* si, pour chaque symbole a de Σ, $|\{q' \in Q \mid \exists q \in Q, (q, a, q') \in \delta\}| \leq 1$. Il est de plus *standard* si son état initial n'a pas de transition entrante.

Lemme 1.31. *Les trois énoncés suivants sont équivalents :*

1. *L est un langage local,*
2. *L est reconnu par un automate local standard,*
3. *L est reconnu par un automate local.*

Démonstration.
1⇒2. On construit l'automate $\mathcal{A} = \langle \Sigma, \Sigma \uplus \{i\}, \{i\}, F, \delta \rangle$ avec pour relation de transition

$$\delta = \{(i, a, a) \mid a \in P\} \cup \{(b, a, a) \mid ba \notin N\}$$

et S pour ensemble final, plus potentiellement l'état i si ε appartient à L. On vérifie aisément que $\mathcal{A}$ est déterministe, local et standard. Il reconnaît de plus L : soit une exécution de $\mathcal{A}$

$$i \xrightarrow{a_1} a_1 \xrightarrow{a_2} a_2 \cdots \xrightarrow{a_n} a_n$$

avec $n > 0$ et $a_n \in F$. Alors $w = a_1 \cdots a_n$ vérifie $a_1 \in P$, $a_n \in S$ et pour tout i de 1 à $n-1$, $a_i a_{i+1} \notin N$, donc $w \in L$. Inversement, si un mot $w \neq \varepsilon$ vérifie ces trois conditions, alors il existe bien cette même exécution dans $\mathcal{A}$, et donc $w \in L(\mathcal{A})$.
2⇒3. Évident.
3⇒1. À partir d'un automate déterministe local $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$, on calcule les ensembles

$$\begin{aligned} P(\mathcal{A}) &= \{a \in \Sigma \mid \delta(q_0, a) \neq \emptyset\} \\ S(\mathcal{A}) &= \{a \in \Sigma \mid \exists q \in Q, \delta(q, a) \in F\} \\ N(\mathcal{A}) &= \Sigma^2 \backslash \{ab \in \Sigma^2 \mid \exists q \in Q, \delta^*(q, ab) \neq \emptyset\} \\ L &= (P(\mathcal{A})\Sigma^* \cap \Sigma^* S(\mathcal{A}))\backslash(\Sigma^* N(\mathcal{A}) \Sigma^*) \end{aligned}$$

On peut noter que L est un langage local. Soit maintenant une exécution dans $\mathcal{A}$

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_n} q_n$$

avec $n > 0$ et $q_n \in F$. Alors $a_1 \in P(\mathcal{A})$, $a_n \in S(\mathcal{A})$ et pour tout i de 1 à $n-1$, $a_i a_{i+1} \notin N(\mathcal{A})$, donc $w = a_1 \cdots a_n$ appartient à L.

Inversement, soit $w = a_1 \cdots a_n$ dans L. On montre inductivement pour chaque i de 1 à n qu'il existe une exécution

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_i} q_i$$

dans $\mathcal{A}$. Puisque $a_1 \in P(\mathcal{A})$, la transition $\delta(q_0, a_1) = q_1$ est bien définie. Puis, puisque $a_i a_{i+1} \notin N$, il existe q et q' tels que $\delta^*(q, a_i a_{i+1}) = q'$. Comme $\mathcal{A}$ est local, $\delta(q, a_i) = \delta(q_{i-1}, a_i) = q_i$, et par suite $q' = \delta(q_i, a_{i+1})$ peut être pris en guise de q_{i+1}. Enfin, $a_n \in S$ implique qu'il existe q dans Q et q_f dans F tels que $\delta(q, a_n) = q_f$, et encore une fois, comme $\mathcal{A}$ est local, $\delta(q, a_n) = \delta(q_{n-1}, a_n)$, donc l'exécution dans $\mathcal{A}$ se finit dans $q_n = q_f$. □

On s'intéresse aussi aux propriétés de clôture des langages locaux.

Lemme 1.32. *Soient L_1 et L_2 deux langages locaux sur des alphabets* disjoints *Σ_1 et Σ_2. Alors $L_1 \cup L_2$, $L_1 \cdot L_2$ et L_1^* sont des langages locaux.*

Démonstration. Par le lemme 1.31, il suffit de prendre des automates locaux standards $\mathcal{A}_1 = \langle \Sigma_1, \Sigma_1 \uplus \{i_1\}, i_1, F_1, \delta_1 \rangle$ et $\mathcal{A}_2 = \langle \Sigma_2, \Sigma_2 \uplus \{i_2\}, i_2, F_2, \delta_2 \rangle$ pour L_1 et L_2 et de construire un automate local $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ pour le langage désiré.

Pour l'union, on pose $\Sigma = \Sigma_1 \uplus \Sigma_2$, $Q = \Sigma_1 \uplus \Sigma_2 \uplus \{q_0\}$ et

$$F = \begin{cases} F_1 \uplus F_2 \uplus \{q_0\} & \text{si } i_1 \in F_1 \text{ ou } i_2 \in F_2 \\ F_1 \uplus F_2 & \text{sinon} \end{cases}$$
$$\delta = \{(q_0, a, a) \mid (i_1, a, a) \in \delta_1 \text{ ou } (i_2, a, a) \in \delta_2\} \cup ((\delta_1 \cup \delta_2) \cap \Sigma^3) \,.$$

Pour la concaténation, on pose $\Sigma = \Sigma_1 \uplus \Sigma_2$, $Q = \Sigma_1 \uplus \Sigma_2 \uplus \{i_1\}$, $q_0 = i_1$ et

$$F = \begin{cases} F_1 \cup F_2 & \text{si } i_2 \in F_2 \\ F_2 & \text{sinon} \end{cases}$$
$$\delta = \delta_1 \cup \{(q, a, a) \mid q \in F_1, (i_2, a, a) \in \delta_2\} \cup (\delta_2 \cap \Sigma^3) \,.$$

Pour l'étoile de KLEENE, on pose $\Sigma = \Sigma_1$, $Q = \Sigma_1 \uplus \{i_1\}$, $q_0 = i_1$, $F = F_1 \cup \{i_1\}$, et $\delta = \delta_1 \cup \{(a, b, b) \mid a \in F_1 \text{ et } (i_1, b, b) \in \delta_1\}$. □

Proposition 1.33. *Pour toute expression linéaire E sur Σ, on peut construire un automate local $\mathcal{A}$ sur Σ tel que $L(E) = L(\mathcal{A})$.*

Démonstration. Comme E est linéaire, par le lemme 1.32, son langage $L(E)$ est local. On calcule inductivement sur E les ensembles de symboles préfixes $P(E)$, de symboles suffixes $S(E)$, et de paires de symboles facteurs $F(E)$, en utilisant

aussi $\lambda(E)$ égal à $\{\varepsilon\}$ si $\varepsilon \in L(E)$ et à $\emptyset$ sinon :

$$\begin{aligned}
\lambda(\emptyset) &= \emptyset & P(\emptyset) &= \emptyset \\
\lambda(\varepsilon) &= \{\varepsilon\} & P(\varepsilon) &= \emptyset \\
\lambda(a) &= \emptyset & P(a) &= \{a\} \\
\lambda(E^*) &= \{\varepsilon\} & P(E^*) &= P(E) \\
\lambda(E_1 + E_2) &= \lambda(E_1) \cup \lambda(E_2) & P(E_1 + E_2) &= P(E_1) \cup P(E_2) \\
\lambda(E_1E_2) &= \lambda(E_1) \cap \lambda(E_2) & P(E_1E_2) &= P(E_1) \cup \lambda(E_1)P(E_2)
\end{aligned}$$

$$\begin{aligned}
S(\emptyset) &= \emptyset & F(\emptyset) &= \emptyset \\
S(\varepsilon) &= \emptyset & F(\varepsilon) &= \emptyset \\
S(a) &= \{a\} & F(a) &= \emptyset \\
S(E^*) &= S(E) & F(E^*) &= F(E) \cup S(E)P(E) \\
S(E_1 + E_2) &= S(E_1) \cup S(E_2) & F(E_1 + E_2) &= F(E_1) \cup F(E_2) \\
S(E_1E_2) &= S(E_2) \cup \lambda(E_2)P(E_1) & F(E_1E_2) &= F(E_1) \cup F(E_2) \cup S(E_1)P(E_2)
\end{aligned}$$

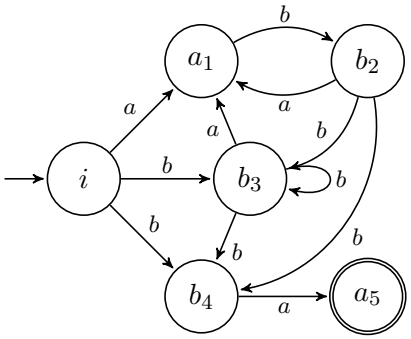
Comme $L(E)$ est local, il vérifie

$$L(E)\backslash\{\varepsilon\} = (P(E)\Sigma^* \cap \Sigma^* S(E))\backslash(\Sigma^*(\Sigma^2\backslash F(E))\Sigma^*)$$

On conclut en utilisant la construction du lemme 1.31 pour obtenir un automate local pour $L(E)$. □

En combinant les propositions 1.28 et 1.33, on obtient une méthode de construction d'un automate fini sans ε-transitions à partir d'une expression rationnelle E, avec *exactement* $\|E\| + 1$ états, où $\|E\|$ est le nombre d'apparitions de symboles de Σ dans E, et au pire $O(\|E\|^2)$ transitions – réellement obtenues sur l'expression correspondant à l'exemple 1.9.

Exemple 1.34. Considérons à nouveau l'expression rationnelle $E = (ab + b)^*ba$. On peut la linéariser en $(a_1b_2 + b_3)^*b_4a_5$, et en déduire l'automate suivant.



*Cette section reprend les preuves d'*ANTIMIROV *[2] ; voir aussi [Sak03, sec. I.5.2], et la construction similaire de* BRZOZOWSKI *[7] dans [Sak03, sec. I.4.4].*

Expressions dérivées partielles d'ANTIMIROV

Cette technique de construction d'un automate équivalent à une expression rationnelle fournit des automates, généralement non déterministes, encore plus compacts que ceux de l'automate de GLUSHKOV. Cette construction est un raffinement de celle mise au point par BRZOZOWSKI [7], et le déterminisé de l'automate que l'on obtient est exactement celui que l'on aurait obtenu par la construction de BRZOZOWSKI.

Définition 1.35 (Dérivée d'une expression)**.** La *dérivée partielle* $\partial_a(E)$ d'une expression rationnelle E sur Σ par une lettre a de Σ est l'ensemble d'expressions rationnelles sur Σ défini par

$$\begin{aligned}
\partial_a(\emptyset) &= \emptyset \\
\partial_a(\varepsilon) &= \emptyset \\
\partial_a(b) &= \begin{cases} \{\varepsilon\} & \text{si } a = b \\ \emptyset & \text{sinon} \end{cases} \\
\partial_a(E+F) &= \partial_a(E) \cup \partial_a(F) \\
\partial_a(E^*) &= \partial_a(E) \cdot E^* \\
\partial_a(EF) &= \begin{cases} \partial_a(E) \cdot F & \text{si } \varepsilon \notin L(E) \\ \partial_a(E) \cdot F \cup \partial_a(F) & \text{sinon} \end{cases}
\end{aligned}$$

où l'opération de concaténation est étendue de manière évidente aux ensembles d'expressions rationnelles. Attention, dans $\partial_a(\emptyset) = \emptyset$, le symbole « $\emptyset$ » du terme de gauche est une expression rationnelle, tandis que celui du terme de droite est l'ensemble vide !

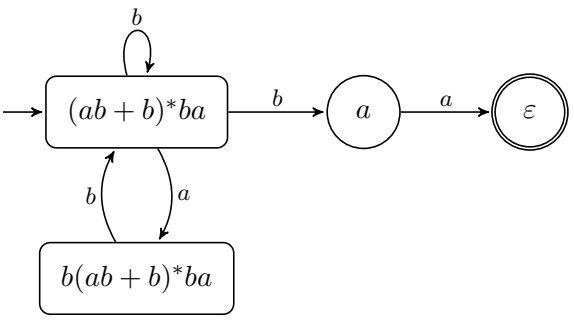
On étend cette définition à des mots w de Σ^* et à des ensembles d'expressions rationnelles S par

$$\begin{aligned}
\partial_\varepsilon(E) &= \{E\} \\
\partial_{wa}(E) &= \partial_a(\partial_w(E)) \\
\partial_w(S) &= \bigcup_{E \in S} \partial_w(E) \ .
\end{aligned}$$

Exemple 1.36. Par exemple, si on pose $E = (ab+b)^*ba$, on obtient les dérivées partielles successives

$$\begin{aligned}
\partial_a(E) &= \{b(ab+b)^*ba\} \\
\partial_b(E) &= \{E, a\} \\
\partial_a(b(ab+b)^*ba) &= \emptyset \\
\partial_b(b(ab+b)^*ba) &= \{E\} \\
\partial_a(a) &= \{\varepsilon\} \\
\partial_b(a) &= \emptyset \\
\partial_a(\varepsilon) &= \emptyset \\
\partial_b(\varepsilon) &= \emptyset \ .
\end{aligned}$$

Cet ensemble de dérivées partielles se traduit aisément en un automate :



L'automate construit à partir des expressions dérivées partielles d'une expression rationnelle E est $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ ayant pour ensemble d'états des expressions rationnelles, et dont les transitions respectent les dérivations :

$$\begin{aligned} Q &= \{E_1 \mid \exists w \in \Sigma^*, E_1 \in \partial_w(E)\} \\ I &= \{E\} \\ F &= \{E_1 \in Q \mid \varepsilon \in L(E_1)\} \\ \delta &= \{(E_1, a, E_2) \in Q \times \Sigma \times Q \mid E_2 \in \partial_a(E_1)\} \ . \end{aligned}$$

Le fait que cet automate reconnaît $L(E)$ découle de la proposition 1.37, et le fait qu'il soit bien un automate *fini* de la proposition 1.39.

