UNIVERSITÉ DE VALENCIENNES ET DU HAINAUT-CAMBRÉSIS

INSTITUT DES SCIENCES ET TECHNIQUES

LICENCE D'INFORMATIQUE

Théorie des langages : les automates à états finis

version 1.3

Par : Sylvain Piechowiak

15 Septembre 2020

Résumé

L'objectif de ce cours est de présenter les bases de la théorie des langages. Il ne s'agit pas de réaliser une *bible sur les automates* mais un support pour le cours dispensé pendant une durée de 6h. Nous nous intéressons aux langages réguliers et aux automates à états finis. Les passages entre expressions régulières et automates sont abordés. Le lemme de l'étoile et le théorème de Kleene sont 2 résultats importants de ce cours. Ce cours est suivi d'un autre cours qui concerne les langages algébriques qui ne sont donc pas étudiés ici.

Je tiens à préciser que je me suis largement inspiré de documents rédigés par d'autres collègues que je tiens à remercier ici.

Table des matières

xx44 yy46

1 Alphabet, mot et langage

1.1 Alphabet

Définition 1 (Alphabet)**.** *On appelle alphabet (ou ensemble de caractères) tout ensemble fini Σ non vide*

Dans nos applications, l'ensemble Σ sera presque toujours l'ensemble des caractères accessibles sur un clavier d'ordinateur (lettres minuscules, lettres majuscules, chiffres, symboles et caractères spéciaux comme l'espace ou le retour à la ligne), mais rien n'empêche d'imaginer d'autres ensembles tels que des ensembles de formes, de couleurs, d'odeurs etc.

Exemple 1. *Voici des exemples d'alphabets :*
- Σ_1 = { ♢, ♣, ♡, ♠ }
- Σ_2 = { *a, b, c, ..., z* }
- Σ_3 = { *if, then, else, id, nb, =, +, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, ...* } *On remarque que certains caractères comportent plusieurs symboles.*

1.2 Mots

Définition 2 (Mot)**.** *On appelle mot sur Σ toute suite finie (éventuellement vide) de caractères pris dans l'alphabet Σ. La longueur d'un mot est le nombre de ses caractères. L'ensemble des mots sur Σ sera noté Σ^*.*

La longueur du mot m est notée $|m|$. Le nombre d'occurrences du caractère x dans le mot m est noté $|m|_x$.

Le mot de longueur nulle est noté ε : $|\varepsilon| = 0$. Il es également appelé mot vide.

On notera $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ ou bien $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

Exemple 2. *Voici des exemples de mots :*
- ♢♣♡♠ , ♢♣♠♡♠♣♡♢♡ *ou* ♢♠♣♠♡♡♡♠♣♡♢♡♢♠♠ *sont des mots sur l'alphabet Σ_1*
- *abbcd , azeucbgerdydhsb ou tyes sont des mots sur l'alphabet Σ_2*
- $if\ x < 3 + 2\ then\ y := 6\ else\ u := v + w$
  ou $tant\ que\ x < 5\ do\ x := x + 1$ *sont des mots sur l'alphabet Σ_3*

Définition 3 (Concaténation)**.** *Soient $X = x_1x_2...x_{|X|}$ et $Y = y_1y_2...x_{|Y|}$ deux mots construits sur les alphabets Σ_1 et Σ_2. On appelle concaténation (ou produit) de X et Y (et notée $X.Y$) le mot : $x_1x_2...x_{|X|}y_1y_2...x_{|Y|}$*

On a : $|m| = |m_1.m_2| = |m_1| + |m_2|$ et trivialement : $|m|_x = |m_1.m_2|_x = |m_1|_x + |m_2|_x$ et :

$$|m| = \sum_{x \in \Sigma} |m|_x$$

La concaténation de deux mots n'est pas une opération commutative. Par exemple, si on prend les deux mots $m_1 = a$ et $m_2 = b$ on a $m_1.m_2 = ab \neq ba = m_2.m_1$. Par contre, la concaténation est une opération associative : $(m_1.m_2).m_3 = m_1.(m_2.m_3) = m_1.m_2.m_3$

On désigne l'ensemble de tous les mots de longueur n construits sur l'alphabet Σ par Σ^n. On a :

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

Définition 4 (Itération)**.** *On appelle itération sur le mot m, l'ensemble { ε, m, $m.m$, $m.m.m$, ...} noté m^*. Et on désigne par m^+ l'ensemble { m, $m.m$, $m.m.m$, ...}*

$m^0 = \varepsilon$ et on notera $m.m.m.m... = mmmmm... = m^n$

1.3 Langages

Définition 5 (Langage)**.** *On appelle langage sur Σ toute partie $\mathcal{L}$ de Σ^*.*

Un langage est donc un ensemble fini ou infini de mots. L'ensemble vide de mots est un langage particulier noté { } ou $\emptyset$.
Un langage $\mathcal{L}$ étant une partie de Σ^*, la difficulté est d'être capable de distinguer les mots qui font partie de $\mathcal{L}$ parmi tous les mots qu'on peut construire sur Σ. Si le langage est fini, la méthode la plus simple pour le décrire est de donner la liste de tous ses mots. Si le langage comporte un nombre infini de mots, on peut les décrire soit en composant des langages plus simples, soit en fournissant les règles d'écriture des mots (ce qu'on appelle grammaires). Il existe également des langages infinis qu'on peut décrire mais pour lesquels on ne sait pas toujours dire si un mot fait partie du langage.

Exemple 3. *Voici des exemples de langages :*
- *$\mathcal{L}_1$ = $\{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, baa, bba, bab, abb, bbb\}$ langage construit sur $\Sigma = \{a, b\}$ dont les mots sont de taille ≤ 3. $\mathcal{L}_1 = \{m \in \Sigma^*, |m| \leq 3\}$*
- *$\mathcal{L}_2$ = ensemble des programmes qui s'arrêtent (qui ne bouclent pas indéfiniment). On ne peut pas décrire formellement $\mathcal{L}_2$. C'est ce qu'on appelle le problème de l'arrêt des programmes.*

Pour le moment la seule façon dont nous disposons pour décrire les mots d'un langage est d'utiliser la langue française.

Exemple 4. *Σ étant l'ensemble des caractères accessibles sur le clavier de l'ordinateur, voici des exemples de langages :*
- *les représentations décimales de nombres entiers : suites de caractères commençant éventuellement par un symbole $+$ ou $-$ suivi uniquement de chiffres de 0 à 9*
- *les représentations binaires des entiers : toute suite finie de 0 et de 1*
- *les représentations de nombres décimaux : suites de caractères commençant éventuellement par un symbole + ou - suivi de chiffres de 0 à 9, puis d'un point décimal (pour les anglo-saxons), puis de chiffres de 0 à 9*
- *les mots composés uniquement de caractères alphabétiques commençant par un i, terminés par un e et comportant les caractères consécutifs m, a*
- *les mots ne comportant que des caractères alphabétiques et ayant plus de a que de e et plus de e que de i*

Exemple 5. *Soit l'alphabet $\Sigma = \{a, b\}$.*
- *$\mathcal{L}_2$ = {mots sur Σ^* qui contiennent autant de a que de b } = $\{m \in \Sigma^*, |m|_a = |m|_b\}$*
- *$\mathcal{L}_3$ = { mots sur Σ^* de longueur impaire } = $\{m \in v^*, |m| = 1$ modulo $2\} = \Sigma(\Sigma^2)^*$*
- *$\mathcal{L}_4$ = $\{m \in \Sigma^*, m = a^p b^q, p \geqslant 0, q \geqslant 0\} = a^* b^*\}$*
- *$\mathcal{L}_5$ = $\{m \in \Sigma^*, m = a^p b^p, p \geqslant 0\}$*
- *$\mathcal{L}_2 \cap \mathcal{L}_3 = \emptyset$*
- *$\mathcal{L}_2 \cap \mathcal{L}_4 = \mathcal{L}_5$*

L'objectif de ce cours est de décrire des moyens formels pour décrire (des règles d'écriture des mots) et manipuler des langages. Ces moyens doivent nous permettre de construire les mots d'un langage, reconnaitre les mots qui font partie d'un langage, etc. Il existe différentes classes de langages qu'on peut ou non décrire et manipuler avec différents outils. Chomsky a classé les différents langages selon ce qu'on pourrait appeler leur expressivité. Ici nous nous focaliserons sur les langages de premier niveau selon cette hiérarchie de Chomsky.

1.4 Opérations sur les langages

Les langages sont des ensembles. Donc tout ce qui est connu sur la théorie des ensembles s'applique sur les langages. En particulier, les opérations sur les ensembles (intersection, union, produit, différence) ont un sens sur les langages.

Définition 6 (Opérateurs ensemblistes sur les langages)**.** *Soit $\mathcal{L}_1$ et $\mathcal{L}_2$ deux langages construits sur les alphabets Σ_1 et Σ_2. On définit les opérateurs ensenblistes sur $\mathcal{L}_1$ et $\mathcal{L}_2$:*

- $\mathcal{L}_1 \cup \mathcal{L}_2 = \{m \in \mathcal{L}_1\} \cup \{m \in \mathcal{L}_2\} = \{m \in \mathcal{L}_1 \cup \mathcal{L}_2\}$
- $\mathcal{L}_1 \cap \mathcal{L}_2 = \{m \in \mathcal{L}_1\} \cap \{m \in \mathcal{L}_2\} = \{m \in \mathcal{L}_1 \cap \mathcal{L}_2\}$
- $\mathcal{L}_1 \times \mathcal{L}_2 = \{m \in \mathcal{L}_1\} \times \{m \in \mathcal{L}_2\} = \{(m_1, m_2)/m_1 \in \mathcal{L}_1 \wedge m_2 \in \mathcal{L}_2\}$
- $\mathcal{L}_1 - \mathcal{L}_2 = \{m \in \mathcal{L}_1\} - \{m \in \mathcal{L}_2\} = \{m \in \mathcal{L}_1 \wedge m \notin \mathcal{L}_2\}$

Définition 7 (Concaténation)**.** *Soit $\mathcal{L}_1$ et $\mathcal{L}_2$ deux langages construits sur les alphabets Σ_1 et Σ_2. On appelle concaténation de $\mathcal{L}_1$ et $\mathcal{L}_2$, noté $\mathcal{L}_1.\mathcal{L}_2$, l'ensemble des mots $\{m = m_1.m_2/m_1 \in \mathcal{L}_1 \wedge m_2 \in \mathcal{L}_2\}$.*

Attention : en général $\mathcal{L}_1.\mathcal{L}_2 \neq \mathcal{L}_2.\mathcal{L}_1$ Pour s'en convaincre il suffit de prendre $\mathcal{L}_1 = \{a\}$ et $\mathcal{L}_2 = \{b\}$. On a $\mathcal{L}_1.\mathcal{L}_2 = \{ab\}$ et $\mathcal{L}_2.\mathcal{L}_1 = \{ba\}$

Définition 8 (Puissance)**.** *soit $\mathcal{L}$ un langage construit sur l'alphabet Σ. On appelle puissance d'un langage l'ensemble $\mathcal{L}^n = \mathcal{L}.\mathcal{L}.\mathcal{L}...\mathcal{L}$ obtenu en concaténant n fois le langage $\mathcal{L}$.*

On a $\mathcal{L} = \mathcal{L}.\mathcal{L}^{n-1} = \mathcal{L}^{n-1}.\mathcal{L}$

Exemple 6. *Soit l'alphabet $\Sigma = \{a, b\}$ et $\mathcal{L} = \{aa, ab, bb\}$ (les mots de longueur 2 sur Σ).*

- $\mathcal{L}^0 = \emptyset$
- $\mathcal{L}^1 = \mathcal{L} =$ *{ aa, ab, bb }*
- $\mathcal{L}^2 = \mathcal{L}.\mathcal{L} =$ *{ aa, ab, bb } . { aa, ab, bb } = { aaaa, aaab, aabb, abaa, abab, abbb, bbaa, bbab, bbbb }*
- $\mathcal{L}^3 = \mathcal{L}.\mathcal{L}.\mathcal{L} =$ *{ aa, ab, bb } . { aa, ab, bb } . { aa, ab, bb } = { aaaaaa, aaaaab, aaaabb, aaabaa, aaabab, aaabbb, aabbaa, aabbab, aabbbb, abaaaa, abaaab, abaabb, ababaa, ababab, ababbb, abbbaa, abbbab, abbbbb, bbaaaa, bbaaab, bbaabb, bbabaa, bbabab, bbabbb, bbbbaa, bbbbab, bbbbbb }*
  On a aussi $\mathcal{L}^3 = \mathcal{L}.\mathcal{L}^2 =$ { aa, ab, bb } . { aaaa, aaab, aabb, abaa, abab, abbb, bbaa, bbab, bbbb }
  Et aussi $\mathcal{L}^3 = \mathcal{L}^2.\mathcal{L} =$ { aaaa, aaab, aabb, abaa, abab, abbb, bbaa, bbab, bbbb } . { aa, ab, bb }

Définition 9 (Itération)**.** *On appelle itération sur le langage $\mathcal{L}$, l'ensemble $\mathcal{L}^* = \bigcup_{n \geqslant 0}^{\infty} \mathcal{L}^n$. Et on désigne par $\mathcal{L}^+$ l'ensemble $\bigcup_{n \geqslant 1}^{\infty} \mathcal{L}^n$. L'opérateur $*$ est également appelé itération de Kleene.*

$\mathcal{L}^* = \mathcal{L}^0 \cup \mathcal{L}^1 \cup \mathcal{L}^2 \cup ... = \mathcal{L}^0 \cup \bigcup_{n \geqslant 1}^{\infty} (\mathcal{L}^n) = \mathcal{L}^0 \cup \mathcal{L}^+ = \{\varepsilon\} \cup \mathcal{L}^+$

Proposition 1. *Voici quelques propriétés :*

- $\mathcal{L}_1 \subseteq \mathcal{L}_2 \Rightarrow \mathcal{L}_1^* \subseteq \mathcal{L}_2^*$
- $\mathcal{L}^* = (\mathcal{L}^*)^*$
- $\mathcal{L}^+ = \mathcal{L}\mathcal{L}^* = \mathcal{L}^*\mathcal{L}$

2 Langages réguliers

2.1 Expressions régulières

Définition 10 (Opération régulière)**.** *On appelle opérations régulières les trois opérations suivantes sur les langages :*
- *la réunion finie de langages* ($\cup$)
- *la concaténation* ($.$)
- *l'itération* ($*$)

Définition 11 (Langage régulier)**.** *Un langage est dit régulier ssi on peut le construire, à partir de langages finis, par un nombre fini d'applications d'opérations régulières.*

Définition 12 (Expression régulière)**.** *Soit un alphabet* Σ*. On définit comme expression régulière toute expression construite en utilisant les symboles* $\emptyset$ *(ou { }), les caractères* $x \in \Sigma$*,* +*, .,* *, (et) et telle que :*
- $\forall x \in \Sigma$*,* x *est une expression régulière*
- $\emptyset$ *est une expression régulière*
- *Si* e_1 *et* e_2 *sont des expressions régulières, alors les expressions* $(e_1 + e_2)$*,* $(e_1.e_2)$ *et* e_1^* *sont également des expressions régulières.*

Il s'agit maintenant de faire le lien entre un langage régulier et une expression régulière. Ce lien consiste à associer chaque expression régulière à un ensemble de mots (on dira que l'expression désigne, dénote ou interprète un langage).