On note par $w^{-1}L = \{u \in \Sigma^* \mid wu \in L\}$ le quotient à gauche de $L \subseteq \Sigma^*$ par $w \in \Sigma^*$.

Proposition 1.37. *Soit S un ensemble d'expressions rationnelles sur Σ et w un mot de Σ^*. Alors $L(\partial_w(S)) = w^{-1}L(S)$.*

Démonstration. On opère par récurrence sur la longueur de w. Si $w = \varepsilon$, alors par définition $L(\partial_\varepsilon(S)) = L(S)$. Soit maintenant $w = ua$, a appartenant à Σ ; on vérifie trivialement par induction sur les expressions E de $\partial_u(S)$ que $L(\partial_a(E)) = a^{-1}L(E)$. On a donc

$$L(\partial_{ua}(S)) = L(\partial_a(\partial_u(S))) = \bigcup_{E \in \partial_u(S)} L(\partial_a(E)) = \bigcup_{E \in \partial_u(S)} a^{-1}L(E) = a^{-1}L(\partial_u(S))$$

et, en appliquant l'hypothèse de récurrence sur $L(\partial_u(S))$,

$$L(\partial_{ua}(S)) = a^{-1}u^{-1}L(S) = (ua)^{-1}L(S) \ .$$

□

On note l'ensemble des suffixes propres d'un mot w par

$$\text{Suff}_+(w) = \{v \in \Sigma^+ \mid \exists u \in \Sigma^*, w = uv\} \ .$$

Lemme 1.38. *Pour tout mot w de Σ^+ et expressions E et F sur Σ, on a :*

$$\partial_w(E + F) = \partial_w(E) \cup \partial_w(F) \tag{1.1}$$

$$\partial_w(EF) \subseteq \partial_w(E) \cdot F \cup \bigcup_{v \in \textit{Suff}_+(w)} \partial_v(F) \tag{1.2}$$

$$\partial_w(E^*) \subseteq \bigcup_{v \in \textit{Suff}_+(w)} \partial_v(E) \cdot E^* \tag{1.3}$$

Démonstration. L'équation (1.1) est immédiate par récurrence sur $w = ua$, a appartenant à Σ :

$$\begin{aligned} \partial_{ua}(E + F) &= \partial_a(\partial_u(E + F)) = \partial_a(\partial_u(E) \cup \partial_u(F)) = \partial_a(\partial_u(E)) \cup \partial_a(\partial_u(F)) \\ &= \partial_{ua}(E) \cup \partial_{ua}(F) \ . \end{aligned}$$

L'équation (1.2) est démontrée par recurrence sur $w = ua$, a appartenant à Σ. Si $u = \varepsilon$, alors on a bien

$$\partial_a(EF) \subseteq \partial_a(E) \cdot F \cup \partial_a(F) = \partial_a(E) \cdot F \cup \bigcup_{v \in \text{Suff}_+(a)} \partial_v(F)$$

puisque $\text{Suff}_+(a) = \{a\}$. Puis

$$\begin{aligned}
\partial_{ua}(EF) &= \partial_a(\partial_u(EF)) \\
&\subseteq \partial_a\left(\partial_u(E)\cdot F \cup \bigcup_{v\in\text{Suff}_+(u)} \partial_v(F)\right) \\
&= \partial_a(\partial_u(E)\cdot F) \cup \bigcup_{v\in\text{Suff}_+(u)} \partial_a(\partial_v(F)) \\
&\subseteq \partial_a(\partial_u(E))\cdot F \cup \partial_a(F) \cup \bigcup_{v\in\text{Suff}_+(u)} \partial_{va}(F) \\
&= \partial_{ua}(E)\cdot F \cup \bigcup_{v\in\text{Suff}_+(ua)} \partial_v(F)
\end{aligned}$$

Enfin l'équation (1.3) est elle aussi démontrée par récurrence sur $w = ua$, a appartenant à Σ. Si $u = \varepsilon$, alors

$$\partial_a(E^*) = \partial_a(E)\cdot E^* = \bigcup_{v\in\text{Suff}_+(a)} \partial_v(E)\cdot E^* \ .$$

Puis,

$$\begin{aligned}
\partial_{ua}(E^*) &= \partial_a(\partial_u(E^*)) \\
&\subseteq \partial_a\left(\bigcup_{v\in\text{Suff}_+(u)} \partial_v(E)\cdot E^*\right) \\
&= \bigcup_{v\in\text{Suff}_+(u)} \partial_a(\partial_v(E)\cdot E^*) \\
&\subseteq \bigcup_{v\in\text{Suff}_+(u)} (\partial_a(\partial_v(E))\cdot E^* \cup \partial_a(E^*)) \\
&= \left(\bigcup_{v\in\text{Suff}_+(u)} \partial_{va}(E)\cdot E^*\right) \cup \partial_a(E)\cdot E^* \\
&= \bigcup_{v\in\text{Suff}_+(ua)} \partial_v(E)\cdot E^* \ .
\end{aligned}$$

□

On montre maintenant que l'ensemble des dérivées partielles d'une expression E est fini, et même qu'il est borné par $\|E\|+1$, ce qui rend l'automate d'ANTIMIROV plus compact que celui de GLUSHKOV.

La taille de l'automate demeure cependant en $O(\|E\|^2)$, comme on peut le constater sur l'exemple 1.9.

Proposition 1.39. *L'ensemble des dérivées partielles différentes d'une expression rationnelle E par tous les mots de Σ^* contient au plus $\|E\|+1$ éléments.*

Démonstration. Montrons par induction sur E que $|\bigcup_{w\in\Sigma^+} \partial_w(E)| \leq \|E\|$. Le lemme sera alors démontré puisque

$$|\bigcup_{w\in\Sigma^*} \partial_w(E)| = |\partial_\varepsilon(E) \cup \bigcup_{w\in\Sigma^+} \partial_w(E)| \leq \|E\|+1 \ .$$

Les cas de base, pour $E = \emptyset$, $E = \varepsilon$ et $E = a$ avec a dans Σ sont évidents. On vérifie ensuite grâce aux équations du lemme 1.38

$$
\begin{aligned}
|\bigcup_{w\in\Sigma^+} \partial_w(E+F)| &= |\bigcup_{w\in\Sigma^+} (\partial_w(E)\cup\partial_w(F))| = |\bigcup_{w\in\Sigma^+}\partial_w(E)| + |\bigcup_{w\in\Sigma^+}\partial_w(F)| \\
&\leq \|E\| + \|F\| \\
|\bigcup_{w\in\Sigma^+} \partial_w(EF)| &\leq \left|\bigcup_{w\in\Sigma^+}\left(\partial_w(E)\cdot F \cup \bigcup_{v\in\mathsf{Suff}_+(w)}\partial_v(F)\right)\right| \\
&= \left|\left(\bigcup_{w\in\Sigma^+}\partial_w(E)\right)\cdot F\right| + \left|\bigcup_{w\in\Sigma^+}\bigcup_{v\in\mathsf{Suff}_+(w)}\partial_v(F)\right| \\
&\leq \|E\| + \|F\| \\
|\bigcup_{w\in\Sigma^+} \partial_w(E^*)| &\leq \left|\bigcup_{w\in\Sigma^+}\bigcup_{v\in\mathsf{Suff}_+(w)}\partial_v(E)\cdot E^*\right| \\
&= \left|\bigcup_{w\in\Sigma^+}\partial_w(E)\cdot E^*\right| \\
&\leq \|E\| .
\end{aligned}
$$

□

1.2.2 Des automates aux expressions

Cette direction du théorème de KLEENE est la moins utile en pratique, et la plus coûteuse. Nous allons voir trois techniques pour réaliser cette conversion, qui sont en réalité trois présentations d'une même technique (c.f. [Sak03, sec. 4.3]).

C.f. MCNAUGHTON *et* YAMADA *[20] et [Car08, p. 38], [Sak03, pp. 103–104], [Aut94, p. 53], [3, p. 305], [SSS88, théorème 3.17], [HU79, théorème 2.4], [Har78, sec. 2.4].*

Construction de MCNAUGHTON et YAMADA

La construction la plus simple à démontrer, que l'on trouvera dans beaucoup d'ouvrages. En revanche, elle est délicate à utiliser à la main.

Soit $\mathcal{A} = \langle \Sigma, Q, I, F, \delta\rangle$ un automate fini ; on va chercher à calculer des expressions rationnelles dérivant les langages

$$L_{p,q} = \{w \in \Sigma^* \mid q \in \delta^*(p,w)\}$$

reconnus entre deux états p et q de Q. En effet, le langage de $\mathcal{A}$ est l'union finie

$$L(\mathcal{A}) = \bigcup_{p\in I, q\in F} L_{p,q} .$$

On ordonne les états de Q, alors vu comme l'ensemble $\{1,\dots,n\}$, et on construit incrémentalement des expressions rationnelles pour les ensembles de chemins $L^{(k)}_{p,q}$ de p à q qui ne passent que par des états intermédiaires inférieurs à un certain k. Cette méthode fournit les expressions désirées puisque $L^{(n)}_{p,q} = L_{p,q}$. On pose initialement

$$L^{(0)}_{p,q} = \begin{cases} \sum_{(p,a,q)\in\delta} a + \varepsilon & \text{si } p = q \\ \sum_{(p,a,q)\in\delta} a & \text{sinon} \end{cases}$$

Puis, pour l'étape d'induction sur k de 1 à n, on vérifie

$$L_{p,q}^{(k)} = L_{p,q}^{(k-1)} + L_{p,k}^{(k-1)} \cdot (L_{k,k}^{(k-1)})^* \cdot L_{k,q}^{(k-1)}$$

Construction de BRZOZOWSKI et MCCLUSKEY

C.f. BRZOZOWSKI *et* MCCLUSKEY *[8] et [Car08, p. 39], [Sak03, pp. 105–107].*

Cette construction est plus intuitive, et plus aisée à utiliser à la main. Elle consiste à éliminer un à un les états de l'automate, et de mettre ses transitions à jour en consquence, en utilisant des parties rationnelles de Σ^* comme étiquettes de transitions.

Définition 1.40 (Automate généralisé)**.** Un *automate généralisé* est un automate fini $\mathcal{A} = \langle 2^{\Sigma^*}, Q, I, F, \delta \rangle$, c'est-à-dire un automate dont les transitions sont étiquetées par des langages L sur Σ.

Proposition 1.41. *Soit $\mathcal{A} = \langle \mathit{Rat}(\Sigma^*), Q, I, F, \delta \rangle$ un automate généralisé. Alors $L(\mathcal{A}) \in \mathit{Rat}(\Sigma^*)$.*

Démonstration. La preuve consiste en la construction d'un automate

$$\mathcal{A}'_n = \langle \mathrm{Rat}(\Sigma^*), \{q_0, q_f\}, \{q_0\}, \{q_f\}, \{(q_0, L(\mathcal{A}), q_f)\} \rangle$$

équivalent à $\mathcal{A}$.

On construit tout d'abord un automate $\mathcal{A}'_0$ avec un unique état initial q_0, un unique état final q_f, et une seule transition entre deux états q et q' de $\mathcal{A}$:

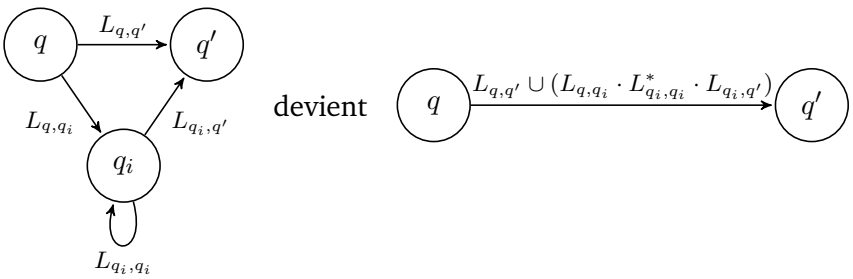
$$\begin{aligned}
\mathcal{A}'_0 &= \langle \mathrm{Rat}(\Sigma^*), Q \uplus \{q_0, q_f\}, \{q_0\}, \{q_f\}, \delta_0 \rangle \\
\delta_0 &= \{(q_0, \{\varepsilon\}, q) \mid q \in I\} \cup \{(q_0, \emptyset, q) \mid q \in (Q \backslash I) \uplus \{q_f\} \\
&\quad \cup \{(q, \{\varepsilon\}, q_f) \mid q \in F\} \cup \{(q, \emptyset, q_f) \mid q \in (Q \backslash F) \uplus \{q_0\}\} \\
&\quad \cup \{(q, \bigcup_{(q,L,q') \in \delta} L, q') \mid q \neq q' \in Q\} \cup \{(q, \{\varepsilon\} \cup \bigcup_{(q,L,q) \in \delta} L, q) \mid q \in Q\} \,.
\end{aligned}$$

Notons qu'une transition existe entre toute paire d'états de $\mathcal{A}'_0$, potentiellement étiquetée par ε ou $\emptyset$. Cet automate est bien équivalent à $\mathcal{A}$, et a bien ses étiquettes dans $\mathrm{Rat}(\Sigma^*)$.