Définition 13 (Interprétation)**.** *Désignons par* $\mathcal{E}(\Sigma)$ *l'ensemble des expressions régulières sur l'alphabet* Σ *et par* $\mathcal{P}(\Sigma^*)$ *l'ensembles des parties de* Σ^**. L'interprétation d'une expression régulière est l'application* $\mathcal{I}$ *définie de* $\mathcal{E}$ *dans* $\mathcal{P}(\Sigma^*)$ *par :*
- $\forall x \in \Sigma, \mathcal{I}(x) = x$ *(langage réduit au seul mot d'un seul caractère pris dans* Σ*)*
- $\mathcal{I}(\emptyset) = \emptyset$ *(langage vide)*
- $\mathcal{I}(e_1 + e_2) = \mathcal{I}(e_1) \cup \mathcal{I}(e_2)$ *(union de langages)*
- $\mathcal{I}(e_1.e_2) = \mathcal{I}(e_1).\mathcal{I}(e_2)$ *(concaténation de langages)*
- $\mathcal{I}(e^*) = (\mathcal{I}(e))^*$ *(itération d'un langage)*

Remarque : il est fréquent d'utiliser le symbole + au lieu du symbole $\cup$: c'est ce qui sera fait dans ce cours.

2.2 Langages réguliers

Définition 14 (Langage régulier)**.** $\mathcal{L} \subseteq \Sigma^*$ *est un langage régulier sur* Σ *ssi il existe une expression régulière* $\alpha \in \mathcal{E}(\Sigma)$ *telle que :* $\mathcal{L} = \mathcal{I}(\alpha)$*.*

On dira alors que α est une expression régulière de $\mathcal{L}$, ou que α *dénote* $\mathcal{L}$. Remarque : il peut y avoir plusieurs expressions régulières qui dénotent un même langage. La difficulté est de reconnaitre que deux expressions différentes dénotent un même langage (on dira que ces expressions sont équivalentes).

Proposition 2. *Propriété : L'ensemble* $Reg(\Sigma)$ *des langages réguliers sur* Σ *est le plus petit ensemble qui vérifie :*
- $\forall x \in \Sigma, x \in Reg(\Sigma)$
- $\emptyset \in Reg(\Sigma)$
- $\mathcal{L}_1 \in Reg(\Sigma) \wedge \mathcal{L}_2 \in Reg(\Sigma) \Rightarrow \mathcal{L}_1 + \mathcal{L}_2 \in Reg(\Sigma)$
- $\mathcal{L}_1 \in Reg(\Sigma) \wedge \mathcal{L}_2 \in Reg(\Sigma) \Rightarrow \mathcal{L}_1.\mathcal{L}_2 \in Reg(\Sigma)$
- $\mathcal{L} \in Reg(\Sigma) \Rightarrow \mathcal{L}^* \in Reg(\Sigma)$

Exemple 7. *Voici quelques exemples de langages réguliers :*
- *Le langage vide $\emptyset \subseteq \Sigma^*$ est régulier.*
- *Le langage $\{\varepsilon\} \subseteq \Sigma^*$, interprétation de l'expression régulière $\emptyset^*$ est régulier. On utilise souvent ε pour désigner l'expression régulière $\emptyset^*$ elle–même.*
- *Un langage réduit à un seul mot est régulier. En particulier $\forall x \in \Sigma$, $\{x\}$ est un langage régulier.*
- *Tout langage fini est la réunion d'une famille finie de langages à un seul mot : c'est pourquoi un langage fini est régulier.*
- *Rappelons que nous ne considérons que des alphabets Σ finis : ceci sert à assurer que $\Sigma \subseteq \Sigma^*$ lui-même est un langage régulier.*
- *En appliquant la clôture de $Reg(\Sigma)$ par itération, on déduit du point précédent que $\Sigma^* \subseteq \Sigma$ est régulier.*
- *Pour toute expression régulière α et tout entier n, on peut définir α^n par récurrence sur n : le résultat est une façon d'écrire une expression régulière.*

Exemple 8. *On peut évidemment donner une expression régulière de langages plus particuliers que ceux qui précèdent. Par exemple, pour l'alphabet $\Sigma = \{a, b\}$ comportant deux symboles distincts :*
- *$\emptyset$, ε, a, b, $\Sigma = a + b$ et $\Sigma^* = (a + b)^*$ désignent des langages réguliers*
- *$\Sigma^n = (a + b)^n$ est une expression régulière de l'ensemble des mots de longueur n,*
- *$(\varepsilon + \Sigma)^n = (\varepsilon + a + b)^n$ est une expression régulière de l'ensemble des mots de longueur inférieure ou égale à n,*
- *$\Sigma^m(\varepsilon + \Sigma)^n = (a + b)^m(\varepsilon + a + b)^n$ est une expression régulière de l'ensemble des mots dont la longueur est comprise entre m et $m + n$,*
- *$((\varepsilon + a)b)^*(\varepsilon + a)$ est une expression régulière du langage formé des mots qui ne comportent pas le facteur aa.*

3 Automates finis déterministes

En théorie des langages, il y a un lien fort entre les langages et les *reconnaisseurs* de ces langages. Un *reconnaisseur* de langage est un programme qui prend un mot en entrée et répond si oui ou nom ce mot appartient au langage. Le formalisme que nous choisissons pour décrire ces *reconnaisseurs* sont les **automates**.

3.1 Automates finis

Soit Q l'ensemble des états possibles de la mémoire d'un ordinateur : cette mémoire étant finie, l'ensemble des états possibles est également fini. L'application $\delta : Q \times \Sigma \longrightarrow Q$ appelée fonction de transition qui à un état p et un caractère c de l'alphabet Σ associe $q = \delta(p, c)$ qui est le nouvel état après la lecture du caractère c. Avant la lecture du premier caractère du mot à reconnaître, notre machine sera dans un état connu $p_0 \in Q$ appelé état de départ. Après la lecture du dernier caractère, la mémoire de l'ordinateur sera dans un état q qui doit déterminer si le mot appartient ou non au langage. Certains états seront donc privilégiés : ce sont les états *finaux*, *terminaux* ou *acceptants*. Modélisation suivante du travail de reconnaissance (linéaire, c'est à dire sans retour en arrière) d'un mot par un ordinateur :

Définition 15 (Automate)**.** *On appelle automate fini $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ (ou plus simplement automate) la donnée :*
- *d'un alphabet Σ*
- *d'un ensemble fini (et non vide) Q appelé ensemble des états de l'automate*
- *d'un élément $p_0 \in Q$ appelé l'état initial ou état de départ*
- *d'une partie $F \subseteq Q$ appelée l'ensemble des états finaux ou états acceptants*
- *d'une application $\delta : Q \times \Sigma \longrightarrow Q$ appelée fonction de transition de l'automate*

Un automate peut avoir plusieurs états initiaux qui forment l'ensemble I. La fonction de transition est souvent donnée sous la forme d'un ensemble de transitions élémentaires (p, x, q) où p et q sont des états de Q et x est un caractère de l'alphabet Σ. En définitive un automate est donc un quintuplet : $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$.

Définition 16 (Etat puits)**.** *Soit $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ un automate. On appelle rebut ou puits tout état non final $r \in Q$ tel que $\forall a \in \Sigma, \delta(r, a) = r$.*

En clair : lorsque qu'on « tombe » dans le puits, on n'en sort jamais !

Il peut arriver que lorsque la machine est dans un état p, on ne veuille pas accepter un caractère donné c. Cela revient à admettre que la fonction de transition n'est pas définie sur $Q \times \Sigma$ tout entier, mais simplement sur une partie $\Delta \subseteq Q \times \Sigma$.

Définition 17 (Automate complet)**.** *Soit $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ un automate. $\mathcal{A}$ est dit complet ssi $\forall p \in Q, \forall a \in \Sigma, \exists q \in Q \wedge (p, a, q) \in \delta$.*

Un automate est donc complet si, de chacun de ses états, il est possible de sortir en lisant chaque lettre de l'alphabet.

Pour chaque automate $\mathcal{A}$, on peut construire un automate complet $\mathcal{B} = \langle \Sigma, Q', I, F, \delta' \rangle$ équivalent (c'est à dire qui reconnaîtra exactement le même langage) de la manière suivante :

- on crée un état puits $puits$ qu'on ajoute à Q : $Q' = Q \cup \{puits\}$
- on prolonge $\delta : \Delta \longrightarrow Q$ en $\delta' : Q' \times \Sigma \longrightarrow Q$ de la manière suivante :
  - $\delta'(p, a) = \delta(p, a)$ si $(p, a) \in \Delta$
  - $\delta'(p, a) = puits$ si $(p, a) \notin \Delta$ (on envoie au rebut les valeurs non définies)
  - $\delta'(puits, a) = puits$ pour tout $a \in \Sigma$ (quand on est au rebut, on n'en bouge plus)
- On conserve les états initiaux I et les états finaux F (le puits n'est donc pas ni un état initial ni un état final).

On obtient ainsi un nouvel automate dont la fonction de transition est définie partout et qui effectuera le même travail que l'automate initial, c'est à dire que ce nouvel automate reconnaitra exactement les mêmes mots.

3.2 Processus de reconnaissance par un automate

On souhaite soumettre un mot m à un automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ et savoir si cet automate reconnait le mot m. Initialement, notre automate est dans un état $p_0 \in I$. Avant la lecture du $n^{ième}$ caractère, c'est à dire lorsqu'il a lu les $n-1$ premiers caractères c_1,...,c_{n-1} ou encore lorsqu'il a lu le mot c_1,...,c_{n-1}, l'automate est dans l'état p. Après la lecture du caractère c_n, c'est à dire lorsqu'il a lu le mot c_1,...,c_n, il passe dans l'état $q = \delta(p, c_n)$. A tout mot $w = c_1...c_n$ est associé un état q qui est l'état de l'automate partant de l'état initial p_0 une fois qu'il a lu les caractères c_1,...,c_n. Nous noterons cet état $\delta(p_0, w)$. Le processus de reconnaissance d'un mot w est simplement le test pour savoir si $\delta(p_0, w)$ appartient ou non à l'ensemble F des états finaux.

Ce que nous venons de faire en partant de l'état initial p_0 peut naturellement être étendu en partant d'un autre état p : si on suppose qu'à un instant donné l'automate est dans un état p et que l'on procède ensuite à la lecture d'un mot $w = c_1...c_n$, l'automate passera alors dans un état $q = \delta(p, w)$. Ceci revient à définir par récurrence une fonction $\delta : Q \times \Sigma^* \longrightarrow Q$ de la manière suivante :

- $\forall p \in \Sigma, \delta(p, \varepsilon) = p$ (la lecture du mot vide de modifie pas l'état de l'automate)
- si $n \geqslant 1, \delta(p, c_1...c_n) = \delta(\delta(p, c_1...c_{n_1}), c_n)$ (en partant de l'état p, pour lire le mot $c_1...c_n$ on lit d'abord le mot $c_1...c_{n-1}$ pour se retrouver dans l'état $\delta(p, c_1...c_{n_1})$, puis le caractère c_n fait passer dans l'état $\delta(\delta(p, c_1...c_{n_1}), c_n)$

Définition 18. *On appelle langage reconnu par l'automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ l'ensemble des mots wßΣ^* tels que $\delta(p_0, w)$ est un état final.*

3.3 Diagramme d'un automate

Etant donné un automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$, il est d'usage d'adopter une représentation dynamique des transitions de l'automate. C'est ainsi que la notation $p \overset{a}{\longmapsto} q$ dénotera l'égalité $q = \delta(p, a)$: la lecture du caractère $a \in \Sigma$ fait passer de l'état p à l'état q. On peut même représenter tout l'automate par un diagramme sur lequel figureront tous les états de la machine (il est d'usage de les placer à l'intérieur de cercles) ; ces états seront reliés par des flèches étiquetées par les éléments de l'alphabet comme ci dessus, la relation $q = \delta(p, a)$ étant symbolisée par le sous-diagramme de la figure 1.

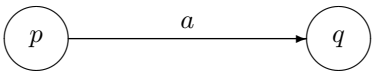


FIGURE 1 – Représentation d'une transition

L'état initial est repéré par une flèche entrante provenant de l'extérieur et les états finaux sont distingués par un double cerclage ou par une flèche sortante comme dans les figures 2 et 3.

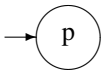


FIGURE 2 – Représentation d'un état initial

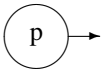


FIGURE 3 – Représentation d'un état final

Exemple 9. *Considérons par exemple l'automate* $\mathcal{A}_1 = \langle \Sigma, Q, I, F, \delta \rangle$ *avec :*

- $\Sigma = \{0, 1\}$
- $Q = \{p_0, p_1, p_2, p_3, p_4\}$ *où* p_4 *est un puits*
- *l'état initial est* $I = \{p_0\}$
- *le seul état final est* $F = \{p_3\}$
- *la fonction de transition est définie par :*

$$\begin{array}{ll} \delta(p_0, 0) = \{p_1\} & \delta(p_0, 1) = \{p_3\} \\ \delta(p_1, 0) = \{p_4\} & \delta(p_1, 1) = \{p_2\} \\ \delta(p_2, 0) = \{p_0\} & \delta(p_2, 1) = \{p_2\} \\ \delta(p_3, 0) = \{p_2\} & \delta(p_3, 1) = \{p_4\} \\ \delta(p_4, 0) = \{p_4\} & \delta(p_4, 1) = \{p_4\} \end{array}$$

L'automate $\mathcal{A}_1$ *sera symbolisé par le graphe de la figure 4.*

La figure 4 est une représentation graphique de l'automate $\mathcal{A}_1$ et le tableau 1 est une autre représentation de ses transitions.

Les noms des noeuds n'a en définitive que peu d'intérêt théorique quant à la manipulation d'un automate. Ils ont surtout un intérêt pratique qui permet à une personne de désigner chaque noeud. On pourrait donc très bien ne pas faire apparaitre le nom des noeuds ce qui donnerait l'automate de la figure 5.