La suite de la construction procède par induction sur $n = |Q|$ pour construire des automates $\mathcal{A}'_i$ où le $i$ème état est éliminé de $\mathcal{A}'_{i-1}$, tels que $\mathcal{A}'_i$ soit encore équivalent à $\mathcal{A}$. Notons q_i l'état de Q éliminé à l'étape i, on a alors

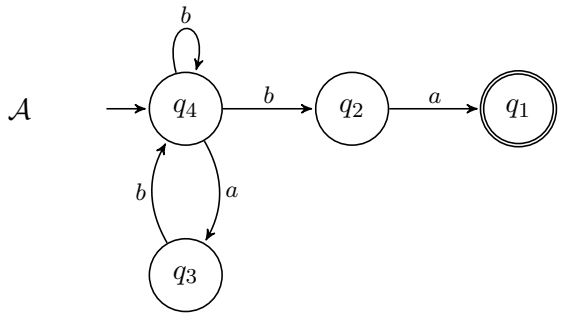
$$\begin{aligned}
\mathcal{A}'_i &= \langle \mathrm{Rat}(\Sigma^*), \{q_{i+1}, \ldots, q_n, q_0, q_f\}, \{q_0\}, \{q_f\}, \delta_i \rangle \\
\delta_i &= \{(q, L_{q,q'} \cup (L_{q,q_i} \cdot L_{q_i,q_i}^* \cdot L_{q_i,q'}), q') \\
&\qquad \mid (q, L_{q,q'}, q'), (q, L_{q,q_i}, q_i), (q_i, L_{q_i,q_i}, q_i), (q_i, L_{q_i,q'}, q') \in \delta_{i-1}\} \,.
\end{aligned}$$

Schématiquement :

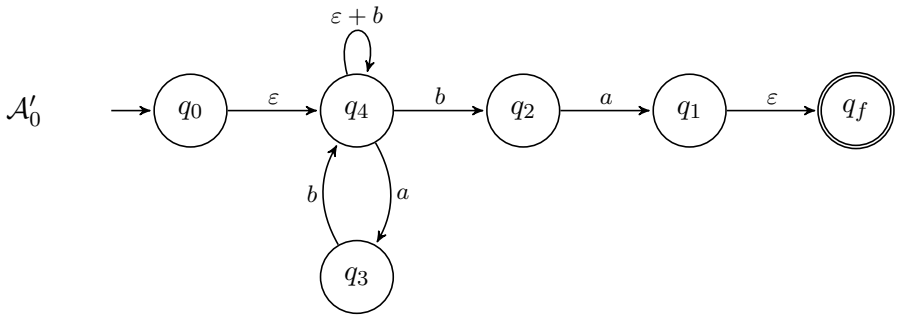


On vérifie bien que $\mathcal{A}'_i$ est équivalent à $\mathcal{A}'_{i-1}$, et donc par hypothèse d'induction, à l'automate $\mathcal{A}$ d'origine. □

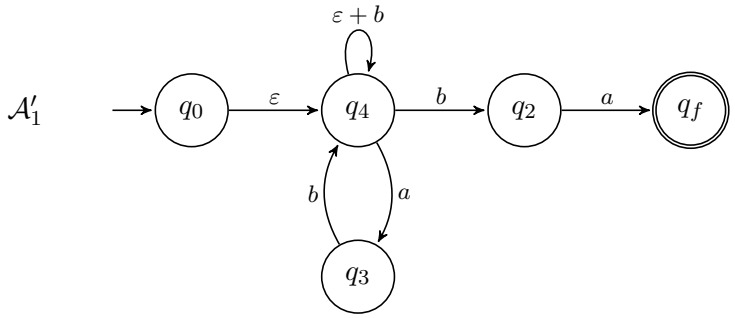
Exemple 1.42. Appliquons cette technique d'élimination à l'automate $\mathcal{A}$ suivant :

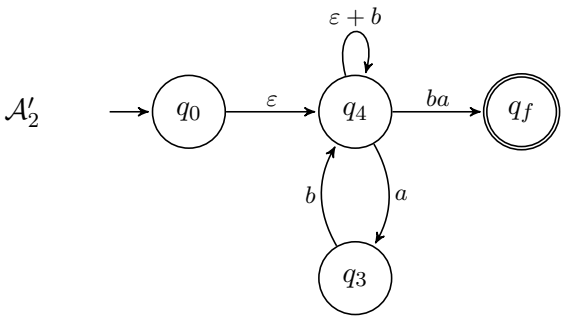


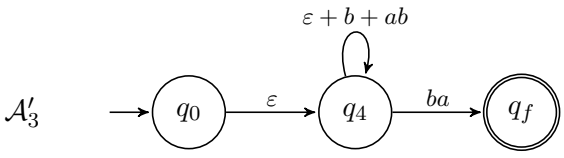
La première étape est la construction de l'automate $\mathcal{A}'_0$; ici nous étiquetons les transitions avec des expressions rationnelles, et ne faisons pas apparaître les transitions étiquetées par $\emptyset$, ni les boucles élémentaires uniquement étiquetées par ε (qui devraient apparaître sur chacun des états q_1 à q_3) :

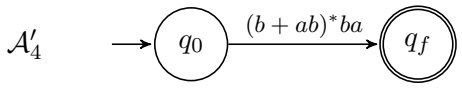


On obtient alors successivement les automates









Lemme d'ARDEN et élimination de GAUSS

C.f. [Car08, p. 40], [Sak03, p. 107], [3, p. 306].

Voici enfin une troisième méthode pour calculer une expression rationnelle à partir d'un automate fini. Le principe est d'exprimer les langages de chaque état p de Q dans un système d'équations d'inconnues X_p, à valeur dans 2^{Σ^*} :

$$X_p = \begin{cases} \sum_{(p,a,q)\in\delta} aX_q + \varepsilon & \text{si } p \in F \\ \sum_{(p,a,q)\in\delta} aX_q & \text{sinon} \end{cases}$$

Comme $L(\mathcal{A})$ est l'union finie des solutions $\bigcup_{p\in I} X_p$, il suffit de montrer que ces solutions sont elles-même rationnelles.

Un tel système se résout par une élimination de GAUSS, en ordonnant les X_i. On exprime alors chaque X_i comme la solution d'une équation $X_i = K_iX_i + L_i$ avec K_i et L_i des langages sur $\Sigma \uplus \{X_{i+1}, \ldots, X_n\}$ pour $n = |Q|$, équation que l'on résout en utilisant le lemme d'ARDEN ci-dessous. On substitue ensuite cette solution à X_i dans les équations pour X_j, $j > i$. Le fait que ces solutions soient rationnelles découle de la rationnalité de K_1 et L_1 (qui sont dans $\mathrm{Rat}((\Sigma \uplus \{X_2, \ldots, X_n\})^*)$), puis de X_1 par le lemme d'ARDEN, et par induction pour chacun des K_i et L_i.

Lemme 1.43 (ARDEN)**.** *Soient K et L deux langages sur Σ, et l'équation $X = KX + L$ d'inconnue X à valeur dans 2^{Σ^*}. Alors*

1. *$X = K^*L$ est la plus petite solution,*
2. *si $\varepsilon \notin K$, $X = K^*L$ est l'unique solution,*
3. *si $\varepsilon \in K$, les solutions sont $X = K^*Y$ où $L \subseteq Y \subseteq \Sigma^*$.*

Démonstration. Montrons tout d'abord que K^*L est une solution de l'équation. En effet, $K \cdot (K^*L) + L = K^+L + L = (K^+ + \varepsilon)L = K^*L$.

1. Montrons maintenant que si X est une solution, alors $K^*L \subseteq X$, et donc que K^*L est la plus petite solution. Procédons par induction et montrons pour cela que $K^nL \subseteq X$ pour tout $n \geq 0$. Initialement, $L \subseteq X$. Par suite, $KX \subseteq X$ et en appliquant l'hypothèse d'induction $K^nL \subseteq X$, on déduit $K^{n+1}L \subseteq X$.
2. Si $\varepsilon \notin K$, montrons que toute solution X vérifie $X \subseteq K^*L$, qui sera alors l'unique solution. Par l'absurde, soit w le mot de longueur minimale dans $X \backslash K^*L$. Alors $w \notin L$, et donc nous n'avons pas d'autre choix que $w \in KX$ pour satisfaire l'équation. On en déduit $w = uv$ avec $u \in K$ différent de ε et v dans X est strictement plus court que w, donc v est aussi dans K^*L. Mais alors $w \in K^*L$, une contradiction.
3. Si $\varepsilon \in K$, soit un langage Y sur Σ tel que $L \subseteq Y$, alors K^*Y est solution de l'équation. Inversement, si X est une solution, $L \subseteq K^*L \subseteq X$ et par induction K^*X est aussi solution.

□

À noter que si l'on part d'un automate sans ε-transitions, on n'utilise alors que l'énoncé (2) du lemme – l'énoncé (3) est là pour répondre aux jurys facétieux...

Complexité des expressions rationnelles

Si l'on dispose de complexités raisonnables pour la conversion d'une expression rationnelle en un automate équivalent, ce n'est pas le cas des expressions rationnelles que l'on peut construire à partir d'un automate fini.

Borne supérieure Le premier point, le plus simple, est que l'on peut trouver au pire une expression E avec $\|E\| = O(4^n)$ pour un automate de taille n. Cette borne s'obtient en examinant l'algorithme de McNaughton et Yamada, et en posant S_k pour la taille de la plus grande expression $L_{p,q}^{(k)}$:

$$\begin{aligned} S_k &= \max\{\|L_{p,q}^{(k)}\| \mid p, q \in Q\} \\ S_0 &\leq |\Sigma| \\ S_k &\leq 4S_{k-1} \ . \end{aligned}$$

La preuve de cette borne inférieure est tirée de Ehrenfeucht *et* Zeiger *[11] ; voir aussi [SSS88, exercice 3.16].*

Borne inférieure Le second point est qu'il existe une famille d'automates de taille n^2 pour laquelle toute expression rationnelle équivalente vérifie $|E| \geq 2^{n-1}$. Soit en effet l'automate $\mathcal{A}_n = \langle Q_n, \Sigma_{n^2}, \{1\}, \{1\}, \delta\rangle$ défini par

$$\begin{aligned} Q_n &= \{1, \ldots, n\} \\ \Sigma_{n^2} &= \{a_{ij} \mid i, j \in Q_n\} \\ \delta &= \{(i, a_{ij}, j) \mid i, j \in Q_n\} \ . \end{aligned}$$

Le graphe sous-jacent de $\mathcal{A}_n$ est le graphe complet sur l'ensemble de sommets $\{1, \ldots, n\}$.

Définition 1.44 (Expression normale)**.** Une expression rationnelle est *normale* à un automate $\mathcal{A} = \langle Q, \Sigma, I, F, \delta\rangle$ s'il existe deux fonctions i et f des sous-expressions de E, notées sub(E), dans Q telles que

1. si $E_1 + E_2 \in \text{sub}(E)$, alors $i(E_1) = i(E_2) = i(E_1 + E_2)$ et $f(E_1) = f(E_2) = f(E_1 + E_2)$,
2. si $E_1E_2 \in \text{sub}(E)$, alors $i(E_1) = i(E_1E_2)$, $f(E_2) = f(E_1E_2)$, et $f(E_1) = i(E_2)$,
3. si $E_1^* \in \text{sub}(E)$, alors $i(E_1) = i(E_1^*) = f(E_1) = f(E_1^*)$,
4. si $E_1 \in \text{sub}(E)$, alors $L(E_1) \subseteq L_{i(E_1),f(E_1)}$.

Remarquons que *toute* expression rationnelle E telle que $L(E) = L(\mathcal{A}_n)$ est normale à $\mathcal{A}_n$: il suffit de trouver le point de découpe dans une sous-expression E_1E_2 (les autres cas ne laissant pas de choix), et pour cela d'inspecter $P(E_2)$ (c.f. proposition 1.33) pour trouver a_{ij} et en déduire $f(E_1) = i(E_2) = i$. On pourrait procéder de manière équivalente en inspectant $S(E_1)$ pour trouver un symbole a_{ki}. Ce choix de i est bien unique : en effet, s'il existait aussi a_{ml} dans $P(E_2)$ avec $i \neq m$, alors de E_1E_2 on pourrait déduire un facteur $a_{ki}a_{ml}$ dans un mot de $L(E)$, contredisant $L(E) = L(\mathcal{A}_n)$.

Définition 1.45 (Couverture d'un mot)**.** Une expression rationnelle E *couvre* un mot u de Σ^+ si $u \in \text{Fact}(L(E))$, où Fact(L) dénote l'ensemble des facteurs de L.

Définition 1.46 (Index d'un mot)**.** L'*index* $I_u(E)$ d'un mot u dans une expression E est la plus grande valeur de m telle que E couvre u^m. Si $I_u(E) = \infty$, alors on dit que E est u-infinie, et sinon qu'elle est u-finie.

Lemme 1.47. *Soit u un mot de Σ^+ et E une expression rationnelle sur Σ. Si E est u-finie, alors*

$$I_u(E) \leq |E|$$

Démonstration. On vérifie en général

$$\begin{aligned} I_u(\emptyset) &= 0 \\ I_u(\varepsilon) &= 0 \\ I_u(a) &\leq 1 \\ I_u(E_1 + E_2) &= \max(I_u(E_1), I_u(E_2)) \\ I_u(E_1E_2) &\leq I_u(E_1) + I_u(E_2) + 1 \\ I_u(E^*) &= \sup\{I_u(E^n) \mid n \geq 0\} \ . \end{aligned}$$

Le seul point potentiellement surprenant est le $+1$ dans l'inégalité pour E_1E_2, qui provient de la possibilité de découper u en u_1u_2, et d'avoir $u^{I_u(E_1)}u_1$ dans $\mathrm{Suff}(L(E_1))$ et $u_2u^{I_u(E_2)}$ dans $\mathrm{Pref}(L(E_2))$.

L'expression de $I_u(E^*)$ dissimule en fait deux cas, selon que $I_u(E^*) = \infty$ ou que $I_u(E^*) \leq I_u(E) + 1$. Cette remarque permet de conclure. □

On déduit la borne cherchée de la proposition suivante.

Proposition 1.48. *Il existe une boucle u dans $\mathcal{A}_n$ passant par l'état 1 telle que, pour toute expression E couvrant u, $|E| \geq 2^n$.*

Démonstration. On procède par induction sur n. Dans $\mathcal{A}_1$, la seule boucle possible est celle sur a_{11}, avec pour expression $E = a_{11}^*$ de taille 2. Supposons maintenant que l'on ait trouvé une boucle u dans $\mathcal{A}_{n-1}$ vérifiant l'énoncé, i.e. que pour toute expression E_{n-1} couvrant u, $|E_{n-1}| \geq 2^{n-1}$.

Dans l'automate $\mathcal{A}_n$, pour tout k de Q_n, on considère les chemins u_k, permutations circulaires de u qui commencent et finissent en l'état k. Par exemple, si $u = a_{11}a_{13}a_{34}a_{45}a_{51}$ dans $\mathcal{A}_5$, alors dans $\mathcal{A}_6$ on aura

$$\begin{aligned} u_1 &= a_{11}a_{13}a_{34}a_{45}a_{51} = u \\ u_2 &= a_{22}a_{24}a_{45}a_{56}a_{62} \\ u_3 &= a_{33}a_{35}a_{56}a_{61}a_{13} \\ u_4 &= a_{44}a_{46}a_{61}a_{12}a_{24} \\ &\dots \end{aligned}$$

On vérifie ainsi que u_k boucle sur k mais ne passe pas par l'état $k-1$.

Considérons à présent la boucle

$$w = u_1^{2^n}a_{12}u_2^{2^n}a_{23}\cdots u_n^{2^n}a_{n1} \ .$$

Soit E une expression couvrant w. Par définition de w, pour tout k, $I_{u_k}(E) \geq 2^n$, et donc par le lemme 1.47, soit $|E| \geq 2^n$ et nous avons notre preuve, soit E est u_k-infinie.

Dans ce dernier cas, pour chaque k, on peut trouver des sous-expressions F_k^* de $\mathrm{sub}(E)$ minimales (pour l'ordre « sous-expression de ») telles que ces F_k^* soit elles aussi u_k-infinies et couvrent u_k. On considère alors parmis ces sous-expressions

– un état $j = i(E_1^*)$ pour l'une de ces sous-expressions E_1^* couvrant un certain chemin u_k, et

– une autre sous-expression E_2 pour la boucle $u_{(j \mod n)+1}$, qui par définition étiquette un chemin qui ne passe pas par j.

On observe que, si l'on substitue ε pour E_1 dans E (et donc indirectement dans E_2), alors E_2 couvre toujours $u_{(j \mod n)+1}$.

Dès lors,

$|E_2^*| \geq 2^{n-1}$ après substitution, par hyp. d'ind. puisque E_2^* couvre $u_{(j \mod n)+1}$

$|E_1^*| \geq 2^{n-1}$ avant substitution, par hyp. d'ind. puisque E_1^* couvre u_k

$|E| \geq 2^n$ avant substitution

□

1.3 Minimisation

C.f. [Car08, sec. 1.7], [Sak03, sec. I.3.3], [Aut94, ch. 6], [HU79, sec. 3.4].

Morphismes d'automates Nous allons souvent utiliser dans cette section la notion de morphisme d'automates :

Définition 1.49 (Morphisme d'automates)**.** Un *morphisme d'automates* $\varphi : \mathcal{A}_1 \to \mathcal{A}_2$ entre deux automates $\mathcal{A}_1 = \langle \Sigma, Q_1, I_1, F_1, \delta_1 \rangle$ et $\mathcal{A}_2 = \langle \Sigma, Q_2, I_2, F_2, \delta_2 \rangle$ est une fonction de Q_1 dans Q_2 qui vérifie

1. $\varphi(I_1) \subseteq I_2$,
2. $\varphi(F_1) \subseteq F_2$ et
3. si (q, a, q') est une transition de δ_1, alors $(\varphi(q), a, \varphi(q'))$ est une transition de δ_2.

C.f. [Sak03, chap. II.3].

On obtient par induction le lemme suivant :

Lemme 1.50. *Soit $\varphi : \mathcal{A}_1 \to \mathcal{A}_2$ un morphisme d'automates. Alors $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$.*

Une condition garantissant l'équivalence des langages d'un automate et de son image par un morphisme d'automate est la surjectivité, définie par :

Définition 1.51. Un morphisme d'automates $\varphi : \mathcal{A}_1 \to \mathcal{A}_2$ est *localement surjectif* si $\varphi(I_1) = I_2$ et pour tout état q de Q_1, et toute transition $(\varphi(q), a, q')$ de δ_2, il existe une transition (q, a, q'') de δ_1 telle que $q' = \varphi(q'')$ et $q'' \in F_1$ si $q' \in F_2$.

À noter que dans le cas d'automates déterministes complets, la surjectivité locale de φ se résume à demander que $\varphi(q) \in F_2$ implique $q \in F_1$ pour tout q de Q_1.

Lemme 1.52. *Si $\varphi : \mathcal{A}_1 \to \mathcal{A}_2$ est un morphisme localement surjectif d'automates, alors $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.*

La surjectivité locale n'interdit pas d'avoir des états non accessibles dans Q_2 sans antécédent dans Q_1.

Démonstration. D'après le lemme 1.50, il ne nous reste qu'à montrer que $L(\mathcal{A}_2) \subseteq L(\mathcal{A}_1)$. Soit donc une exécution de $\mathcal{A}_2$

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$$

avec q_0 dans I_2 et q_n dans F_2. On montre par récurrence sur i de 0 à n qu'il existe une exécution de $\mathcal{A}_1$

$$q'_0 \xrightarrow{a_1} q'_1 \xrightarrow{a_2} q'_2 \cdots q'_{i-1} \xrightarrow{a_i} q'_i$$

avec q'_0 dans I_1 et $\varphi(q'_i) = q_i$. Tout d'abord, comme $\varphi(I_1) = I_2$, il existe bien un antécédent $q'_0 \in I_1$ pour q_0 par φ. Puis, s'il existe une telle exécution dans $\mathcal{A}_1$ jusqu'à q'_i, alors comme $(\varphi(q'_i), a, q_{i+1})$ appartient à δ_2, il existe (q'_i, a, q'_{i+1}) dans δ_1 avec $q_{i+1} = \varphi(q'_{i+1})$.

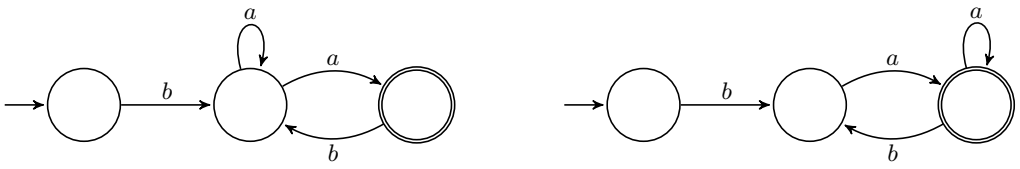
Il ne nous reste plus pour conclure qu'à rappeler que si q_n est dans F_2, alors l'état q'_n fourni par l'induction précédente est lui dans F_1, et donc que le mot $a_1 \cdots a_n$ est accepté par $\mathcal{A}_1$. □

Enfin, un morphisme d'automates est surjectif s'il est localement surjectif et si $\varphi(Q_1) = Q_2$. On définit un morphisme injectif de manière duale. En particulier, deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$ sont *isomorphes* s'il existe un morphisme bijectif de $\mathcal{A}_1$ à $\mathcal{A}_2$.

Les automates finis déterministes ont une forme minimale *canonique* modulo isomorphisme. Ce résultat est à comparer à l'absence d'un tel automate canonique dans le cas non déterministe.

Même en l'absence d'une forme canonique, il est intéressant de minimiser des automates non déterministes. Ce problème a été montré PSPACE*-complet par* JIANG *et* RAVIKUMAR *[17].*

Exemple 1.53. Le langage décrit par l'expression $(ba^+)^+$ est reconnu par les deux automates non isomorphes suivants :



On pourrait énumérer modulo isomorphisme tous les automates avec deux états, ou trois états et trois transitions, et vérifier qu'il n'en existe aucun qui reconnaisse ce langage.

1.3.1 Automate des quotients

C.f. [Car08, déf. 1.83], [Sak03, p. 121], [Aut94, p. 64], [3, p. 312].

Rappelons que $w^{-1}L = \{u \in \Sigma^* \mid wu \in L\}$ est le quotient à gauche (ou *résiduel*) de $L \subseteq \Sigma^*$ par $w \in \Sigma^*$, et notons $L_q = \{w \in \Sigma^* \mid \delta^*(q, w) \cap F \neq \emptyset\}$ pour un état q d'un automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$. On a les équations

$$v^{-1}u^{-1}L = (uv)^{-1}L$$
$$L_q = \bigcup_{a \in \Sigma} \bigcup_{(q,a,q') \in \delta} \{a\} L_{q'}$$

pour tout u, v de Σ^*, $L \subseteq \Sigma^*$, et q de Q.

Lemme 1.54. *Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe, q un état de Q, et w un mot de Σ^*. Si $\delta^*(q_0, w) = q$, alors $L_q = w^{-1}L$.*

Démonstration. Par récurrence sur $|w|$: initialement, $L_{q_0} = L = \varepsilon^{-1}L$, puis pour $w = ua$ avec u dans Σ^* et a dans Σ, et $q_0 \xrightarrow{u} q' \xrightarrow{a} q$, par hypothèse de récurrence $L_{q'} = u^{-1}L$ et on vérifie bien $L_q = a^{-1}L_{q'} = a^{-1}u^{-1}L = (ua)^{-1}L$ puisque l'automate est déterministe. □

Le lemme 1.54 permet de majorer le nombre de quotients à gauche d'un langage reconnaissable ; en particulier, ce nombre est fini.

Corollaire 1.55. *Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe complet. Le nombre de quotients à gauche de $L(\mathcal{A})$ est inférieur à $|Q|$.*

Nous introduisons maintenant un automate particulier dont les états sont des langages sur Σ, et plus précisément les quotients à gauche de L par Σ^* :

Définition 1.56 (Automate des quotients)**.** L'*automate des quotients* d'un langage $L \subseteq \Sigma^*$ est $\mathcal{A}_L = \langle \Sigma, Q_L, L, F_L, \delta_L \rangle$ défini par

$$Q_L = \{w^{-1}L \mid w \in \Sigma^*\}$$
$$F_L = \{w^{-1}L \mid \varepsilon \in w^{-1}L\} = \{w^{-1}L \mid w \in L\}$$
$$\delta_L = \{(w^{-1}L, a, (wa)^{-1}L) \mid w \in \Sigma^*, a \in \Sigma\}$$

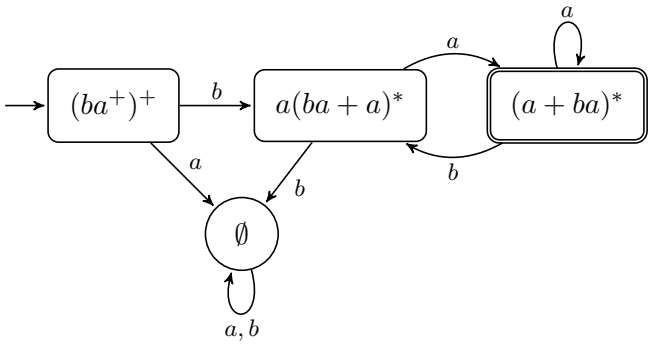
On déduit de cette définition que $\mathcal{A}_L$ est un automate déterministe complet accessible. On vérifie aisément dans $\mathcal{A}_L$ que $\delta^*(L, w) = w^{-1}L$, et donc que $L(\mathcal{A}_L) = L$. Par le corollaire 1.55, si L est reconnaissable, alors $\mathcal{A}_L$ est fini et minimal.