Dans l'exemple précédent, l'état p_4 est un puits. Il est parfois inutile de faire apparaitre ces puits qui peuvent être supprimés. Il est clair que tout automate ayant plusieurs rebuts est équivalent à l'automate obtenu en identifiant tous ces rebuts en un seul. Ceci conduit au diagramme simplifié de la figure 6.

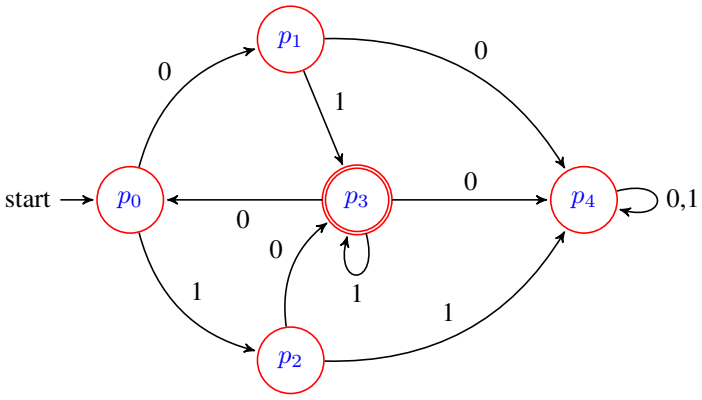


FIGURE 4 – Représentation graphique de l'automate $\mathcal{A}_1$

	↱	0	1
⇒	p_0	$\{p_1\}$	$\{p_3\}$
	p_1	$\{p_4\}$	$\{p_2\}$
	p_2	$\{p_0\}$	$\{p_2\}$
⇐	p_3	$\{p_2\}$	$\{p_4\}$
	p_4	$\{p_4\}$	$\{p_4\}$

TABLE 1 – Tableau des transitions de l'automate $\mathcal{A}_1$

Sur un tel diagramme, on peut suivre le chemin parcouru lors de la tentative de reconnaissance d'un mot. Par exemple, pour le mot 01010101 on a le chemin de la figure 7 (on voit que le mot est reconnu).

Par contre le mot 1101 n'est pas reconnu puisque lorsqu'on a avancé jusque 110 on est dans l'état p_3 mais on ne peut plus avancé ou bien on tombe dans le puits.

Exemple 10. *Considérons par exemple l'automate* $\mathcal{A}_2 = \langle \Sigma, Q, I, F, \delta \rangle$ *avec :*

- $\Sigma = \{a, b\}$
- $Q = \{p_0, p_1, p_2, p_3, p_4\}$
- *l'état initial est* $I = \{p_0\}$
- *le seul état final est* $F = \{p_3, p_4\}$
- *la fonction de transition est définie par :*

$\delta(p_0, a) = \{p_1, p_3\}$ $\quad \delta(p_0, b) = \{\}$
$\delta(p_1, a) = \{p_4\}$ $\quad \delta(p_1, b) = \{p_1\}$
$\delta(p_2, a) = \{p_2\}$ $\quad \delta(p_2, b) = \{p_0\}$
$\delta(p_3, a) = \{p_0\}$ $\quad \delta(p_3, b) = \{p_2, p_4\}$
$\delta(p_4, a) = \{p_1, p_4\}$ $\quad \delta(p_4, b) = \{p_4\}$

Il sera symbolisé par le graphe :

Le tableau 2 est une représentation des transitions par un tableau.

↱		a	b
⇒	p_0	$\{p_1, p_3\}$	$\{\}$
	p_1	$\{p_4\}$	$\{p_1\}$
	p_2	$\{p_2\}$	$\{p_0\}$
⇐	p_4	$\{p_1, p_4\}$	$\{p_4\}$

TABLE 2 – Tableau des transitions de l'automate $\mathcal{A}_2$

Cette fois-ci on remarque plusieurs choses. D'abord il y a plusieurs états de sortie : p_3 et p_4. Ensuite à partir de l'état p_0 on ne peut pas lire la lettre b. Ceci signifie que les mots reconnus par cet automate

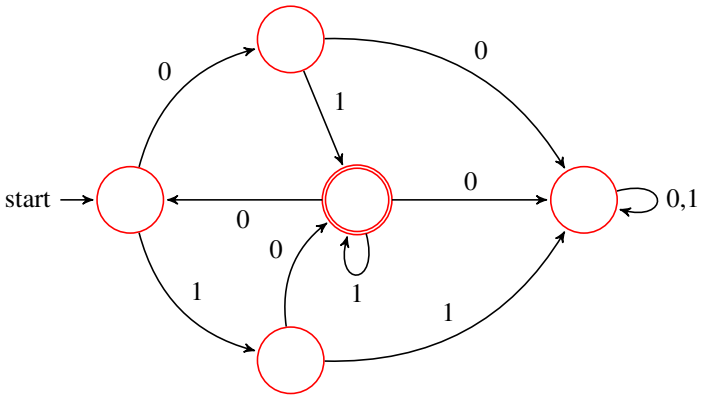


FIGURE 5 – Représentation graphique de l'automate $\mathcal{A}_1$

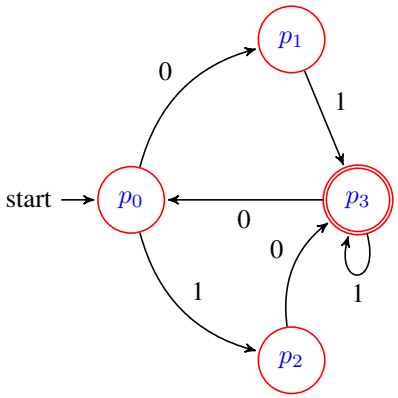


FIGURE 6 – Représentation graphique de l'automate $\mathcal{A}_2$ sans le puits p_4

commencent tous par la lettre a. Enfin, à partir de p_0, si on lit la lettre a on peut attendre deux états différents p_1 et p_3 ; à partir de p_3, si on lit la lettre b on peut attendre deux états différents p_2 et p_4 ; et à partir de p_4, si on lit la lettre a on peut attendre deux états différents p_1 et p_4. Lorsque l'on va essayer de reconnaitre un mot, il y aura des choix à faire dans le parcours du graphe associé à l'automate. Ceci rend les choses plus compliquées. En effet, quand il y a un choix à faire, rien ne dit que le choix effectué soit le bon et dans ce cas il faudra revenir en arrière.

4 Automates finis non déterministes

Lorsque qu'un automate a un comportement complètement déterminé par le mot qu'il a à examiner, on dit qu'il est *déterministe*. Pour des raisons de commodité et en particulier pour pouvoir définir certaines opérations de base sur les automates, nous allons généraliser les automates en leur adjoignant un comportement *a priori* aléatoire, ou plutôt *quantique*.
Nous allons autoriser l'automate, lorsqu'il est dans un état p et qu'il lit un caractère c, à passer dans un état quelconque d'un certain sous-ensemble $\delta(p,c)$. Si nous notons $\Delta = \{(p,c,q) \in Q \times \Sigma \times Q / q \in \delta(p,c)\}$, nous avons également $\delta(p,c) = \{q \in Q / (p,c,q) \in \Delta\}$, si bien que la donnée de Δ est équivalente à celle de $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ (comme précédemment, lorsqu'une transition n'est pas définie, on l'envoie dans un puits). Si bien que l'on peut poser :

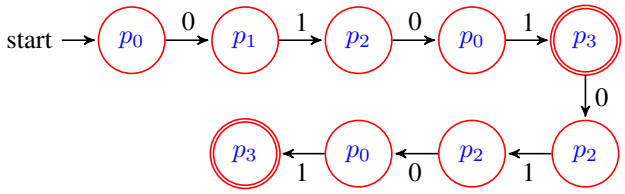


FIGURE 7 – Reconnaissance du mot 01010101 par l'automate $\mathcal{A}_1$

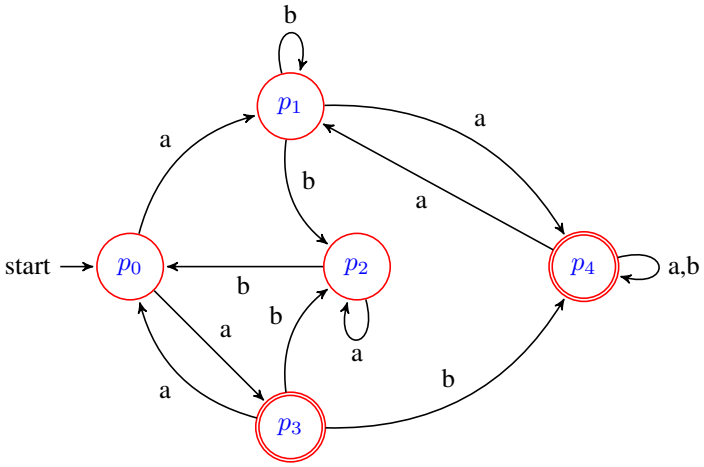


FIGURE 8 – Représentation graphique de l'automate $\mathcal{A}_2$

Définition 19. *On appelle automate fini non déterministe (ou plus simplement automate non déterministe, AND) la donnée :*

- *d'un alphabet* Σ
- *d'un ensemble fini (et non vide)* Q *appelé ensemble des états de l'automate*
- *d'un élément* p_0 *de* Q *appelé l'état initial ou état de départ*
- *d'une partie* F *de* Q *appelée l'ensemble des états finaux*
- *d'une partie* Δ *de* $Q \times \Sigma \times Q$ *appelée ensemble des transitions de l'automate*

On peut représenter un automate non déterministe par un diagramme sur lequel figureront tous les états de la machine. Ces états seront reliés par des flèches étiquetées par les éléments de l'alphabet comme ci dessus, la relation $(p, a, q) \in \Delta$ étant symbolisée par le même sous diagramme que d'habitude. La différence par rapport au diagramme d'un automate déterministe est que d'un même état peuvent partir plusieurs flèches ayant la même étiquette. Partons maintenant d'un état p et lisons un mot $w = c_1...c_n$. Nous pouvons parcourir plusieurs chemins dans l'automate suivant la transition choisie à chaque étape. Il existe donc plusieurs suites $p, q_1, ..., q_n$ d'états telles que :

$$p \overset{c_1}{\rightarrowtail} q_1 \overset{c_2}{\rightarrowtail} q_2 \overset{c_3}{\rightarrowtail} ... \overset{c_n}{\rightarrowtail} q_n$$

Nous noterons $\delta(p, w) \subseteq Q$ l'ensemble des états qn qui peuvent être atteints de cette manière. Ceci nous permet d'étendre la fonction $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ en une fonction $\delta : Q \times \Sigma^* \to \mathcal{P}(Q)$.

Nous poserons $\Delta = \{(p, w, q) \in Q \times \Sigma^* \times Q / q \in \delta(p, w)\}$ si bien que l'on a également $\delta(p, w) = \{q \in Q | (p, w, q) \in \Delta\}$. Nous écrirons encore $p \overset{w}{\rightarrowtail} q$ à la place de $(p, w, q) \in \Delta$.

Définition 20. *On dit que le mot* w *est reconnu par l'automate non déterministe* $\mathcal{A}$ *s'il existe un état final dans* $\delta(p_0, w)$.

Remarque : Il faut donc considérer qu'un AND est plus un automate quantique ou qu'un automate probabiliste. Il ne choisit pas un chemin au hasard, mais au contraire examine d'un seul coup tous les chemins possibles : si au moins l'un de ces chemins aboutit à un état final, il déclare qu'il a reconnu le mot.

5 Automates asynchrones

La notion d'automate non déterministe a déjà introduit un certain effet quantique dans la reconnaissance des langages, puisque ces automates explorent tous les chemins possibles à la fois. Nous allons persévérer dans cette direction en introduisant l'analogue d'un effet tunnel : nous allons admettre que l'automate peut changer d'état alors qu'il ne lit aucun caractère, autrement dit alors qu'il lit le mot vide ε ; un tel changement d'état, sera appelé une transition instantanée, ou encore une ε-transition. Nous verrons en effet que les transitions instantanées facilitent la construction d'un certain nombre d'opérations fondamentales sur les automates.

Définition 21 (Automate asynchrone)**.** *On appelle automate fini asynchrone ou automate non déterministe à transitions instantanées (ou automate non déterministe à ε-transitions, AND-ε) la donnée :*

- *d'un alphabet Σ*
- *d'un ensemble fini (et non vide) Q appelé ensemble des états de l'automate*
- *d'un élément p_0 de Q appelé l'état initial ou état de départ*
- *d'une partie F de Q appelée l'ensemble des états finaux*
- *d'une partie Δ de $Q \times \Sigma \times Q$ appelée ensemble des transitions de l'automate*
- *d'une partie Φ de $Q \times Q$ appelée ensemble des transitions instantanées de l'automate*

Un automate avec ε-transitions est donc de la forme : $\mathcal{A} = \langle \Sigma, Q, I, F, \delta, \Phi \rangle$ On peut représenter un automate non déterministe à transitions instantanées par un diagramme sur lequel figureront tous les états de la machine. Ces états seront reliés par des flèches étiquetées par les éléments de l'alphabet comme ci dessus, ou par le mot vide ε, la relation $(p, a, q) \in \Delta$ étant symbolisée par le sous-diagramme :

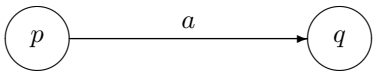


FIGURE 9 – Représentation d'une transition

la relation $(p, q) \in I$ étant symbolisée par

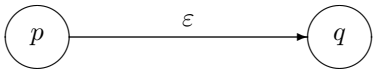


FIGURE 10 – Représentation d'une ε-transition

Définition 22. *Soit $w = c_1 ... cn$ un mot sur l'alphabet Σ. Nous allons définir récursivement la relation $p \overset{w}{\rightarrowtail} q$ (la notation $\rightarrowtail$ permet de distinguer les transitions complétées des transitions usuelles) de la façon suivante :*

- *si $w = \varepsilon$ (c'est à dire $n = 0$), on a $p\ \varepsilon \rightarrowtail q$ si et seulement si :*
  - *soit $q = p$,*
  - *soit il existe $q_1, ..., q_p$ ($p \geqslant 0$) tels que (p, q_1), (q_1, q_2), ..., (q_p, q) soient toutes des transitions instantanées*
- *si $w = a \in \Sigma$ (c'est à dire si $n = 1$), on a $p\ a \rightarrowtail q$ si et seulement si il existe $q_1 \in Q$ et $q_2 \in Q$ tels que $p\ \varepsilon \rightarrowtail q_1$, $q_1\ a \rightarrowtail q_2$ et $q_2\ \varepsilon \rightarrowtail q$*
- *si $w = c_1 ... c_n$ (avec $n \geqslant 2$) et si $w' = c_1 ... c_n$, alors $p\ w \rightarrowtail q$ signifie qu'il existe $q' \in Q$ tel que $p\ w' \rightarrowtail q'\ c_n \rightarrowtail q$*

Définition 23. *On dit qu'un mot w est reconnu par l'automate non déterministe à transitions instantanées $\mathcal{A}$ s'il existe un état final $q \in F$ tel que $p_0 w \mapsto q$.*

6 Opération sur les automates

6.1 Somme de deux automates

Définition 24. *Soient $\mathcal{A}_1 = \langle \Sigma, Q_1, I_1, F_1, \delta_1 \rangle$ et $\mathcal{A}_2 = \langle \Sigma, Q_2, I_2, F_2, \delta_2 \rangle$ deux automates définis sur le même alphabet Σ. L'automate $\mathcal{A}^{\oplus} \langle \Sigma, Q^{\oplus}, I^{\oplus}, F^{\oplus}, \delta^{\oplus} \rangle$ somme de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par :*

- $Q^{\oplus} = Q_1 \cup Q_2$
- $I^{\oplus} = I_1 \cup I_2$
- $F^{\oplus} = F_1 \cup F_2$
- $\delta^{\oplus} = \delta_1 \cup \delta_2$

Proposition 3. $\mathcal{L}(\mathcal{A}_1 \oplus \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$

6.2 Produit de deux automates

Définition 25. *Soient $\mathcal{A}_1 = \langle \Sigma, Q_1, I_1, F_1, \delta_1 \rangle$ et $\mathcal{A}_2 = \langle \Sigma, Q_2, I_2, F_2, \delta_2 \rangle$ deux automates définis sur le même alphabet Σ. L'automate $\mathcal{A}^{\otimes} \langle \Sigma, Q^{\otimes}, I^{\otimes}, F^{\otimes}, \delta^{\otimes} \rangle$ produit de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par :*

- $Q^{\otimes} = Q_1 \times Q_2$
- $I^{\otimes} = I_1 \times I_2$
- $F^{\otimes} = F_1 \times F_2$
- $\delta^{\otimes} = \{(p_1, p_2), x, (q_1, q_2), (p_1, x, q_1) \in \delta_1, (p_2, x, q_2) \in \delta_2\}$

Chaque état possède deux composantes : la première vient de Q_1 et la seconde vient de Q_2. Pour chaque transition $t^{\otimes}$ de $\mathcal{A}^{\otimes}$, la projection de $t^{\otimes}$ sur la première composante des états est une transition de $\mathcal{A}_1$: (p_1, x, q_1) et la projection de $t^{\otimes}$ sur la deuxième composante des états est une transition de $\mathcal{A}_2$: (p_2, x, q_2). Ces deux projections portent la même étiquette x.

Proposition 4. $\mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$

Ainsi, grâce à une opération sur les automates ($\otimes$) on obtient le résultat d'une opération sur les langages ($\cap$).

Exemple 11. *Soient deux automates* $\mathcal{A}_1 = \langle \Sigma, Q_1, I_1, F_1, \delta_1 \rangle$ *et* $\mathcal{A}_2 = \langle \Sigma, Q_2, I_2, F_2, \delta_2 \rangle$ *définis par :*

- $\Sigma = \{a, b\}$
- $Q_1 = \{X, Y\}$
- $I_1 = \{X\}$
- $F_1 = \{Y\}$
- $\delta_1 = \{(X, a, X), ((X, b, Y), (Y, a, Y), (Y, b, X)\}$
- $Q_2 = \{A, B\}$
- $I_2 = \{A\}$
- $F_2 = \{A\}$
- $\delta_2 = \{(A, a, B), ((A, b, A), (B, b, B), (B, a, A)\}$

L'automate produit $\mathcal{A}^{\otimes} = \mathcal{A}_1 \otimes \mathcal{A}_2 = \langle \Sigma, Q^{\otimes}, I^{\otimes}, F^{\otimes}, \delta^{\otimes} \rangle$ *est défini par :*

- $\Sigma = \{a, b\}$
- $Q^{\otimes} = Q_1 \times Q_2 = \{(X, A), (X, B), (Y, A), (Y, B)\}$
- $I^{\otimes} = I_1 \times I_2 = \{(X, A)\}$
- $F^{\otimes} = F_1 \times F_2 = \{(Y, A)\}$
- $\delta^{\otimes} = \{(p_1, p_2), x, (q_1, q_2), (p_1, x, q_1) \in \delta_1, (p_2, x, q_2) \in \delta_2\}$
  $= \{((X, A), a, (X, B)), ((X, A), b, (Y, A)), ((X, B), a, (X, A)), ((X, B), b, (Y, A)),$
  $((Y, A), a, (Y, B)), ((Y, A), b, (X, A)), ((Y, B), a, (Y, A)), ((Y, B), b, (X, B))\}$

Le produit des 2 automates est dessiné en figure 12.

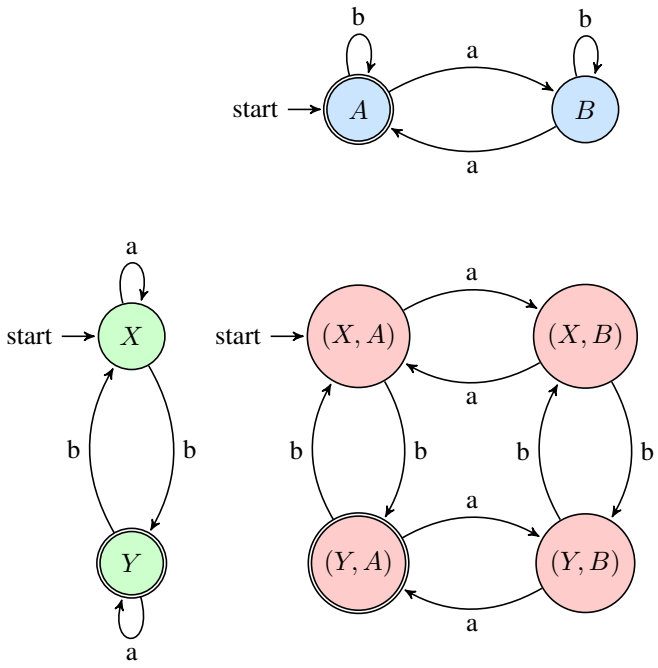


FIGURE 11 – L'automate rouge est le produit des automates bleu et vert. Le langage reconnu par l'automate rouge est l'intersection des langages reconnus par les automates bleu et vert.

6.3 Concaténation de deux automates

Définition 26. *Soient $\mathcal{A}_1 = \langle \Sigma, Q_1, I_1, F_1, \delta_1 \rangle$ et $\mathcal{A}_2 = \langle \Sigma, Q_2, I_2, F_2, \delta_2 \rangle$ deux automates définis sur le même alphabet Σ. L'automate $\mathcal{A}^{\odot} = \langle \Sigma, Q^{\odot}, I^{\odot}, F^{\odot}, \delta^{\odot} \rangle$ produit de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par :*

- $Q^{\odot} = Q_1 \cup Q_2$
- $I^{\odot} = I_1$
- $F^{\odot} = F_2$
- $\delta^{\odot} = \delta_1 \cup \delta_2 \cup \{(f, \varepsilon, i), f \in F_1, i \in I_2\}$

Proposition 5. $\mathcal{L}(\mathcal{A}_1 \odot \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \odot \mathcal{L}(\mathcal{A}_2)$

Ainsi, grâce à une opération sur les automate ($\odot$) on obtient le résultat d'une opération sur les langages (.).

Exemple 12. *On reprend les 2 automates $\mathcal{A}_1$ et $\mathcal{A}_2$ de l'exemple précédent.*
L'automate concaténation $\mathcal{A}^{\odot} = \mathcal{A}_1 \odot \mathcal{A}_2 = \langle \Sigma, Q^{\odot}, I^{\odot}, F^{\odot}, \delta^{\odot} \rangle$ est défini par :

- $\Sigma = \{a, b\}$
- $Q^{\odot} = Q_1 \cup Q_2 = \{X, Y, A, B\}$
- $I^{\odot} = I_1 = \{X\}$
- $F^{\odot} = F_2 = \{A\}$
- $\delta^{\odot} = \{(X, a, X), ((X, b, Y), (Y, a, Y), (Y, b, X), (A, a, B), ((A, b, A), (B, b, B), (B, a, A)\} \cup \{(f, \varepsilon, i), f \in F_1 = \{Y\}, i \in I_2 = \{A\}\}$
  $= \{(X, a, X), ((X, b, Y), (Y, a, Y), (Y, b, X), (A, a, B), ((A, b, A), (B, b, B), (B, a, A), (Y, \varepsilon A)\}$

La concaténation des 2 automates est dessiné en figure **??**.

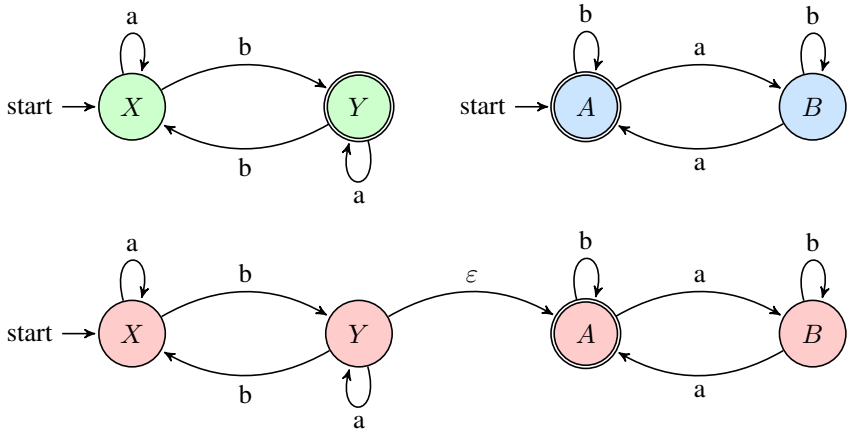


FIGURE 12 – L'automate rouge est la concaténation des automates bleu et vert. Le langage reconnu par l'automate rouge est le langage concaténation des langages reconnus par les automates bleu et vert.

7 Réduction des automates

7.1 Accessibilité - automates émondés

Définition 27. *Si p_0 et p_m sont deux états d'un automate $\mathcal{A}$, on appelle chemin de p_0 à p_m toute suite $(p_0, p_1, ..., p_m)$ d'états de $\mathcal{A}$ telle que, pour tout $i \in [1, m]$, il existe une transition (soit étiquetée par un élément de $\mathcal{A}$, soit instantanée) de p_{i-1} à p_i. On dira qu'un chemin est réussi s'il aboutit à un état final.*

Définition 28. *Soit $\mathcal{A}$ un automate d'état initial p_0 dont l'ensemble des états finaux est F . On dit qu'un état p de $\mathcal{A}$ est accessible s'il existe un chemin de p_0 à p. On dit que $\mathcal{A}$ est accessible si tout état de $\mathcal{A}$ est accessible. On dit qu'un état p de $\mathcal{A}$ est coaccessible s'il existe un chemin de p à un état final. On dit que $\mathcal{A}$ est coaccessible si tout état de $\mathcal{A}$ est coaccessible. On dit que l'automate est émondé s'il est à la fois accessible et coaccessible.*

Il est clair qu'un état non accessible d'un automate ne fera jamais partie du processus de reconnaissance d'un mot sur l'alphabet Σ et peut donc être supprimé de l'automate sans changer le langage reconnu. En ce qui concerne un état non coaccessible p, si le processus de reconnaissance d'un mot passe par p, ce processus ne peut aboutir à la reconnaissance du mot, puisqu'aucun chemin ne va de p à un état final. Les états non coaccessibles peuvent donc être remplacés par un unique puits. Tout automate peut donc être remplacé par un automate émondé, sans changer le langage reconnu.

Il reste à donner un algorithme qui permette, à partir d'un automate, de construire l'ensemble des états accessibles et celui des états coaccessibles. Ensuite la construction de l'automate émondé reconnaissant le même langage sera élémentaire. Soit donc $\mathcal{A}$ un automate, Q l'ensemble de ses états, p_0 son état initial, F l'ensemble des états finaux. Définissons deux suites $(P_{i\in\mathbb{N}})$ et $(Q_{i\in\mathbb{N}})$ de parties de Q par :

- $P_0 = \{p_0\}$
- $Q_0 = F$
- $P_{i+1} = P_i \cup \{p \in Q$ tel que il exite une transition d'un élément de P_i à $p\ \}$
- $Q_{i+1} = Q_i \cup \{p \in Q$ tel que il exite une transition d'un élément de p à $Q_i\ \}$

On montre immédiatement par récurrence que P_n est exactement l'ensemble des états tels qu'il existe un chemin de longueur au plus n de p_0 à p et que Q_n est l'ensemble des états p tels qu'il existe un chemin de longueur au plus n de p à un état final, si bien que :

Proposition 6. *L'ensemble des états accessibles de $\mathcal{A}$ est $\bigcup_{n\in\mathbb{N}} P_n$ et l'ensemble des états coaccessibles de $\mathcal{A}$ est $\bigcup_{n\in\mathbb{N}} Q_n$.*

Mais la suite P_n est une suite croissante (au sens de l'inclusion) dans l'ensemble fini des parties de Q. Il existe donc $M \in \mathbb{N}$ tel que $P_{M+1} = P_M$ auquel cas la définition même montre que $P_{M+2} = P_{M+1} = P_M$, puis par une récurrence triviale que $\forall n \geqslant M, P_n = P_M$. On en déduit donc que l'ensemble des états accessibles est $\bigcup_{n\in\mathbb{N}} P_n = P_M$

Le même raisonnement montre que l'ensemble des états coaccessibles est égal à Q_N où N est le premier entier tel que $Q_{N+1} = Q_N$.

L'algorithme de construction de l'automate émondé équivalent à un automate donné peut donc se formuler de la manière suivante :

- Poser $P_0 = \{p_0\}$ et calculer les P_n jusqu'à ce que $P_{M+1} = P_M$
- Supprimer $P - P_M$
- Poser $Q_0 = F$ et calculer les Q_n jusqu'à ce que $Q_{N+1} = Q_N$
- Remplacer $Q - Q_N$ par un unique puits

La complexité de la construction provient bien entendu de la nécessité d'explorer systématiquement toutes les transitions possibles pour construire P_{n+1} à partir de P_n.