Une autre démarche pour ces résultats est de démontrer la proposition suivante :

Proposition 1.57. *Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe complet reconnaissant L et $\mathcal{A}_L = \langle \Sigma, Q_L, L, F_L, \delta_L \rangle$ l'automate des quotients associé. La fonction $\varphi : Q \to Q_L, q \mapsto L_q$ est un morphisme surjectif d'automates.*

Démonstration. Tout d'abord, φ est bien un morphisme d'automates : $\varphi(q_0) = L$, $\varphi(F) = F_L$, et si (q, a, q') est une transition de δ, alors $(L_q, a, L_{q'})$ est une transition de δ_L. De plus, φ est localement surjectif, puisque, encore une fois $\varphi(q_0) = L$ et, comme $\mathcal{A}$ est complet, pour toute transition $(L_q, a, w^{-1}L)$ de δ_L il existe une transition (q, a, q') de δ telle que $L_{q'} = w^{-1}L$, et $q' \in F$ si $L_{q'} \in F_L$. Enfin, puisque $\mathcal{A}_L$ est accessible, $\varphi(Q) = Q_L$, donc φ est surjectif. □

Exemple 1.58. Les quotients à gauche du langage L de l'expression $(ba^+)^+$ sont L, $\emptyset$, $L(a(ba + a)^*)$ et $L((a + ba)^*)$, ce qui fournit l'automate $\mathcal{A}_L$ suivant :



1.3.2 Algorithme par renversement de BRZOZOWSKI

C.f. BRZOZOWSKI *[6] et [Sak03, p. 125].*

Voici un résultat bien utile pour démontrer qu'un automate que l'on vient de déterminiser est minimal (c.f. l'automate de l'exemple 1.14) :

Proposition 1.59. *Le déterminisé d'un automate co-déterministe co-accessible qui reconnaît L est minimal.*

Démonstration. Soient $\mathcal{A} = \langle \Sigma, Q, I, \{q_f\}, \delta \rangle$ un automate co-déterministe et co-accessible qui reconnaît L, et $\mathcal{A}_d = \langle \Sigma, 2^Q, I, F, \delta \rangle$ son déterminisé. Montrons que le morphisme $\varphi : 2^Q \to Q_L$ est injectif, i.e. que si $u^{-1}L = v^{-1}L$ pour deux mots de Σ^*, alors $\delta^*(I, u) = \delta^*(I, v)$. On aura ainsi montré que $\mathcal{A}_d$ est isomorphe à l'automate minimal $\mathcal{A}_L$.

Supposons $u^{-1}L = v^{-1}L$ et soit un état p de $\delta^*(I, u)$, montrons qu'il appartient aussi à l'ensemble $\delta^*(I, v)$. Comme $\mathcal{A}$ est co-accessible, il existe un mot w de Σ^* tel que q_f appartienne à $\delta^*(p, w)$. Donc uw et vw appartiennent à L. Comme $\mathcal{A}$ est co-déterministe, p est l'unique état tel que q_f appartienne à $\delta^*(p, w)$, donc p appartient aussi à $\delta^*(I, v)$.

Symmétriquement, on prouve aussi $\delta^*(I, v) \subseteq \delta^*(I, u)$. □

On en déduit un algorithme extrêmement simple pour minimiser un automate (y compris un automate non déterministe) : calculer l'automate transposé, le déterminiser, prendre son transposé (on a alors un automate co-accessible et co-déterministe), et déterminiser à nouveau :

$$\mathcal{A}_L = \det(\mathrm{tr}(\det(\mathrm{tr}(\mathcal{A})))) \ .$$

Une transposition se fait en temps linéaire, une déterminisation en temps exponentiel. A priori, cet algorithme travaille donc en temps $O(2^{2^n})$. Cependant, comme l'automate final est minimal, on sait qu'il est plus petit que $\det(\mathcal{A})$, donc de taille au pire $O(2^n)$. L'argument est donc le suivant : le coût d'une déterminisation est en $O(|\mathcal{A}| + |\det(\mathcal{A})|)$, l'automate intermédiaire $\mathrm{tr}(\det(\mathrm{tr}(\mathcal{A})))$ est de taille $O(2^n)$, et l'automate final aussi, d'où une complexité totale en $O(2^n)$.

Cette complexité est particulièrement intéressante si $\mathcal{A}$ est initialement non déterministe, puisque les autres algorithmes de minimisation doivent commencer par une déterminisation. En pratique, même dans le cas où $\mathcal{A}$ est déterministe, cet algorithme reste performant.

1.3.3 Congruence de NERODE

C.f. [Car08, sec. 1.7.2], [Sak03, déf. I.3.3], [Aut94, p. 60], [3, p. 315].

Définition 1.60 (Congruence)**.** Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe, et $\sim$ une relation d'équivalence sur Q. Cette relation est une *congruence* si elle est compatible avec les transitions de $\mathcal{A}$:

$$q \sim q' \text{ implique } \forall a \in \Sigma, \delta(q, a) \sim \delta(q', a) \tag{1.4}$$

et si elle sature $\mathcal{A}$:

$$q \sim q' \text{ implique } q \in F \text{ ssi } q' \in F \tag{1.5}$$

pour tous états q, q' de Q.

Notons $[q]$ pour la classe d'équivalence de l'état q.

Définition 1.61 (Quotient)**.** Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe et $\sim$ une équivalence sur Q. L'*automate quotient* de $\mathcal{A}$ par $\sim$ est l'automate $\mathcal{A}/\!\sim = \langle \Sigma, Q/\!\sim, [q_0], \{[q_f] \mid q_f \in F\}, \delta/\!\sim\}$ avec

$$\delta/\!\sim = \{([q], a, [\delta(q, a)]) \mid q \in Q, a \in \Sigma\} .$$

Observons que $\mathcal{A}/\!\sim$ est déterministe et complet.

Lemme 1.62. *Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe et $\sim$ une congruence sur Q. L'automate quotient $\mathcal{A}/\!\sim$ reconnaît $L(\mathcal{A})$.*

Démonstration. L'inclusion $L(\mathcal{A}) \subseteq L(\mathcal{A}/\!\sim)$ est immédiate pour toute équivalence $\sim$. Soit maintenant $w = a_1 \cdots a_n \in L(\mathcal{A}/\!\sim)$, reconnu par une exécution

$$[q_0] \xrightarrow{a_1} [q_1] \to \cdots \xrightarrow{a_n} [q_n]$$

avec $q_n \sim q_f$ pour un état final q_f de F. En particulier, cette exécution ne passe pas par la classe d'équivalence vide (qui sert d'état puit).

On montre par récurrence sur n que w étiquette une exécution de $\mathcal{A}$

$$q_0 \xrightarrow{a_1} q'_1 \to \cdots \xrightarrow{a_n} q'_n$$

avec $q'_i \sim q_i$ pour chaque i de 1 à n. C'est vrai par (1.4) pour $i = 1$, et de même pour l'hypothèse de récurrence.

Enfin, $q'_n \sim q_n \sim q_f$ pour un certain état final q_f (avec $q'_n = q_0$ si $w = \varepsilon$), et par la propriété de saturation (1.5), $q'_n \in F$, donc w est bien dans $L(\mathcal{A})$. □

Définition 1.63 (Congruence de NERODE)**.** Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe. On appelle la relation d'équivalence $\cong$ suivante la *congruence de* NERODE :

$$q \cong q' \text{ ssi } L_q = L_{q'}$$

pour tous états q, q' de Q.

Sans surprise, la congruence de NERODE est bien une congruence au sens de la définition 1.60. La définition de $\cong$ donne immédiatement :

Proposition 1.64. *Soit $\mathcal{A}$ un automate déterministe complet pour L. Alors $\mathcal{A}/\cong$ est isomorphe à $\mathcal{A}_L$.*

1.3.4 Algorithme de MOORE

C.f. [Car08, sec. 1.7.3], [Sak03, proposition I.3.12], [Aut94, p. 61], [HU79, sec. 3.4], [3, p. 318].

La proposition 1.64 suggère un moyen de calculer l'automate minimal, en opérant à des raffinements successifs d'une congruence jusqu'à obtenir la congruence de NERODE.

Définition 1.65. Soit $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ un automate déterministe. On définit $\cong_i$ sur Q par

$$q \cong_0 q' \text{ ssi } (q \in F \text{ ssi } q' \in F) \tag{1.6}$$
$$q \cong_{i+1} q' \text{ ssi } q \cong_i q' \text{ et } \forall a \in \Sigma, \delta(q,a) \cong_i \delta(q',a) \tag{1.7}$$

pour tous états q, q' de Q.

Proposition 1.66. *Il existe k tel que $\cong_k = \cong_{k+j}$ pour tout $j \geq 0$, et tel que $\cong_k = \cong$.*

Démonstration. On vérifie par récurrence que $\cong \subseteq \cong_i$ pour tout $i \geq 0$: $\cong \subseteq \cong_0$ est vérifié puisque $L_q = L_{q'}$ implique $q \in F$ ssi $q' \in F$, puis $L_q = L_{q'}$ implique $q \cong_i q'$ par hypothèse de récurrence, et pour tout a de Σ, $L_{\delta(q,a)} = L_{\delta(q',a)}$, donc encore par hypothèse de récurrence $\delta(q,a) \cong_i \delta(q',a)$.

Notons que $\cong_{i+1} \subseteq \cong_i$ pour tout i, donc $\cong_{i+1}$ est d'index supérieur à celui de $\cong_i$, mais d'index inférieur à celui de $\cong$, qui est le nombre (fini) d'états de l'automate minimal. Donc il existe k tel que $\cong_k = \cong_{k+1} = \cong_{k+j}$ pour tout $j \geq 0$.

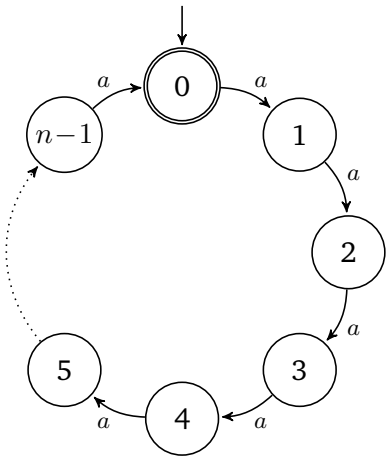
Il reste à montrer que $\cong_k \subseteq \cong$. Soient q et q' deux états de Q tels que $q \cong_k q'$, et $w = a_1 \cdots a_n$ dans L_q. Alors il existe deux exécutions dans $\mathcal{A}$

$$q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n \qquad\qquad q' \xrightarrow{a_1} q'_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q'_n$$

avec $q_j \cong_k q'_j$ pour tout $j \geq 1$ par (1.7), et $q_n \in F$ ssi $q'_n \in F$ par (1.6), donc w est aussi dans $L_{q'}$. Cela montre $L_q \subseteq L_{q'}$, et on démontre de même l'inclusion inverse. □

L'algorithme de MOORE consiste alors à calculer les partitions successives de Q par $\cong_i$.

Exemple 1.67. L'algorithme de MOORE travaille en temps $O(n^2)$ dans le pire des cas, que l'on peut rencontrer avec l'automate suivant, qui reconnaît le langage $\{a^m \mid m = 0 \mod n\}$:



La partition initiale est

$$Q/\cong_0 = \{\{0\}, \{1, \ldots, n-1\}\}$$

puis pour chaque i

$$Q/\cong_i = \{\{0\}, \{1, \ldots, n-i-1\}, \{n-i\}, \ldots, \{n-1\}\}$$

et enfin pour $i = n - 1$

$$Q/\cong = \{\{0\}, \{1\}, \ldots, \{n-1\}\} \ .$$

Il existe une version optimisée par HOPCROFT de cet algorithme, qui utilise une technique de diviser pour régner, en $O(n \log n)$. En pratique, elle est en fait moins efficace !

1.4 Applications

Voici enfin quelques applications des automates finis et des expressions rationnelles. Les deux premières sont tellement classiques qu'elles sont même officiellement au programme de l'agrégation... mais on trouvera des exemples plus originaux dans les sections suivantes.

1.4.1 Localisation

Cette section reprend un chapitre de CROCHEMORE *et* HANCART *[10] ; voir aussi* BEAUQUIER *et al. [3, sec. 10.1.6, p. 354],* AHO *[1] et [Sak03, sec. I.5.3].*

Sous cette dénomination se cachent de nombreuses applications, depuis la recherche d'une chaîne dans des fichiers (par exemple avec `grep`), jusqu'à celle de motifs approximatifs dans une séquence ADN. Pour limiter le champ de ces algorithmes de *pattern matching*, on se contente ici de la recherche d'un mot fini m dans un texte t sur un alphabet fini Σ connu à l'avance.