7.2 Déterminisation d'un automate

On peut se demander si les automates non déterministes sont plus puissants, du point de vue des langages reconnus, que les automates non déterministes. Il n'en est rien, d'après le théorème suivant :

Théorème 1. *Pour tout automate non déterministe $\mathcal{A}$, il existe un automate déterministe $\mathcal{A}^d$ équivalent : tout mot de Σ est reconnu par $\mathcal{A}$ si et seulement si il est reconnu par $\mathcal{A}^d$.*

Démonstration. Supposons que $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta\rangle$ avec $\Delta \subseteq Q \times \Sigma \times Q$ et comme précédemment soit $\delta : Q\times\Sigma \to \mathcal{P}(Q)$ définie par $\delta(p, c) = \{q \in Q | (p, c, q) \in \Delta\}$. Quitte à ajouter un état puits, nous pouvons

supposer que δ est bien définie sur $Q \times \Sigma$ tout entier. Nous allons poser $Q' = \mathcal{P}(Q)$ (ensemble des parties de Q), $p'_0 = \{p_0\}$, $F' = \{X \subseteq Q | X \cap F \neq \emptyset\}$ et $\delta' : Q' \times \Sigma \to Q'$ définie par $\delta'(X, c) = \bigcup_{x \in X} \delta(x, c)$ (pour $X \subseteq Q$).

Considérons alors $\mathcal{A}^d = \langle \Sigma, Q', \{p'_0\}, F', \delta' \rangle$: il s'agit d'un automate déterministe. Nous allons montrer qu'il reconnaît exactement les mêmes mots que $\mathcal{A}$. Nous noterons $\overline{\delta}$ et $\overline{\delta'}$ les extensions respectives de δ et δ' aux mots de Σ^* □

Lemme 1. *Soit $w \in \Sigma^*$ et $X \subseteq Q$. Alors $\overline{\delta'}(X, x) = \bigcup_{x \in X} \overline{\delta}(x, w)$*

Démonstration. Nous allons procéder par récurrence sur la longueur n de w.

Si $n = 0$, w est le mot vide ε et on a $\overline{\delta}(x, \varepsilon) = \{x\}$ et $\overline{\delta}(X, \varepsilon) = X$. Comme $X = \bigcup_{x \in X} \{x\}$, l'égalité est vérifiée.

Si $n = 1$, w est réduit à un caractère c et par définition $\delta'(X, c) = \bigcup_{x \in X} \delta(x, c) = \bigcup_{x \in X} \overline{\delta}(x, w)$.

Supposons maintenant le résultat démontré pour les mots de longueur $n - 1$ et soit $w = c_1 ... cn$. On a vu que si $w' = c_2 ... cn$, alors :

$$\forall p \in Q, \overline{\delta}(p, w) = \bigcup_{x \in \delta(p, c_1)} \overline{\delta}(x, w')$$

On a donc, si $X \subseteq Q$ alors

$$\overline{\delta'}(X, w) = \overline{\delta'}(\overline{\delta'}(X, c_1), w') = \bigcup_{y \in \delta'(X, c_1)} \overline{\delta}(y, w')$$

par l'hypothèse de récurrence. Mais $\delta'(X, c_1)$ est l'ensemble des $y \in Q$ tels qu'il existe une transition dans $\mathcal{A}$ du type $x a \overset{c_1}{\longmapsto} y$ avec $x \in X$. Donc

$$\bigcup_{y \in \delta'(X, c_1)} \overline{\delta}(y, w')$$

est l'ensemble des $q \in Q$ tels qu'il existe une suite de transitions $x \overset{c_1}{\longmapsto} y \overset{w}{\longmapsto} q$ avec $x \in X$ et $y \in Q$, c'est à dire : $x \overset{w}{\longmapsto} q$ Cet ensemble est donc $\bigcup_{x \in X} \overline{\delta}(x, w)$: ce qui démontre le résultat.

Alors un mot est reconnu par $\mathcal{A}^d$ si et seulement si $\overline{\delta'}(p'_0, w) \in F'$ soit encore

$$\bigcup_{x \in \{p_0\}} \overline{\delta}(x, w) \cap F \neq \emptyset$$

c'est à dire

$$\overline{\delta}(p_0, w) \cap F \neq \emptyset$$

ce qui signifie exactement qu'il est reconnu par $\mathcal{A}$. □

7.3 Complexité de la déterminisation

La déterminisation d'un automate telle qu'elle a ét présentée ici n'est pas interessante d'un point de vue pratique. En effet, si l'on part d'un automate non déterministe dont l'ensemble Q des états possède n éléments, l'automate déterministe associé utilise $\mathcal{P}(Q)$ comme ensemble d'états, qui possède 2^n éléments. Ceci est illustré par l'exemple de l'automate ci-dessous qui reconnait le langage $\Sigma^* 1 \Sigma^3$ sur l'alphabet $\Sigma = \{0, 1\}$

L'automate déterministe associé est représenté ci-dessous (seuls les états accessibles sont représentés).

On est passé de 5 à 16 états. Plus généralement, on montre facilement que si on déterminise l'automate à $n + 2$ états qui reconnait le langage $\Sigma^* 1 \Sigma^n$, on obtient un automate déterministe à 2^{n+1} états, ce qui aboutit à une croissance exponentielle de l'occupation mémoire (sans compter le temps de calcul).

Pour déterminiser un automate, on peut suivre l'algorithme 1.

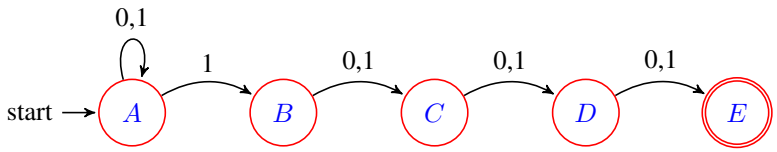


FIGURE 13 – Etat final

Algorithm 1: Algorithme de déterminisation d'un automate

Data: $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$
Result: $\mathcal{A}^d = \langle \Sigma, Q^d, I^d, F^d, \delta^d \rangle$ un automate déterministe équivalent à $\mathcal{A}$

$Q^d \leftarrow \{I\}$;
$I^d \leftarrow Q^d$;
$\delta^d \leftarrow \{\}$;
$F^d \leftarrow \{\}$;
$\Phi \leftarrow Q^d$;
while $\Phi \neq \{\}$ **do**
  soit $\alpha \in \Phi$;
  $\Phi \leftarrow \Phi - \{\alpha\}$;
  foreach $x \in \Sigma$ **do**
    $\beta \leftarrow accessibles(\alpha, x, \mathcal{A})$;
    if $\beta \notin D^d$ **then**
      $Q^d \leftarrow Q^d \cup \{\beta\}$;
      $\delta^d \leftarrow \delta^d \cup \{(\alpha, x, \beta)\}$;
      $\Phi \leftarrow \Phi \cup \{\beta\}$;
foreach $q \in Q^d$ **do**
  if $q \cap F \neq \{\}$ **then**
    $Q^d \leftarrow Q^d - \{q\}$

Exemple 13. *Prenons l'exemple de l'automate $\mathcal{A}_7$ de la figure 15 et appliquons-lui l'algorithme 1. Ceci conduit à construire progressivement les nouveaux états dont les noms sont des ensembles d'états. Cette construction est données dans la figure 16. On a renommé les états ce qui ne change rien à l'automate. Il n'y a qu'un seul état initial :* 1 *et un seul états de sortie :* 4*. Finalement, l'automate déterministe $\mathcal{A}_7^d et$ équivalent à $\mathcal{A}_7$ est donnée en figure 17.*

7.4 Reconnaissance par un AFND

Puisque la déterminisation complète de l'automate n'est pas réaliste, nous nous contenterons lors de la reconnaissance d'un mot, de construire au fur et à mesure les états utiles de l'automate déterministe : à partir d'une partie X de Q, lorsque nous lisons le caractère a, nous passons dans le nouvel état

$$\delta'(X, x) = \bigcup_{p \in X} \overline{\delta}(p, w) \cap F \neq \emptyset$$

Il s'agit alors d'opérations purement ensemblistes pour lesquelles la structure de liste est tout à fait adaptée. Bien entendu, il ne faut plus utiliser la fonction assoc qui ne renvoie que la première association dans une liste d'association, mais in nous faut construire une fonction qui renvoie toutes les associations. C'est le rôle de notre fonction cherche_tous dans le code ci-après.

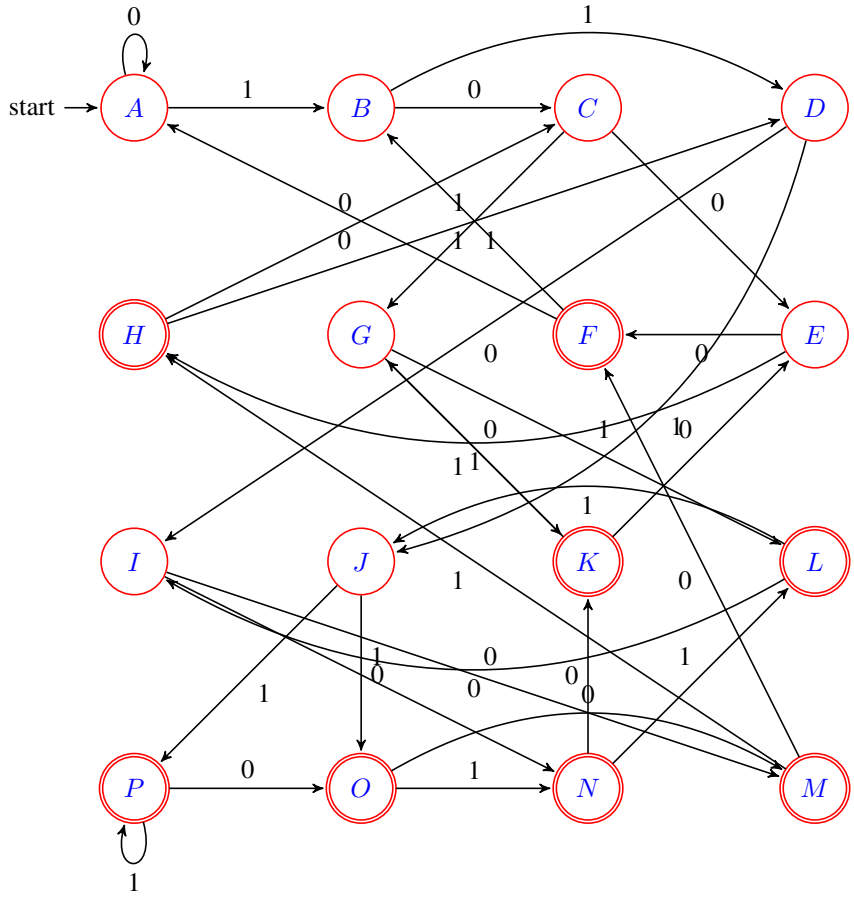


FIGURE 14 – Etat final

7.5 Suppression des ε-transitions

L'introduction des transitions instantanées, qui est une commodité pour la suite de la théorie, n'ajoute rien à la puissance de reconnaissance des automates. C'est ce que nous nous proposons de montrer. A chaque état $p \in Q$, nous pouvons associer l'ensemble des états q tels que $p \overset{\varepsilon}{\longmapsto} q$, c'est à dire l'ensemble des états auxquels on peut accéder à partir de p par une succession de transitions instantanées (ensemble auquel nous adjoindrons p lui-même). C'est donc l'ensemble des états que l'on peut atteindre à partir de p en lisant le mot vide.

Définition 29. *Soit $\mathcal{A}$ un automate non déterministe à transitions instantanées et p un état de $\mathcal{A}$. On appelle clôture instantanée de p le sous ensemble constitué de p et de tous les états q de $\mathcal{A}$ tels que $p \overset{\varepsilon}{\longmapsto} q$*

Théorème 2. *Pour tout automate non déterministe à transitions instantanées $\mathcal{A}$, il existe un automate déterministe $\mathcal{A}'$ équivalent à $\mathcal{A}$, c'est à dire qu'un mot de Σ^* est reconnu par $\mathcal{A}$ si et seulement si il est reconnu par $\mathcal{A}'$.*

Démonstration. soit $\mathcal{A} = \langle \Sigma, Q, I = \{p_0\}, F, \delta, \Phi \rangle$ un automate non déterministe à transitions instantanées, avec $\Delta \subseteq Q \times \Sigma \times Q$ et $\Phi \subseteq Q \times Q$.

Pour chaque $p \in Q$, nous disposons de la clôture instantanée de p, que nous noterons $cl(p)$. pour toute partie U de Q, on peut alors définir $cl(U)$ comme $cl(U) = \bigcup_{p \in U} cl(p)$. On a $U \subseteq cl(U)$ Nous poserons alors $\mathcal{A}' = \langle \Sigma, Q', I = \{p'_0\}, F', \delta', \Phi \rangle$ avec

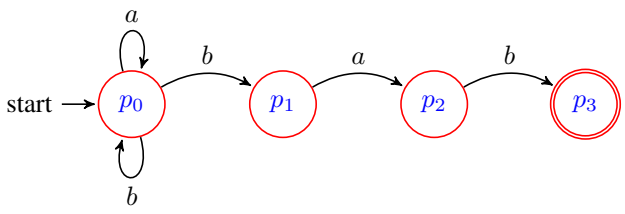


FIGURE 15 – Exemple d'un automate à déterminiser ($\mathcal{A}_7$)

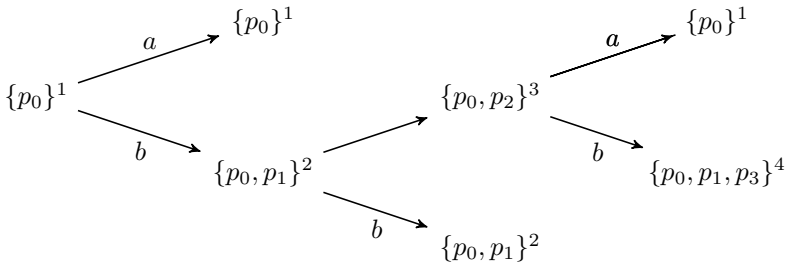


FIGURE 16 – Construction des états de l'automate à déterminiser

- $Q' = \{cl(U) | U \subset Q\}$
- $p'_0 = cl(p_0)$
- $F' = \{U \in Q' | U \cap F \neq \emptyset\}$
- $\delta'(U, a) = \{q \in Q | \exists p \in U, p \overset{a}{\longmapsto} q\}$

Pour justifier cette définition, il suffit de remarquer que si $V = \delta'(U, a) = \{q \in Q | \exists p \in U, p \overset{a}{\longmapsto} q\}$

on a $V = cl(V) \in Q'$; en effet il est clair que $V \subseteq cl(V)$ et inversement, si $q \in cl(V)$, il existe $q' \in V$ tel que $q' \overset{\varepsilon}{\longmapsto} q$; pour ce q', il existe $p \in U$ tel que $p \overset{a}{\longmapsto} q$, mais on a alors $p \overset{a}{\longmapsto} q \overset{\varepsilon}{\longmapsto} q'$ soit encore $p \overset{a}{\longmapsto} q$ et donc $q \in V$. Soit alors $w = c_1 ... c_n$ un mot reconnu par $\mathcal{A}'$; on a donc une suite $U_0, ..., U_n$ de Q' telle que $U_0 = p'_0 = cl(p_0)$, $U_{i+1} = \delta(U_i, c_i)$ et $U_n \cap F \neq \emptyset$; soit alors $q_n \in U_n \cap F$; il existe $q_{n-1} \in U_{n-1}$ tel que tel que $q_| n-1 \overset{c_n}{\longmapsto} q_n$ puis $q_{n-2} \in U_{n-2}$ tel que $q_{n-2} \overset{c_{n-1}}{\longmapsto} q_{n-1}$ et ainsi de suite jusqu'à $q_0 \in U_0$ tel que :

$$q_0 \overset{c_1}{\longmapsto} q_1 \overset{c_2}{\longmapsto} ... \overset{c_{n-1}}{\longmapsto} q_{n-1} \overset{c_n}{\longmapsto} q_n$$