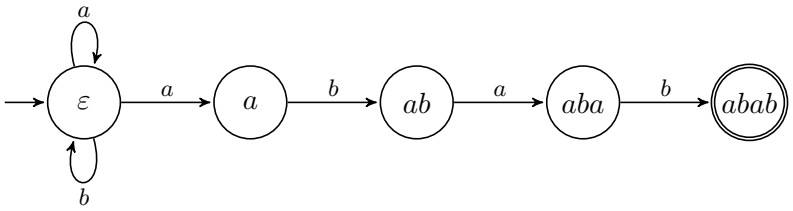
Algorithme 1.68.
LOCALISATION NAÏVE $(m = a_1 \cdots a_p,\ t = b_1 \cdots b_n)$

1. $i \leftarrow 1$
2. $j \leftarrow 1$
3. **tant que** $j \leq n$ **faire**
4. **tant que** $i \leq p \wedge j \leq n$ **faire**

5. **si** $a_i = b_j$ **alors**
6. $i \leftarrow i + 1$
7. $j \leftarrow j + 1$
8. **sinon**
9. $j \leftarrow j - i + 2$
10. $i \leftarrow 1$
11. **fin si**
12. **fin tant que**
13. **si** $i > p$ **alors**
14. **afficher** « motif trouvé en $b_{j-p} \cdots b_j$ »
15. $j \leftarrow j - i + 2$
16. $i \leftarrow 1$
17. **fin si**
18. **fin tant que**

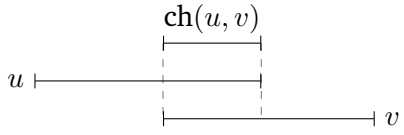
Cet algorithme fonctionne en temps $O(|m| \cdot |t|)$ et espace $O(|m|)$. On obtient un meilleur algorithme de localisation en utilisant un automate fini déterministe pour le langage $\Sigma^* m$: à chaque passage par son état final, on aura trouvé le motif recherché – une utilisation originale des états finaux.

Exemple 1.69. Un automate non déterministe pour le motif $abab$ sur $\{a, b\}^*$ est le suivant :

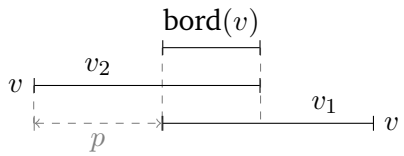


L'utilisation de bordures est aussi au cœur de l'algorithme de KNUTH *et al. [19] qui travaille en temps $O(|m| + |t|)$ et espace $O(m)$. L'idée d'utiliser l'automate minimal pour $\Sigma^* m$ avec une transition par défaut est dûe à* SIMON *[24].*

Définition 1.70 (Chevauchement, bordure, période)**.** Le *chevauchement* $\mathrm{ch}(u, v)$ de deux mots u et v est le plus long suffixe de u qui est un préfixe de v.



Un mot u est une *bordure* d'un mot v s'il est à la fois un préfixe et un suffixe de v, c'est-à-dire s'il existe v_1, v_2 dans Σ^* tels que $v = uv_1 = v_2u$. La bordure la plus longue de v différente de v est notée $\mathrm{bord}(v)$.



Une *période* d'un mot $v = a_1 \cdots a_n$ est un entier p entre 0 et n tel que, pour tout $1 \leq i \leq n - p$, $a_i = a_{i+p}$.

À noter que si un mot v a une bordure non vide, alors $|v| - |\mathrm{bord}(v)|$ est une période de v.

Définition 1.71 (Automate d'un motif)**.** On définit pour un motif m de Σ^* la fonction de retour r_m par

$$r_m(ua) = \begin{cases} ua & \text{si } ua \text{ est un préfixe de } m \\ \text{bord}(ua) & \text{sinon} \end{cases}$$

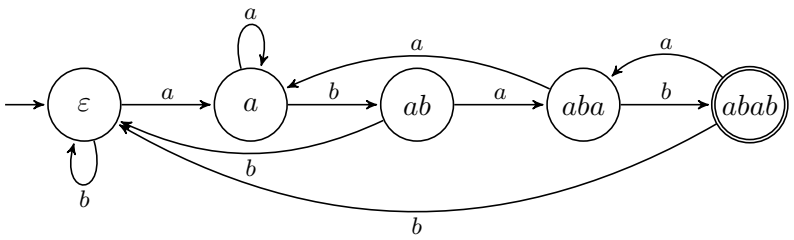
pour tout préfixe u de m et toute lettre a de Σ.

L'*automate du motif* m sur Σ^* est l'automate fini

$$\mathcal{A}_m = \langle \Sigma, \text{Pref}(m), \varepsilon, \{m\}, \{(u, a, r_m(ua)) \mid u \in \text{Pref}(m), a \in \Sigma\} \rangle$$

où $\text{Pref}(m)$ est l'ensemble des préfixes de m.

Exemple 1.72. L'automate du motif $abab$ sur $\{a, b\}^*$ est le suivant :



Proposition 1.73. *L'automate $\mathcal{A}_m$ d'un motif m est l'automate minimal pour le langage $\Sigma^* m$.*

Démonstration. Le fait que $\mathcal{A}_m$ soit déterministe et complet est immédiat d'après les définitions.

Par récurrence sur la longueur du mot u de Σ^*, on montre que

$$\delta^*(\varepsilon, u) = \text{ch}(u, m)$$

dans $\mathcal{A}_m$. C'est vrai pour $u = \varepsilon = \text{ch}(\varepsilon, m)$, et pour $\delta(\text{ch}(v, m), a) = \text{ch}(va, m)$. En prenant $u = vm$ avec v dans Σ^*,

$$\delta^*(\epsilon, vm) = \text{ch}(vm, m) = m$$

d'où l'on déduit que $L(\mathcal{A}_m) = \Sigma^* m$.

L'automate $\mathcal{A}_m$ est de plus minimal puisque les langages acceptés depuis chaque état sont bien différents : soient u_1 et u_2 deux préfixes différents de $m = u_1 v_1 = u_2 v_2$; on peut supposer $|u_1| < |u_2|$ et donc $|v_1| > |v_2|$, alors v_2 ne peut pas être dans le langage accepté à partir de l'état u_1, sinon $u_1 v_2$ de longueur inférieure à $|m|$ serait accepté par l'automate entier. □

Voir l'algorithme de minimisation par renversement de BRZOZOWSKI *en section 1.3.2.*

On notera que l'automate d'un motif est exactement le déterminisé de l'automate non déterministe « naturel » pour $\Sigma^* m$: ce dernier étant co-déterministe, sa déterminisation donne bien un automate minimal (c.f. l'automate non déterministe de l'exemple 1.69, et son déterminisé minimal dans l'exemple 1.72).

L'automate d'un motif m a exactement $|m| + 1$ états et $|\Sigma| \cdot |m|$ transitions. Une fois l'automate construit, on sait donc effectuer la recherche d'un motif m dans un texte t en temps $O(|\Sigma| \cdot |m| + |t|)$ et espace $O(|\Sigma| \cdot |m|)$. Ce n'est pas encore très bon, du fait du facteur $|\Sigma|$ dans la taille de l'automate. La solution est d'utiliser un automate avec les transitions vers l'état ε laissées implicites : dans l'algorithme de reconnaissance d'un mot par un automate déterministe, on ira alors en l'état ε au lieu de rejeter le mot si une transition n'est pas prévue par l'automate. On tire ainsi parti de la propriété suivante sur les automates de motif.

Proposition 1.74. *Une transition (u, a, ε) d'un automate de motif est dite* triviale. *Si $\mathcal{A}_m$ est un automate de motif pour m, alors il contient au plus $2|m|$ transitions non triviales.*

Démonstration. Les transitions de la forme (u, a, ua) dans $\mathcal{A}_m$ pour u un préfixe de $m = a_1 \cdots a_p$ sont au nombre de $p = |m|$; il nous reste donc à montrer qu'il y a au plus $|m|$ transitions de la forme $(u, a, \text{bord}(ua))$ avec $\text{bord}(ua) \neq \varepsilon$. On considère pour cela la différence $|u| - |\text{bord}(ua)| \leq |m|$. Supposons qu'il existe deux telles transitions différentes $(u_1, a, \text{bord}(u_1 a))$ et $(u_2, b, \text{bord}(u_2 b))$ telles que

$$|u_1| - |\text{bord}(u_1 a)| = |u_2| - |\text{bord}(u_2 b)| \tag{1.8}$$

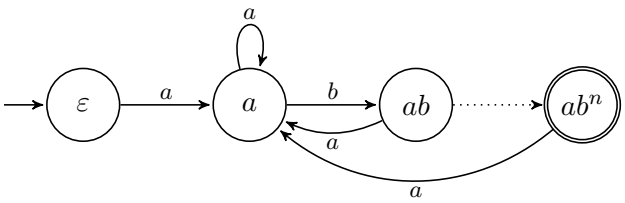
Si $u_1 = u_2$, alors d'après (1.8), $\text{bord}(u_1 a) = \text{bord}(u_2 b)$, qui implique à son tour $a = b$, en contradiction avec le fait que ces deux transitions devaient être différentes.

Si $|u_1| > |u_2|$ et donc $|\text{bord}(u_1 a)| > |\text{bord}(u_2 b)|$, alors

$$\begin{aligned}
b &= a_{|\text{bord}(u_2 b)|} && \text{puisque } \text{bord}(u_2 b) \neq \varepsilon \\
&= a_{|\text{bord}(u_2 b)|+|u_1|-|\text{bord}(u_1 a)|+1} && \text{puisque } \text{bord}(u_1 a) \neq \varepsilon \\
& && \text{donc } |u_1 a| - |\text{bord}(u_1 a)| \text{ une période de } u_1 a \\
& && \text{donc } |u_1| - |\text{bord}(u_1 a)| + 1 \text{ une période de } u_1 a \\
&= a_{|\text{bord}(u_2 b)|+|u_2|-|\text{bord}(u_2 b)|+1} && \text{par (1.8)} \\
&= a_{|u_2|+1}
\end{aligned}$$

Ce qui contredit le fait que $u_2 b$ n'était pas un préfixe de m. □

Exemple 1.75. Une borne de $2|m| - 1$ transitions non triviales est atteinte par l'automate pour le motif ab^n : pour $0 \leq i \leq n$, chaque état ab^i a une transition par a vers l'état a.

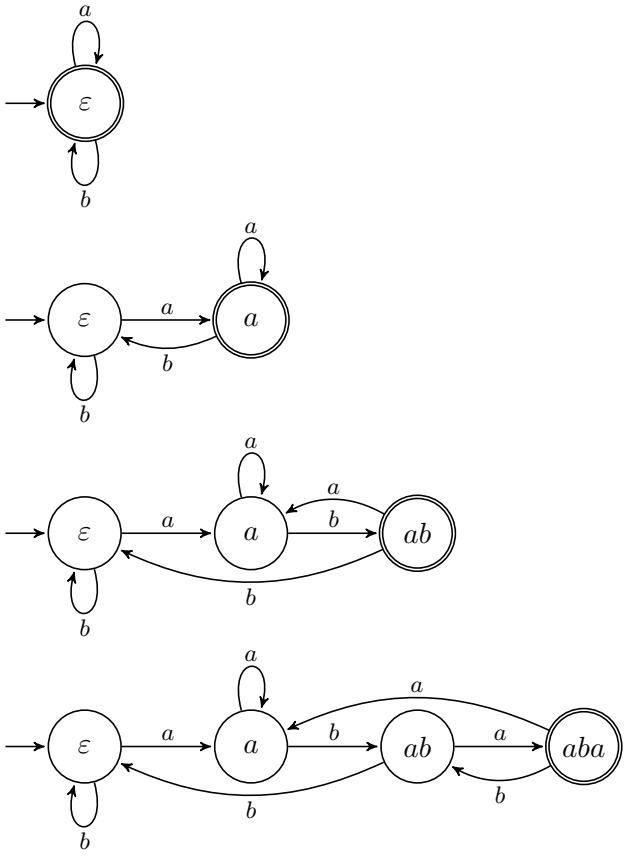


Il ne nous reste plus pour conclure cette section qu'à donner un algorithme pour construire $\mathcal{A}_m$ sans ses transitions triviales en temps et espace $O(|m|)$. L'idée de cette construction est la suivante : en partant d'un automate reconnaissant Σ^*, on « déplie » les transitions qui vont composer le motif m lettre par lettre. L'état atteint par l'ancienne transition est utilisé pour calculer les transitions depuis le nouvel état. L'utilisation de transitions par défaut est ensuite une simple adaptation.

Lemme 1.76. *Soit δ_m l'ensemble des transitions de l'automate $\mathcal{A}_m$ sur Σ. On a pour tout u de Σ^* et a de Σ :*

$$\begin{aligned}
\delta_\varepsilon &= \{(\varepsilon, b, \varepsilon) \mid b \in \Sigma\} \\
\delta_{ua} &= \delta'_{ua} \cup \delta''_{ua} \\
\delta'_{ua} &= \{(u, a, ua)\} \cup (\delta_u \backslash \{(u, a, r_u(ua))\}) \\
\delta''_{ua} &= \{(ua, b, v) \mid (r_u(ua), b, v) \in \delta'_{ua}, b \in \Sigma\}
\end{aligned}$$

Exemple 1.77. La construction incrémentale de l'automate pour $abab$ de l'exemple 1.72 passe par les étapes suivantes avant d'obtenir l'automate du motif :

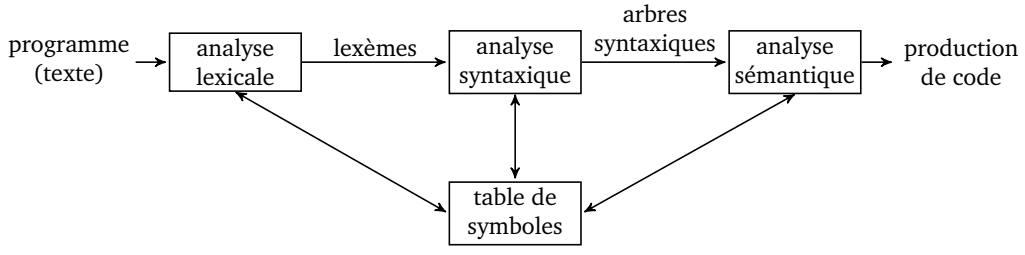


1.4.2 Analyse lexicale

C.f. [GBJL00, sec. 2.1], [SSS88, sec. 3.6], [ASU86, ch. 3], [HU79, sec. 2.8].

Une autre application des automates finis est l'analyse lexicale, ici du point de vue des langages de programmation et dans le contexte de la compilation de programmes. Comme dans le cas de la recherche de motif, les automates sont utilisés de manière non standard : il n'est pas besoin d'être à la fin de l'entrée pour s'intéresser aux états finaux rencontrés, au contraire on souhaite repérer à quels moments cela se produit. Cependant, et c'est là que l'affaire se corse par rapport aux algorithmes de localisation, on veut trouver un *découpage* du texte d'entrée à l'aide de tels états finaux, et comme plusieurs découpages sont généralement possibles, quelques acrobaties deviennent nécessaires...

Rappelons d'abord le schéma général de la *partie avant* d'un compilateur :



1. L'*analyseur lexical* reconnaît les composants élémentaires de la syntaxe, par exemple les mots-clefs, opérateurs, commentaires, etc. Le résultat est une séquence de lexèmes (ou *tokens*) qui sont passés à

2. l'*analyseur syntaxique*, qui vérifie que le programme est syntaxiquement correct et construit un arbre de syntaxe (abstrait) à partir des lexèmes ; enfin
3. l'*analyseur sémantique* statique vérifie le bon typage du programme.

Arrivé à ce point, un compilateur enchaîne encore typiquement plusieurs phases d'analyse statique pour optimiser le code machine qu'il va produire, ce qui constitue sa *partie arrière*. Les trois phases de la partie avant maintiennent généralement une *table des symboles*, contenant les noms des identificateurs, les types utilisateurs, les opérateurs surchargés, etc.

Exemple 1.78. Voici quelques-unes des expressions rationnelles utilisées pour l'analyse lexicale du langage C, et la façon de les noter dans un générateur d'analyseurs lexicaux comme `flex` :

mots clefs dont `int` ou `if`, notés

$$E_{\texttt{if}} = \texttt{if}$$
$$E_{\texttt{int}} = \texttt{int}$$

symboles comme `=`, `+` ou `;`

$$E_{=} = \texttt{=}$$
$$E_{+} = \texttt{+}$$
$$E_{;} = \texttt{;}$$

La notation `[...]` représente un ensemble de caractères (parmi un alphabet comme la table ASCII), et `[^...]` l'ensemble de caractères complémentaire. Ces caractères peuvent être échappés, comme `\n` pour un retour à la ligne, et on peut considérer tout un intervalle de la table ASCII, comme `a-f` pour toutes les lettres de `a` à `f`.

blancs les caractères d'espace, de tabulation, de retour à la ligne etc., notés

$$E_{\text{space}} = \texttt{[\textbackslash n\textbackslash r\textbackslash t]+}$$

identifiants des séquences de lettres et de chiffres, notés

$$E_{\text{id}} = \texttt{[\_a-zA-Z][\_0-9a-zA-Z]*}$$

nombres entiers dans plusieurs bases possibles, de type long ou non, signés ou non, notés

$$E_{\text{ic}} = \texttt{([1-9][0-9]*|0[0-7]*|0x[0-9a-fA-F]+)[lLuU]?}$$

chaînes commencent et finissent par `"`, avec des caractères d'échappement, notées

$$E_{\text{sc}} = \texttt{"([\^{}"]|\textbackslash\textbackslash")*"}$$

...

L'idée à partir d'une telle collection d'expressions rationnelles $(E_i)_i$ est de construire un automate fini reconnaissant l'union de leurs langages, et de l'utiliser pour segmenter le texte d'entrée.

Par exemple, l'analyse lexicale d'un texte d'entrée

```
int a = 12;
int b = 3 + a;
```

va retourner une séquence de lexèmes, qui correspondent chacun à une expression rationnelle, et qui constituent un découpage de la chaîne d'entrée :

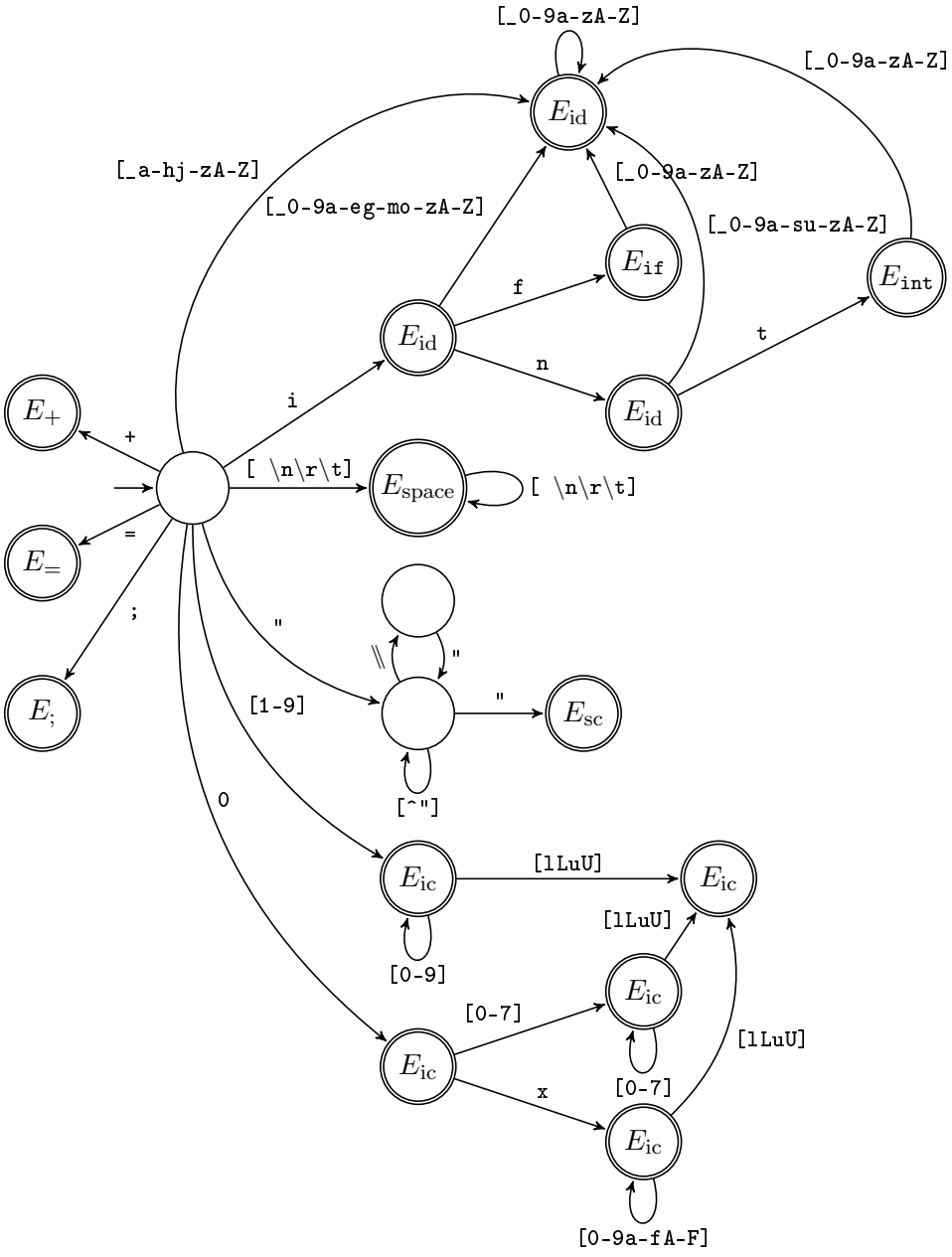
‘`int`’ ‘space’ ‘id’ ‘space’ ‘=’ ‘space’ ‘ic’ ‘ ;’ ‘space’
‘`int`’ ‘space’ ‘id’ ‘space’ ‘=’ ‘space’ ‘ic’ ‘space’ ‘+’ ‘space’ ‘id’ ‘ ;’

On peut observer que ce découpage n'est pas sans problème, puisque

1. `int` est à la fois dans $L(E_{\texttt{int}})$ et $L(E_{\text{id}})$, ce que l'on résout habituellement en donnant la priorité au premier choix,

2. on pourrait aussi diviser `12` en deux lexèmes 'ic', l'un pour `1` et l'autre pour `2`, ce que l'on résout en prenant toujours le segment le plus long possible.

Un automate fini déterministe pour les expressions rationnelles données en exemple serait le suivant. On peut noter qu'il étiquette ses états finaux avec les expressions rationnelles identifiées en ce point, et qu'il incorpore déjà la priorité des mots clefs sur les identifiants.



Algorithme naïf Le principe de l'algorithme naïf d'analyse lexicale est de simuler un automate déterministe $\mathcal{A}$ pour le langage $L(\sum_{i=1}^{p} E_i) = \bigcup_{i=1}^{p} L(E_i)$, comme celui représenté ci-dessus, sur le texte d'entrée w de Σ^*.

Quand il rencontre un état final, l'algorithme ne peut généralement pas immédiatement annoncer avoir trouvé un lexème, puisqu'en lisant plus, il pourrait en trouver un plus long. Il faut donc mémoriser ce dernier état final q_f rencontré et sa position j dans le texte, et continuer la recherche d'un lexème plus long. Quand

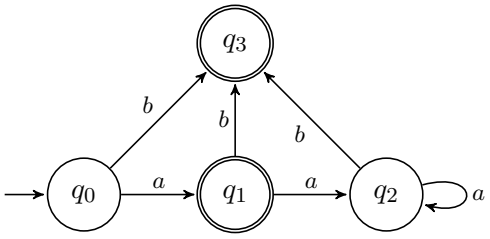
on ne peut appliquer aucune transition de l'automate, on utilise cet état final (par exemple en affichant l'index de l'expression rationnelle qui vient d'être identifiée, donné par une fonction f de F dans $\{1, \ldots, p\}$), et on reprend l'analyse depuis la position j mémorisée.

Algorithme 1.79.
ANALYSE LEXICALE NAÏVE ($\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle, f : F \to \{1, \ldots, p\}, w = a_1 \cdots a_n$)

1. $q \leftarrow q_0$
2. $i \leftarrow 1$
3. **tant que** $i \leq n$ **faire**
4. $\quad q_f \leftarrow \bot$
5. $\quad$ **tant que** $i \leq n \wedge \delta(q, a_i) \neq \emptyset$ **faire**
6. $\quad\quad$ **si** $q \in F$ **alors**
7. $\quad\quad\quad q_f \leftarrow q$
8. $\quad\quad\quad j \leftarrow i$
9. $\quad\quad$ **fin si**
10. $\quad\quad i \leftarrow i + 1$
11. $\quad$ **fin tant que**
12. $\quad$ **si** $q_f = \bot$ **alors**
13. $\quad\quad$ **retourner** échec
14. $\quad$ **fin si**
15. $\quad$ **afficher** $f(q_f)$
16. $\quad i \leftarrow j$
17. **fin tant que**
18. **retourner** succès

Cet algorithme simple souffre cependant d'un défaut : il travaille en temps $O(n^2)$ dans le pire des cas. Comme les fichiers à analyser peuvent être de taille conséquente (par exemple en C par le truchement des `#include`), cette complexité n'est pas acceptable.

Exemple 1.80. Considérons deux expressions rationnelles $E_1 = a$ et $E_2 = a^*b$, pour lesquelles on peut construire l'automate déterministe suivant, associé à la fonction $f : q_1 \mapsto 1, q_3 \mapsto 2$:



Sur le texte d'entrée $w = a^n$, l'algorithme 1.79 va afficher 1^n, mais au prix d'une inspection de la totalité du texte depuis chacune des n positions de départ.