Mais comme q_0 appartient à $cl(p_0)$, on a $p_0 \overset{\varepsilon}{\longmapsto} q_0 \overset{c_1}{\longmapsto} q_1$ et donc $p_0 \overset{c_1}{\longmapsto} q_1$. Ceci montre que :

$$p_0 \overset{c_1}{\longmapsto} q_1 \overset{c_2}{\longmapsto} ... \overset{c_{n-1}}{\longmapsto} q_{n-1} \overset{c_n}{\longmapsto} q_n \in F$$

et donc w est reconnu par $\mathcal{A}$. Inversement, supposons que $w = c_1 ... c_n$ est reconnu par $\mathcal{A}$. On a alors $p_0 \overset{w}{\longmapsto} q$ avec $q \in F$. Il existe donc $q_1, ..., q_n \in Q$ tels que :

$$p_0 \overset{c_1}{\longmapsto} q_1 \overset{c_2}{\longmapsto} ... \overset{c_{n-1}}{\longmapsto} q_{n-1} \overset{c_n}{\longmapsto} q_n = q \in F$$

Si on définit alors $U_0 = cl(p_0)$ puis $U_{i+1} = \delta'(U_i, c_i)$, la définition de δ' et une simple récurrence montrent que $\forall i \geq 1, q_i \in U_i$. En particulier $q = q_n \in F \cap U_n$, donc $U_n \in F'$ et w est reconnu par $\mathcal{A}'$. □

Pour supprimer les ε-transitions d'un automate , on peut suivre l'algorithme 2. Ligne 8, on ne s'interesse qu'aux états q qui aboutissent à l'état q_k par une ε-transition donc dans l'automate $\mathcal{A}^s$ on repère ls transitions $q \overset{\varepsilon}{\longmapsto} q_k$. Ligne 11, on recherche tous les états q' tels que l'on ait une transition $q_k \overset{x \in \Sigma}{\longmapsto} q'$ dans l'automate $\mathcal{A}^s$.

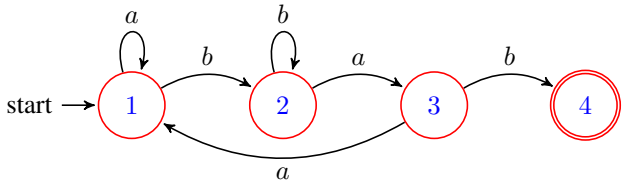


FIGURE 17 – Automate déterministe $\mathcal{A}_7^d$

Algorithm 2: Algorithme de synchronisation d'un automate

Data: $\mathcal{A} = \langle \Sigma \cup \{\varepsilon\}, Q, I, F, \delta \rangle$
Result: $\mathcal{A}^s = \langle \Sigma, Q^s, I^s, F^s, \delta^s \rangle$ un automate synchrone équivalent à $\mathcal{A}$

ordonner (*) les états de Q de manière arbitraire ;
$Q^s \leftarrow Q$;
$I^s \leftarrow I$;
$F^s \leftarrow F$;
$\delta^s \leftarrow \delta$;
while *il y a au moins une ε-transition dans δ^s* **do**
    soit $q_k \in Q^s$ le plus grand état au sens de (*) sur lequel aboutit au moins une ε-transition ;
    foreach $q \in Q^s - \{q_k\}$ *tel que* $(q, \varepsilon, q_k) \in \delta^s$ **do**
        foreach $q' \in Q^s$ **do**
            foreach $x \in \Sigma \cup \{\varepsilon\}$ **do**
                if $(q_k, x, q') \in \delta^s$ **then**
                    $\delta^s \leftarrow \delta^s \cup \{(q, x, q')\}$
    supprimer toutes les transitions (q, ε, q_k) de δ^s ;
    if $q_k \in F^s$ **then**
        $F^s \leftarrow F^s \cup \{q\}$
    if q_k *n'est plus accessible* **then**
        $Q^s \leftarrow Q^s - \{q_k\}$

Exemple 14. *On souhaite rendre synchrone l'automate $\mathcal{A}$ de la figure 18.*

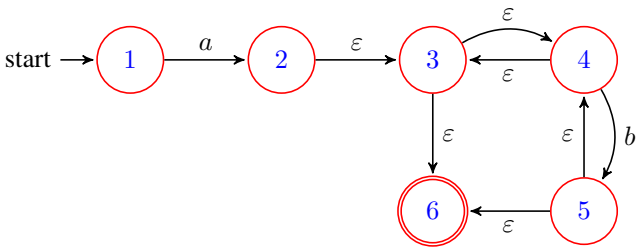


FIGURE 18 – Un automate asyncrone $\mathcal{A}$

Les états étant numérotés, on peut très bien conserver l'ordre existant. Le plus grand état sur lequel aboutit une ε-transition est l'état 6. δ contient les transitions $(3, \varepsilon, 6)$ et $(5, \varepsilon, 6)$ mais aucune transition ne part de l'état 6 donc aucune transition n'est ajoutée. Les transitions $(3, \varepsilon, 6)$ et $(5, \varepsilon, 6)$ sont supprimées. L'état 6 étant terminal, les états 5 et 3 deviennent terminaux. L'état 6 n'est plus accessible : il est supprimé. L'automate obtenu est celui de la figure 19.

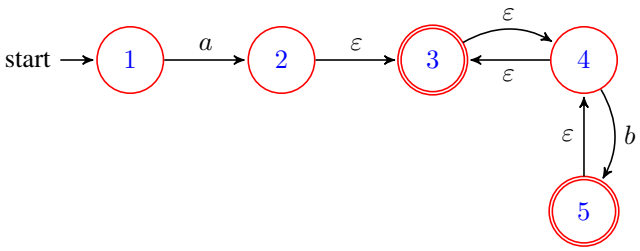


FIGURE 19 – Automate obtenu après suppression de l'état 6

Le plus grand état de cet automate sur lequel aboutit une ϵ-transition est maintenant l'état 4. δ contient les transitions $(3, \varepsilon, 4)$ et $(5, \varepsilon, 4)$ ainsi que les transitions $(4, \varepsilon, 3)$ et $(4, b, 5)$. Les transitions $(3, b, 5)$, $(5, \varepsilon, 3)$ et $(3, b, 5)$ sont ajoutées (il faudrait également ajouter la transition $(3, \varepsilon, 3)$ mais elle est inutile). On supprime les transitions $(3, \varepsilon, 4)$ et $(5, \varepsilon, 4)$. Du coup, l'état 4 devient inaccessible et il est supprimé. L'automate obtenu est celui de la figure 20.

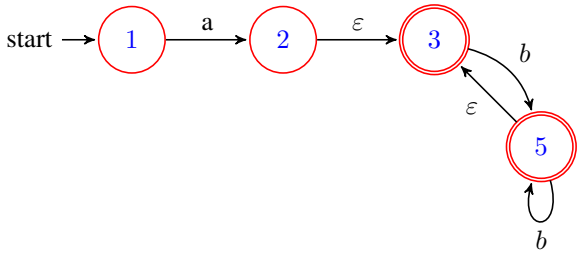


FIGURE 20 – Automate obtenu après suppression de l'état 4

Le plus grand état de cet automate sur lequel aboutit une ϵ-transition est maintenant l'état 3. δ contient les transitions $(2, \varepsilon, 3)$ et $(5, \varepsilon, 3)$ ainsi que la transition $(3, b, 5)$. Les transitions $(2, b, 5)$, $(5, b, 5)$ sont ajoutées (la transition $(5, b, 5)$ était déjà présente : il est inutile de l'ajouter). On supprime les transitions $(2, \varepsilon, 3)$ et $(5, \varepsilon, 3)$. L'état 3 étant un état terminal, l'état 2 devient un état terminal ainsi que l'état 5 (qui l'était déjà). Du coup, l'état 3 devient inaccessible et il est supprimé. L'automate obtenu est celui de la figure 21.

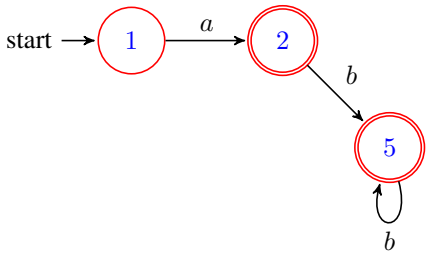


FIGURE 21 – Automate obtenu après suppression de l'état 3

On peut simplifier encore cet automate et finalement on obtient l'automate synchrone de la figure 22.

7.6 Minimalisation des automates

On a vu comment transformer un automate $\mathcal{A}$ en un autre automate $\mathcal{B}$ qui lui est équivalent. On verra plus loin comment exprimer le langage reconnu par un automate (donner une expression régulière qui désigne le langage reconnu par $\mathcal{A}$). Finalement, il est légitime de se demander si parmi tous les automates

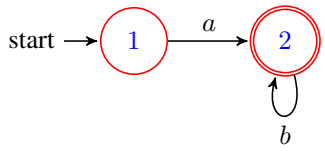


FIGURE 22 – L'automate synchrone équivalent à $\mathcal{A}$ obtenu par l'algorithme 2

qui reconnaissent un même langage il n'y en aurait pas certains qui soient les plus *simples* possibles ? La *simplicité* d'un automate peut dépendre du nombre de ses transitions ou du nombre de ses états.

Nous allons donc rechercher pour un automate donné $\mathcal{A}$ un automate $\mathcal{A}^{min}$ équivalent (donc qui reconnait le même langage) mais qui comporte le moins d'états possible. Il y a plusieurs algorithmes pour minimaliser un automate : l'algorithme de Brzozowski, l'algorithme de Moore et l'algorithme de Hopcroft.

7.6.1 Etats indistinguables

Soit un automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ déterministe et complet. Deux états p et q dont dits indistinguables (on dit parfois inséparables) si tout mot w reconnu par $\mathcal{A}$ à partir de p est aussi reconnu à partir de q et réciproquement : $\forall p \in Q, \forall q \in Q, p$ et q sont inséparables ssi $p.w \in F \iff q.w \in F$ Deux états sont séparables s'ils ne sont pas inséparables.

Définition 30. *Un automate fini déterministe complet et accessible est dit minimal si et seulement si tous ses états sont 2 à 2 séparables.*

7.6.2 Algorithme de minimalisation de Moore

Description L'idée exploitée dans cet algorithme consiste d'une part à regrouper les états et d'autre part à vérifier que tous les états d'un même groupe permettent pour une transition donnée d'aller non pas vers un autre état précis mais vers n'importe quel autre états d'un même groupe. S'il n'y a pas de raison de distinguer les états d'un groupe, ce groupe devient lui-même un état qui remplace tous les états qu'il contient. Les transitions sont les mêmes que celles de l'automate initial sans conserver les transitions internes aux groupes. sans les traqui sont induites sur les classes

Au départ on distingue deux groupes d'états : le groupe des états finaux et le groupe des autres états. Progressivement, on précise le partitionnement jusqu'à ce qu'il ne soit plus possible de distinguer les états d'un même groupe.

On définit des équivalences sur les états d'un automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ par :

$$p \approx_h q \iff L_q^{(h)}(\mathcal{A}) = L_p^{(h)}(\mathcal{A})$$

avec

$$L_q^{(h)}(\mathcal{A}) = \{w \in \Sigma^* | p.w \in F \wedge |w| \leq h\}$$

Chaque ensemble $L_q^{(h)}$ contient des mots de longueur au plus égale à h et qui mènent de l'état p à un état terminal de $\mathcal{A}$. Deux états de $\mathcal{A}$ pour la relation $\approx_h$ si les mêmes mots de longueur au plus égale à h : on ne peut donc pas séparer ces deux états par des mots de longueur plus petite que h. Le calcul des équivalences se fait grâce à la propriété suivante :

Lemme 2. $\forall p \in Q, \forall q \in Q, \forall h \geq 0, p \approx_{h+1} q \iff (p \approx_h q) \wedge (\forall l \in \Sigma, p.l \approx_h q.l)$

Pour calculer l'équivalence $\approx_{h+1}$, on détermine les états qui pour une lettre $l \in \Sigma$ arrivent dans des classes différentes de $\approx_h$: ces états se retrouvent dans des états différents de pour $\approx_{h+1}$. Pour celà, on détermine pour toute lettre $l \in \Sigma$ la partition dont les classes sont constituées des états qui par la lettre l arrivent sur des états équivalents pour $\approx_h$. Pui on calcule l'intersection des ces partitions avec la partition $\approx_h$ elle-même.

La complexité en temps de l'algorithme est, dans le pire des cas, $O(sn^2)$, où s est la taille de l'alphabet et n est le nombre d'états de l'automate. Chacune des étapes du raffinement peut être effectuée en temps $O(sn)$ en utilisant une variante appropriée du tri par base ou *radix sort*. À chaque étape, le nombre de classes augmente strictement, et comme au départ il y a deux classes, au plus $n-2$ étapes suffisent. En fait, la complexité est même en $O(lsn)$, où l est le nombre d'étapes de l'algorithme. On peut observer que le nombre d'étapes est indépendant du nombre d'états de l'automate de départ.

Mise en oeuvre Pour organiser les étapes de la minimalisation, étudions l'exemple suivant.

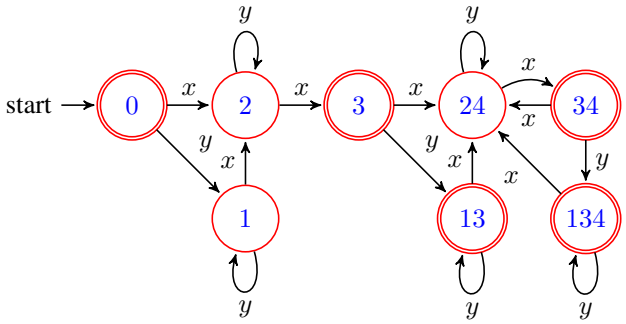


FIGURE 23 – Un automate à minimaliser

On crée un tableau dont les colonnes sont les différents états. Les valeurs qui vont remplir le tableau sont des classes et sont numérotées avec des nombres romains. La première ligne est notée ε et on sépare les états finaux (marqués par I) des autres états (marqués II). Cette première ligne constitue le premier bilan.

	0	1	2	3	13	24	34	134
bilan 0 = ε	I	II	II	I	I	II	I	I

On construit les lignes suivantes une à une en regardant pour chaque état p de l'automate vers quel autre état q mène chaque lettre de l'alphabet. On peut avoir $q = p$. On note la classe à laquelle appartient l'état q dans la dernière *ligne bilan* effectuée (ici il s'agit du bilan 0). Il y a une ligne par lettre de l'alphabet.

	0	1	2	3	13	24	34	134
bilan 0 = ε	I	II	II	I	I	II	I	I
x	II	II	I	II	II	I	II	II
y	II	II	II	I	I	II	I	I

On fait un bilan en donnant le même numéro de classe au colonnes identiques et en différenciant les colonnes différentes.

	0	1	2	3	13	24	34	134
bilan 0 = ε	I	II	II	I	I	II	I	I
x	II	II	I	II	II	I	II	II
y	II	II	II	I	I	II	I	I
bilan 1	I	II	III	IV	IV	III	IV	IV

On réitère ce processus tant que le bilan effectué est différents du bilan précédent. C'est le cas de l'exemple.

	0	1	2	3	13	24	34	134
bilan 1	I	II	III	IV	IV	III	IV	IV
x	III	III	IV	III	III	IV	III	III
y	II	II	III	IV	IV	III	IV	IV
bilan 2	I	II	III	IV	IV	III	IV	IV

On constate que les lignes bilan 1 et bilan 2 sont identiques donc on s'arrête. L'automate obtenu se compose des états I, II, III et IV. Les transitions sont lisibles directement dans le tableau :

bilan 1	I	II	III	IV
x	III	III	IV	III
y	II	II	III	IV

TABLE 3 – Calcul des classes (méthode de Moore)

L'état de départ (I) est l'état correspondant à la classe contenant l'état de départ de l'automate initial (0). Les états finaux sont les états correspondant aux classes contenant des états finaux de l'automate initial (0, 3, 13, 34 et 134) : ici ce sont donc les états I et IV.

Finalement, l'automate minimal recherché équivalent à $\mathcal{A}$ est celui-ci :

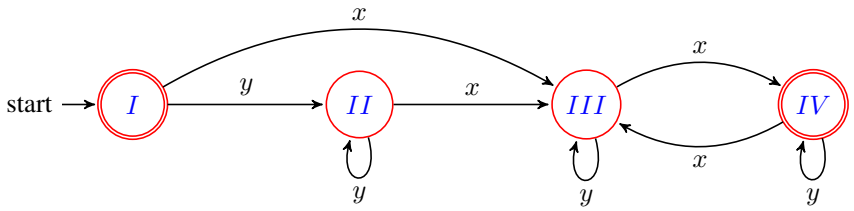


FIGURE 24 – Un automate à minimaliser

7.6.3 Algorithme de minimalisation de Hopcroft

Algorithm 3: Algorithme de minimalisation de Hopcroft

Data: $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$
Result: $\mathcal{A}^s = \langle \Sigma, Q^s, I^s, F^s, \delta^s \rangle$ un automate minimal équivalent à $\mathcal{A}$

```
P ← {F, Q\F} ;
W ← {F} ;
while W ≠ {} do
    choisir et retirer un ensemble A de W ;
    foreach c ∈ Σ do
        soit X l'ensemble des états pour lesquels une transition étiquetée c conduit à un état de A ;
        foreach Y ∈ P pour lequel X ∩ Y ≠ {} et Y\X ≠ {} do
            remplacer Y dans P par les 2 ensembles X ∩ Y et Y\X ;
            if Y ∈ W then
                remplacer Y dans W par les 2 mêmes ensembles ;
            else
                if |X ∩ Y| ≤ |Y\X| then
                    W ← W ∪ {X ∩ Y} ;
                else
                    W ← W ∪ {Y\X} ;
```

8 Passages entre automates et expressions régulières

8.1 Automate reconnaissant un langage donné

Etant donnée une expression régulière décrivant un langage $\mathcal{L}$, comment construire un automate $\mathcal{A}$ qui reconnait ce langage ? L'algorithme de Thompson répond à cette question.

8.1.1 Algorithme de Thompson

Définition 31 (Automate normalisé)**.** *L'automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ est dit normalisé s'il respecte les contraintes suivantes :*

- *L'ensemble I des états initiaux est réduit à un seul état début, L'ensemble F des états finaux est réduit à un seul état fin. Les états i et f sont disjoints.*
- *Il n'y a pas de transition $fin \rightarrow q_i$ ni de transition $q_i \rightarrow$ début*
- *tout état q est ou bien l'origine d'exactement une transition étiquetée par $x \in \Sigma$ ou bien l'origine d'au plus 2 ε-transitions.*

Définition 32 (Taille d'une expression)**.** *Soit une expression e, on définit la taille de e (et on la note $|e|$) la valeur calculée de la manière suivante :*

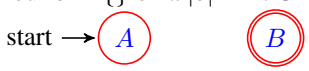
- *si $e = \{\}$, alors $|e| = 1$*
- *si $e = \varepsilon$, alors $|e| = 1$*
- *si $e = x, x \in \Sigma$, alors $|e| = 1$*
- *si $e = f + g$, alors $|e| = 1 + |f| + |g|$*
- *si $e = f.g$, alors $|e| = 1 + |f| + |g|$*
- *si $e = f^*$, alors $|e| = 1 + |f|$*

On peut remarquer que le nombre de transitions d'un automate normalisé est au plus le double du nombre de ses états.

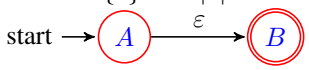
Proposition 7. *A toute expression rationnelle e de taille $m = |e|$, on peut lui associer un automate normalisé qui reconnait $\mathcal{L}(e)$ et dont le nombre des états est au plus $2 \times m$.*

Démonstration. La preuve va se faire par construction.

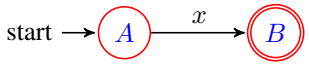
- Pour $e = \{\}$ on a $|e| = 1$. Un automate qui reconnait $\mathcal{L}(e)$ est :

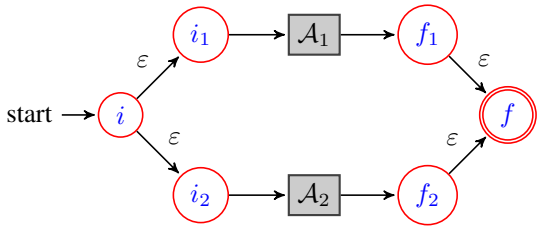
  Le nombre de noeuds est : 2.
- Pour $e = \{\varepsilon\}$ on a $|e| = 1$. Un automate qui reconnait $\mathcal{L}(e)$ est :

  Le nombre de noeuds est : 2.
- Pour $e = \{x \in \Sigma\}$, on a $|e| = 1$. Un automate qui reconnait $\mathcal{L}(e)$ est :

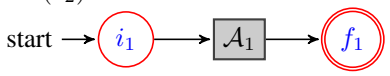
  

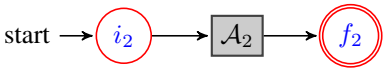
  Le nombre de noeuds est : 2.
- Pour $e = e_1 + e_2$, on a $|e| = 1 + |e_1| + |e_2|$. Un automate qui reconnait $\mathcal{L}(e)$ est :



Le nombre de noeuds est : $nb(\mathcal{A}_1) + nb(\mathcal{A}_2) + 2$

- Pour $e = e_1.e_2$, on a $|e| = 1 + |e_1| + |e_2|$. A partir des 2 automates suivants qui reconnaissent $\mathcal{L}(e_1)$ et $\mathcal{L}(e_2)$:



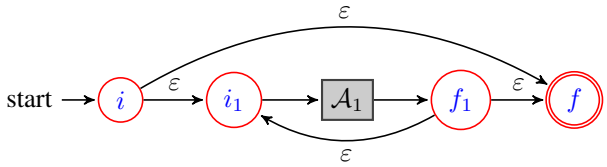


on construit l'automate suivant en fusionnant les états f_1 et d_2 :

Le nombre de noeuds est : $nb(\mathcal{A}_1) + nb(\mathcal{A}_2) - 1$

- Pour $e = e_1^*$, on a $|e| = 1 + |e_1|$. A partir de l'automates suivant qui reconnait $\mathcal{L}(e_1)$:

on construit l'automate suivant en ajoutant l'état initial d (d_1 n'est plus un état initial) et l'état final f (f_1 n'est plus un état final) ainsi que 4 ε-transitions :



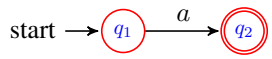
Le nombre de noeuds est : $nb(\mathcal{A}_1) + 4$

□

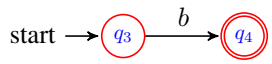
Exemple 15. *Nous allons utiliser l'algorithme de Thompson pour cnstruire un automate normalisé qui reconnais le langage $\mathcal{L}(e)$ avec $e = (a + b)^*b(\varepsilon + a)(a + b)^*$*

Appelons $e_1 = (a + b)^*$, $e_2 = b$, $e_3 = (\varepsilon + a)$, $e_4 = (a + b)^*$ On a $e = e_1.e_2.e_3.e_4$: l'automate recherché sera donc obtenu facilement à partir des automates $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ et $\mathcal{A}_4$. $e_1 = (a + b)^*$: en posant $f_1 = a$, $f_2 = b$ on peut facilement déterminer des automates qui reconnaissent $\mathcal{L}(f_1)$ et $\mathcal{L}(f_2)$:

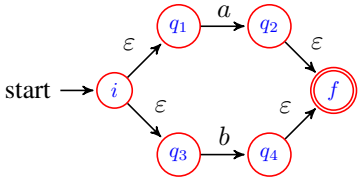
$\mathcal{L}(a)$:



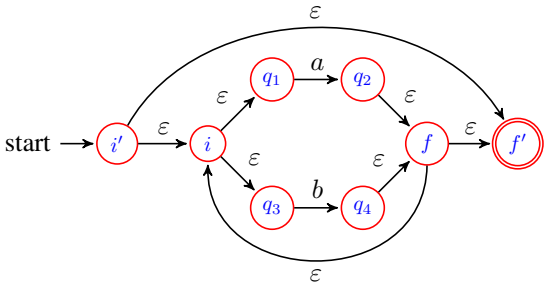
$\mathcal{L}(b)$



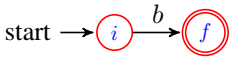
$\mathcal{L}(a+b)$



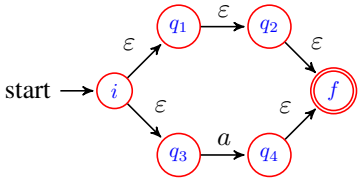
$\mathcal{L}((a+b)^*)$



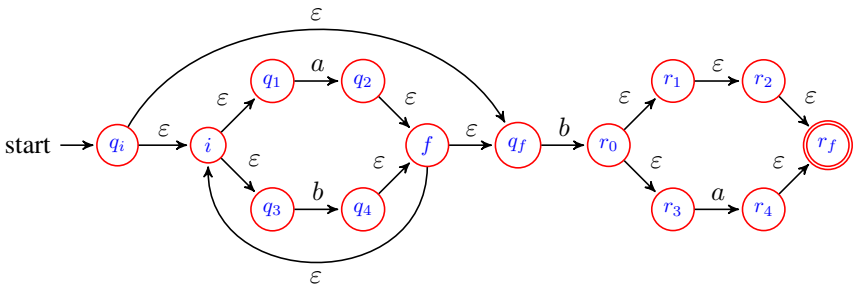
$\mathcal{L}(b)$

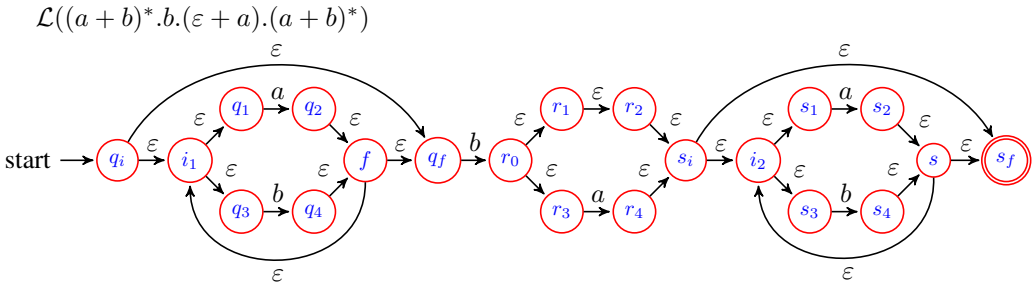


$\mathcal{L}(\varepsilon+a)$



$\mathcal{L}((a+b)^*.b.(\varepsilon+a))$





8.2 Langage reconnu par un automate

Etant donnée un automate $\mathcal{A}$ qui reconnait le langage $\mathcal{L}$, comment construire une expression régulière qui décrit le langage $\mathcal{L}$?

Deux méthodes permettent de répondre à ce problème.

8.2.1 Méthode des systèmes d'équations

Soit un automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ synchrone. Pour chaque état $q \in Q$, on note $\delta_q F = \{\varepsilon\}$ si $q \in F$ et $\delta_q F = \emptyset$ sinon. Pour tout couple $(p, q) \in Q^2$ on note $E_p q = \{x \in \Sigma | (p, x, q) \in \delta\}$. On appelle système d'équations linéaires droit associé à $\mathcal{A}$ l'ensemble des $|Q|$ équations à $|Q|$ inconnues $(X_{p \in Q})$ suivant :

$$\forall p \in Q, X_p = (\bigcup_{p \in Q} E_{p,q} X_q) \cup \delta_{qF}$$

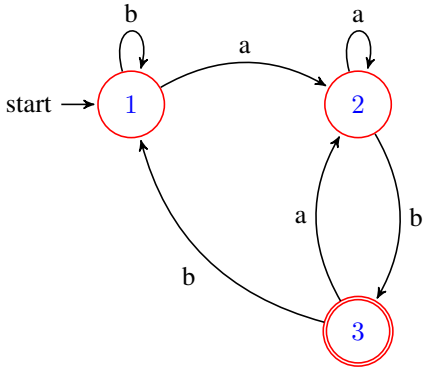


FIGURE 25 – Un automate

Exemple 16. *Pour l'automate de la figure 25, on calcule :*
$E_{1,1} = \{b\}$ $E_{1,2} = \{a\}$ $E_{1,3} = \{\}$ $\delta_{1,F} = \{\}$
$E_{2,1} = \{\}$ $E_{2,2} = \{a\}$ $E_{2,3} = \{b\}$ $\delta_{2,F} = \{\}$ *On a donc le système d'équations :*
$E_{3,1} = \{b\}$ $E_{3,2} = \{a\}$ $E_{3,3} = \{\}$ $\delta_{3,F} = \{\varepsilon\}$

$$X_1 = bX_1 \cup aX_2 \quad (1)$$
$$X_2 = aX_2 \cup bX_3 \quad (2)$$
$$X_3 = bX_1 \cup aX_2 \cup \{\varepsilon\} \quad (3)$$

Ce qui pourrait s'écrire :

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} b & a & \{\} \\ \{\} & a & b \\ b & a & \{\} \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} \cup \begin{pmatrix} \{\} \\ \{\} \\ \{\varepsilon\} \end{pmatrix}$$

Théorème 3. *Soit $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ un automate sunchrone.*
Le système d'équations associé : $\forall p \in Q, X_p = (\bigcup_{p \in Q} E_{p,q} X_q) \cup \delta_{qF}$ possède une unique solution $\mathcal{L}_{p \in Q}(\mathcal{A})$ avec $\mathcal{L}_p = \{w \in \Sigma^* | \exists p \xrightarrow{w} q, q \in F\}$

Lemme 3. *(Arden) Soient L_1 et L_2 2 langages tels que $\varepsilon \notin L_1$.*
L'équation $X = L_1 X \cup L_2$ a pour unique solution le langage $X = L_1^* L_2$.
L'équation $X = X L_1 \cup L_2$ a pour unique solution le langage $X = L_2 L_1^*$.