Notons enfin qu'il est nécessaire que ε n'appartienne à aucun langage d'expression E_i si l'on veut garantir la terminaison…

C.f. REPS *[23], et [GBJL00, sec. 2.1.6.7, p. 85].*

Algorithme par programmation dynamique Une solution à cette complexité excessive est d'adopter un algorithme par *programmation dynamique*. L'idée est d'exprimer une solution au problème d'identifier un segment de longueur maximale à partir d'un état q de l'automate depuis une position i dans le mot d'entrée

$w = a_1 \cdots a_n$ à l'aide d'autres calculs pour q' un état de Q et de la position $i+1$. Ces calculs ne dépendent pas du contexte dans lequel ils sont effectués (*transparence référentielle* : une expression peut être remplacée par son résultat), et sont en nombre polynomial $|Q| \cdot (n+1)$:

$$T(q, n+1) = \begin{cases} (\bot, 0) & \text{si } q \notin F \\ (q, n+1) & \text{sinon} \end{cases}$$

et pour $i \leq n$,

$$T(q,i) = \begin{cases} (\bot, 0) & \text{si } \delta(q, a_i) = \emptyset \text{ et } q \notin F \\ (q, i) & \text{si } \delta(q, a_i) = \emptyset \text{ et } q \in F \\ (q, i) & \text{si } \delta(q, a_i) = q',\ T(q', i+1) = (\bot, 0) \text{ et } q \in F \\ T(q', i+1) & \text{si } \delta(q, a_i) = q' \text{ sinon.} \end{cases}$$

On peut calculer $T(Q, \{1, \ldots, n+1\})$ de manière systématique (en considérant successivement des positions décroissantes de w), en temps linéaire $\Theta(|Q| \cdot |w|)$. On reconstruit ensuite la segmentation en lisant $T(q_0, i)$ pour les positions adéquates du texte d'entrée w :

On peut aisément modifier le calcul de T pour travailler sur un automate non déterministe, ce qui fait davantage ressortir le côté « programmation dynamique », puisqu'un calcul de $T(q,i)$ dépend alors de plusieurs calculs $T(q_j, i+1)$ avec $q_j \in \delta(q, a_i)$. De plus, la complexité reste en $O(|\mathcal{A}| \cdot |w|)$, sans avoir à payer le coût de la déterminisation.

Algorithme 1.81.
ANALYSE LEXICALE ($\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle, f : F \to \{1, \ldots, p\}, w = a_1 \cdots a_n$)

1. $i \leftarrow 1$
2. **tant que** $i \leq n$ **faire**
3. $\quad (q_f, i) \leftarrow T(q_0, i)$
4. $\quad$ **si** $q_f = \bot$ **alors**
5. $\quad\quad$ **retourner** échec
6. $\quad$ **fin si**
7. $\quad$ **afficher** $f(q_f)$
8. **fin tant que**
9. **retourner** succès

Une solution plus efficace utilise une approche par *mémoïsation* (ou *recensement*), et consiste à ne calculer que les valeurs utiles de $T(Q, \{1, \ldots, n+1\})$. Cette version paresseuse travaille alors en temps $O(|Q| \cdot |w|)$.

Exemple 1.82. Reprenons l'analyse de $w = a^n$ de l'exemple 1.80 : on a

$$T(q_0, 1) = T(q_1, 2) = (q_1, 2)$$

puisque

$$T(q_2, 3) = T(q_2, 4) = \cdots = T(q_2, n+1) = (\bot, 0)\ .$$

Une fois ce premier segment trouvé en temps $O(n)$, on peut relancer l'analyse lexicale avec $T(q_0, 2) = T(q_1, 3)$, pour lequel on pourra directement réutiliser la solution du calcul de $T(q_2, 4)$: les étapes ultérieures sont en $O(1)$ dans cet exemple.

1.4.3 Linguistique

Syntaxe Puisque les langages rationnels permettent une description finie d'un ensemble infini, on pourrait espérer représenter la totalité des phrases d'une langue

comme le français, en supposant tout de même un lexique fini et fixe. Cette idée d'employer des règles de dérivation formelles a été notamment explorée par Noam CHOMSKY [9], et l'a amené à définir les différentes classes de grammaires qui composent maintenant la hiérarchie de CHOMSKY. Les grammaires de type 3 y sont équivalentes aux langages rationnels (voir la **??**). Cependant les langages rationnels sont inappropriés pour décrire la totalité d'une langue : en traduisant un exemple de CHOMSKY, un prédicat de la forme

$$\text{l'homme qui dit que } S \text{ arrive demain}$$

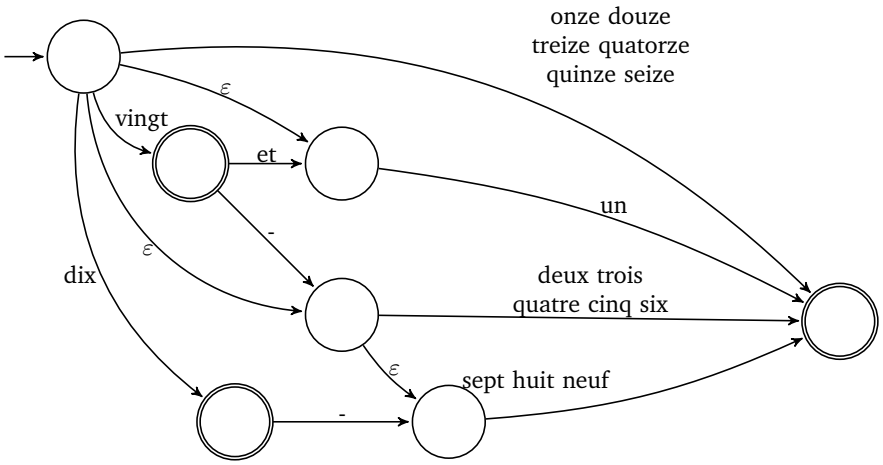
peut avoir n'importe quel prédicat à la place de S, y compris un prédicat de cette même forme. On arrive donc à un ensemble de phrases de la forme

$$(\text{l'homme qui dit que})^n S (\text{arrive demain})^n, n \geq 0$$

sur $\Sigma = \{$l'homme, qui, dit, que, arrive, demain$\}$. Un langage qui contient un tel sous-ensemble n'est pas rationnel – voici une application un peu originale des propriétés de clôture et des lemmes d'itération.

À défaut de modéliser l'intégralité du français, on peut néanmoins utiliser (avec succès) des automates finis et des expressions régulières pour des sous-langages : dates, nombres, etc.

Exemple 1.83. Voici un automate qui reconnaît les nombres de « un » à « vingt-neuf » :



Une analyse syntaxique simple, dite « de surface », peut aussi être effectuée sur des phrases entières, à l'aide de transducteurs ou de cascades de transducteurs, qui vont repérer les ensembles de mots qui forment des groupes nominaux ou des groupes verbaux par exemple. L'intérêt des automates réside alors dans leur bonne complexité d'analyse comparée à des formalismes plus complexes comme les grammaires algébriques, fournissant des analyses suffisantes pour certaines applications à moindre coût.

Morphologie Les descriptions rationnelles sont aussi très bien adaptées aux phases de découpe d'un texte en unités simples : mots, signes de ponctuation, dates, noms propres, etc. C'est une utilisation similaire à celle faite en analyse lexicale des langages de programmation.

Chapitre 2

Références supplémentaires

Ces références supplémentaires contiennent des articles classiques du domaine, malheureusement souvent difficiles à obtenir, et des articles ou des livres plus récents dont la lecture complète celle de la bibliographie recommandée.

[1] Alfred V. AHO, 1990. Algorithms for finding patterns in strings. Dans Jan VAN LEEUWEN, éditeur, *Handbook of Theoretical Computer Science*, volume A, chapitre 5, pages 256–300. Elsevier.

[2] Valentin ANTIMIROV, 1996. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319. doi: 10.1016/0304-3975(95)00182-4.

[3] Danièle BEAUQUIER, Jean BERSTEL, et Philippe CHRÉTIENNE, 1992. *Eléments d'algorithmique*. Masson. URL `http://www-igm.univ-mlv.fr/~berstel/Elements/`.

[4] Jean BERSTEL, 1979. *Transductions and Context-Free Languages*. Teubner Studienbücher: Informatik. Teubner. ISBN 3-519-02340-7. URL `http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/`.

[5] Jean BERSTEL et Jean-Éric PIN, 1996. Local languages and the Berry-Sethi algorithm. *Theoretical Computer Science*, 155(2):439–446. doi: 10.1016/0304-3975(95)00104-2.

[6] Janusz A. BRZOZOWSKI, 1963. Canonical regular expressions and minimal state graphs for definite events. Dans *Symposium on the Mathematical Theory of Automata*, pages 529–561.

[7] Janusz A. BRZOZOWSKI, 1964. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494. doi: 10.1145/321239.321249.

[8] Janusz A. BRZOZOWSKI et Edward J. MCCLUSKEY, 1963. Signal flow graph techniques for sequential circuit state diagrams. *IRE Transactions on Electronic Computers*, 12:67–76. doi: 10.1109/PGEC.1963.263415.

[9] Noam CHOMSKY, 1956. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3):113–124. doi: 10.1109/TIT.1956.1056813.

[10] Maxime CROCHEMORE et Christophe HANCART, 1997. Automata for matching patterns. Dans Grzegorz ROZENBERG et Arto SALOMAA, éditeurs, *Handbook of Formal Languages*, volume 2. Linear Modeling : Background and Application, chapitre 9, pages 399–462. Springer. ISBN 3-540-60648-3.

[11] Andrzej EHRENFEUCHT et Paul ZEIGER, 1976. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146. doi: 10.1016/S0022-0000(76)80034-7.

[12] Andrzej EHRENFEUCHT, Rohit PARIKH, et Grzegorz ROZENBERG, 1981. Pumping lemmas for regular sets. *SIAM Journal on Computing*, 10(3):536–541. doi: 10.1137/0210039.

[13] Calvin C. ELGOT et Jorge E. MEZEI, 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68. URL `http://www.research.ibm.com/journal/rd/091/ibmrd0901E.pdf`.

[14] Shimon EVEN, 1965. On information lossless automata of finite order. *IEEE Transactions on Electronic Computers*, EC-14(4):561–569. doi: 10.1109/PGEC.1965.263996.

[15] V. M. GLUSHKOV, 1961. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1–53. doi: 10.1070/RM1961v016n05ABEH004112.

[16] Juraj HROMKOVIČ, Sebastian SEIBERT, et Thomas WILKE, 1997. Translating regular expressions into small ε-free nondeterministic finite automata. Dans R. REISCHUK, éditeur, *14th International Symposium on Theoretical Aspects of Computer Science (STACS'97)*, volume 1200 de *Lecture Notes in Computer Science*, pages 55–66. Springer. ISBN 3-540-62616-6. doi: 10.1007/BFb0023448.

[17] Tao JIANG et Bala RAVIKUMAR, 1993. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141. doi: 10.1137/0222067.

[18] Stephen C. KLEENE, 1956. Representation of events in nerve nets and finite automata. Dans C. E. SHANNON et J. MCCARTHY, éditeurs, *Automata Studies*, pages 3–40. Princeton University Press.

[19] Donald E. KNUTH, James H. MORRIS, Jr., et Vaughan R. PRATT, 1977. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350. doi: 10.1137/0206024.

[20] Robert MCNAUGHTON et H. YAMADA, 1960. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, 9(1):39–47.

[21] Christos PAPADIMITRIOU, 1993. *Computational Complexity*. Addison-Wesley. ISBN 0-201-53082-1.

[22] Michael O. RABIN et Dana SCOTT, 1959. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125. URL `http://www.research.ibm.com/journal/rd/032/ibmrd0302C.pdf`.

[23] Thomas REPS, 1998. "Maximal-munch" tokenization in linear time. *ACM Transactions on Programming Languages and Systems*, 20(2):259–273. doi: 10.1145/276393.276394.

[24] Imre SIMON, 1994. String matching algorithms and automata. Dans Juliani KARHUMÄKI, Hermann MAURER, et Grzegorz ROZENBERG, éditeurs, *Results and Trends in Theoretical Computer Science: Colloquium in Honor of Arto Salomaa*, volume 812 de *Lecture Notes in Computer Science*, pages 386–395. Springer. ISBN 978-3-540-58131-4. doi: 10.1007/3-540-58131-6_61.

[25] R. E. STEARNS et H. B. HUNT III, 1985. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611. doi: 10.1137/0214044.

[26] Larry J. STOCKMEYER et Albert R. MEYER, 1973. Word problems requiring exponential time (preliminary report). Dans *Fifth Symposium on Theory of Computing (STOC '73)*, pages 1–9. ACM Press. doi: 10.1145/800125.804029.

[27] Robert E. TARJAN, 1972. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160. doi: 10.1137/0201010.

[28] Ken THOMPSON, 1968. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422. doi: 10.1145/363347.363387.

[29] Qiqi YAN, 2008. Lower bounds for complementation of ω-automata via the full automata technique. *Logical Methods in Computer Science*, 4(1):5. doi: 10.2168/LMCS-4(1:5)2008.

[30] Sheng YU, 1997. Regular languages. Dans Grzegorz ROZENBERG et Arto SALOMAA, éditeurs, *Handbook of Formal Languages*, volume 1. Word, Language, Grammar, chapitre 2, pages 41–110. Springer. ISBN 978-3-540-60420-4.