Démonstration. Montrons d'abord que le langage $X = L_1^* L_2$ est bien une solution de l'équation $X = L_1 X \cup L_2$. Commncons par remplacer X par $L_1^* L_2$ dans l'équation : $X = L_1^* L_2 L_2 \cup Ł_2 = (L_1^* L_2 \cup \{\varepsilon\}) Ł_2$. Par définition, $L_1^* = \bigcup_{n \geqslant 0}^{\infty} L_1^n$. Donc $L_1 L_1^* \cup \{\varepsilon\} = L_1 . \bigcup_{n \geqslant 0}^{\infty} L_1^n \cup \{\varepsilon\} = \bigcup_{n \geqslant 1}^{\infty} L_1^n \cup \{\varepsilon\} = \bigcup_{n \geqslant 0}^{\infty} L_1^n = L_1^*$ Et donc $(L_1 L_1^* \cup \{\varepsilon\}) L2 = L_1^* L2$.
$= L^0 \cup L^+ = \{\varepsilon\} \cup L^+$ □

Exemple 17. *(Suite de l'exemple précédent) En remplacant X_3 par $bX_1 + aX_2 + \varepsilon$ d'après l'équation (3) dans l'équation (2), on obtient :*
(4) $X_2 = aX_2 + b(bX_1 + aX_2 + \varepsilon) = (a + ba)X_2 + bbX_1 + \varepsilon$
D'après le lemme d'Arden, l'unique solution de cette équation est $X_2 = (a + ba)^*(bbX_1 + \varepsilon)$.
On remplace X_2 par $(a + ba)^*(bbX_1 + \varepsilon)$ dans l'équation (1) ce qui donne :
(5) $X_1 = bX_1 + a((a + ba)^*(bbX_1 + \varepsilon))$ ou $X_1 = (b + a(a + ba)^*bb)X_1 + a(a + ba)^*$. D'après le lemme d'Arden, l'unique solution de cette équation est $X_1 = (b + a(a + ba)^*bb)^* a(a + ba)^*$. Le langage reconnu par l'automate est finalement : $\mathcal{L}(\mathcal{A}) = (b + a(a + ba)^*bb)^* a(a + ba)^*$.

Sachant que pour tout langage A et B on a $(A + B)^* = A^*(BA^*)^*$ on pourrait simplifier l'expession précédente

8.2.2 Algorithme de Brzozowski - McCluskey (MBC)

La méthode de Brzozowski - McCluskey consiste à supprimer progressivement des états de l'automate. Les transitions qui deviennent donc obsolètes sont remplacées par d'autres transitions dont les étiquettes sont des expressions et non-plus seulement des lettres de l'alphabet.

Soit l'automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ pour lequel on cherche une expression régulière qui dénote le langage qu'il reconnait.

On commence par ajouter deux états α et ω à $\mathcal{A}$.
Puis, pour tout $q_i \in I$ on ajoute l'arc $(\alpha, \varepsilon, q_i)$ et pour tout $q_i \in F$ on ajoute l'arc $(q_i, \varepsilon, \omega)$ à δ. On suprime tous les états de I pour ne retenir que α comme unique état d'entrée et on suprime tous les états de F pour

ne retenir que ω comme unique état de sortie de $\mathcal{A}$.
On effectue ensuite les deux réductions suivantes jusqu'à ce que α et ω soient les duex seuls états restants :

- s'il existe 2 arcs (p, a, q) et (p, b, q) dans δ, les remplacer par l'arc $(p, (a + b), q)$
- choisir un état q autre que α et ω, remplacer chaque séquence de transitions $(p, a, q)(q, b, q)(q, c, r)$ par une transition (p, ab^*c, r) et supprimer l'état q (et donc supprimer également les transitions qui arrivent sur q ou qui partent de q).

A la fin de ces transformations, l'automate est réduit à 2 états α et ω et une seule transition dont l'étiquette est l'expression régulière qui dénote le langage reconnu par l'automate initial $\mathcal{A}$. Les 2 réductions consistent à remplacer les 2 situations suivantes :

FIGURE 26 – BMC :reduction 1

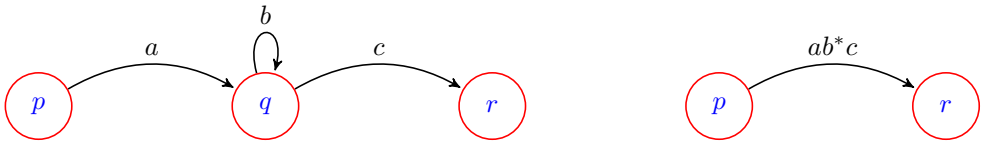


FIGURE 27 – BMC :reduction 2

Exemple 18. *Soit l'automate $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ décrit dans la figure 28.*

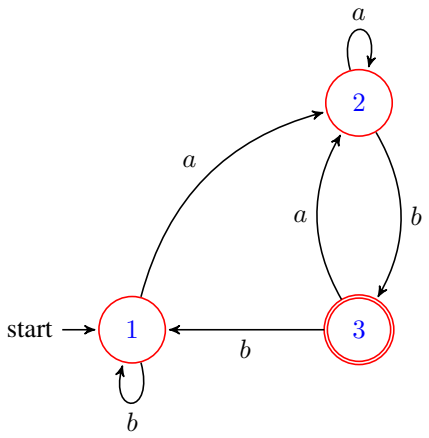


FIGURE 28 – L'automate $\mathcal{A}$ dont on cherche à exprimer le langage reconnu.

On ajoute les deux états α et ω et on ajoute des transitions de manière que α devienne le seul état initial et que ω devienne le seul état final. On obtient l'automate de la figure 29.

On commence par supprimer l'état 2 (par exemple). Ceci va entraîner la suppression des transitions qui concernent cet état et la créations des transitions décrites dans la figure 30.

Après la suppression de l'état 2 et l'ajout des nouvelles transitions, l'automate devient celui de la figure 31.

On supprime maintenat l'état 1 : ceci va entraîner la suppression des transitions qui concernent cet état et la créations des transitions décrites dans la figure 34.

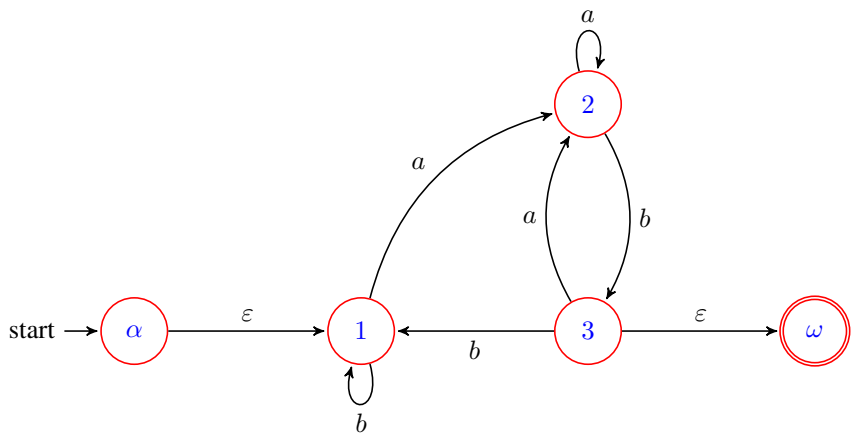


FIGURE 29 – L'automate $\mathcal{A}$ après l'ajout des états α et ω.

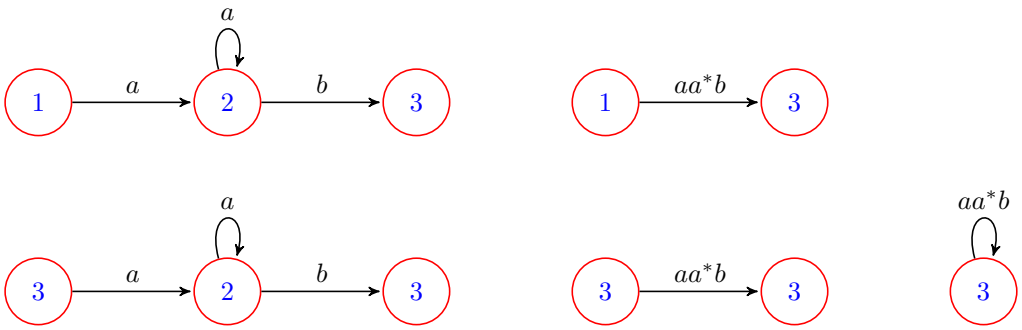


FIGURE 30 – Les transitions qui sont ajoutées lors de la suppression de l'état 2 dans $\mathcal{A}$.

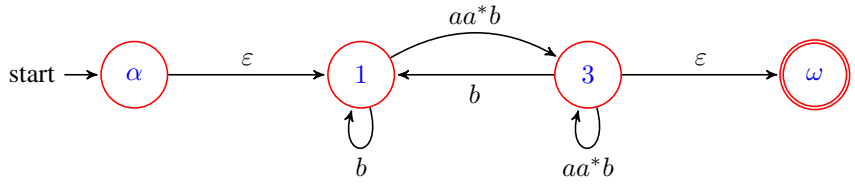


FIGURE 31 – L'automate $\mathcal{A}$ après la suppression de l'état 2.

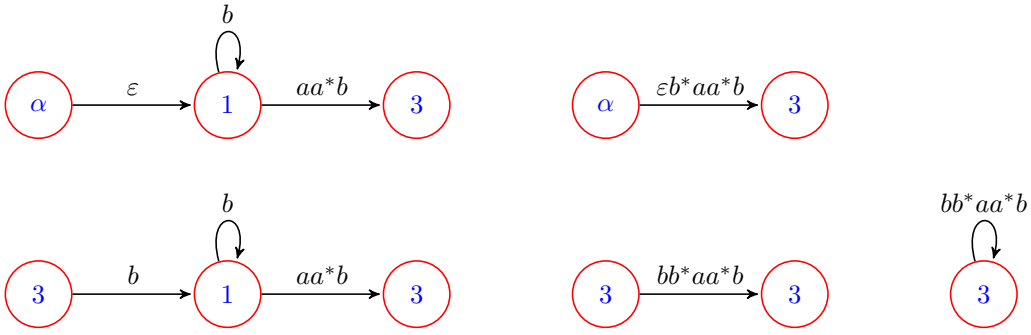


FIGURE 32 – Les transitions qui sont ajoutées lors de la suppression de l'état 1 dans $\mathcal{A}$.

Après la suppression de l'état 1 et l'ajout des nouvelles transitions, l'automate devient celui de la figure 33.

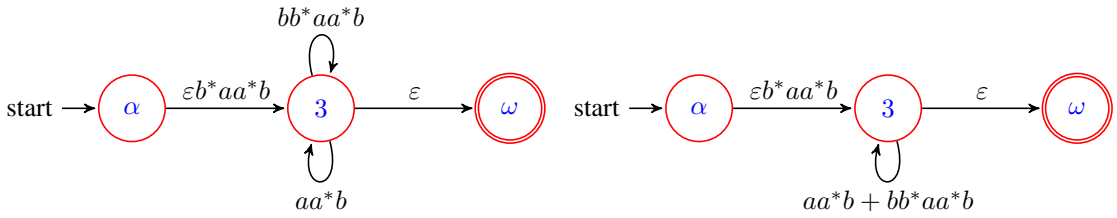


FIGURE 33 – L'automate $\mathcal{A}$ après la suppression de l'état 1.

Enfin, on supprime l'état 3 : ceci va entraîner la suppression des transitions qui concernent cet état et la créations des transitions décrites dans la figure 34.

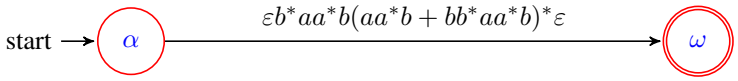


FIGURE 34 – L'automate $\mathcal{A}$ après la suppression de l'état 3.

Le langage reconnu par l'automate est donc $\varepsilon b^*aa^*b(aa^*b + bb^*aa^*b)^*\varepsilon$. Cette expression peut être simplifiée :

$$\begin{array}{rcl} aa^*b + bb^*aa^*b & = & (\varepsilon + bb^*)aa^*b \\ & = & (\varepsilon + b^+)a^+b \\ & = & b^*a^+b \end{array}$$

donc :

$$\begin{array}{rcl} \varepsilon b^*aa^*b(aa^*b + bb^*aa^*b)^*\varepsilon & = & b^*a^+b(b^*a^+b)^* \\ & = & (b^*a^+b)^+ \\ & = & (b^*a^+b)^*(b^*a^+b) \\ & = & ((b^*a^+b)^*b^*)(a^+b) \\ & = & (a^*b)^*a^+b \\ & = & (a^*b)^*a^*ab \\ & = & (a+b)^*ab \end{array}$$

Finalement, le langage reconnu par l'automate est : $(a+b)^*ab$.

8.3 Théorème de Kleene

Le théorème de Kleene affirme qu'il y a équivalence entre langage réguliers et automates à états finis : les langages exprimés par les expressions régulières sont exactement ceux qui son reconnus par les autoamtes à états finis.

Théorème 4. *(Théorème de Kleene)* $Rec(\Sigma^*) = Rat(\Sigma^*)$

Démonstration. La démonstration se fait en deux étapes : $Rec(\Sigma^*) \subseteq Rat(\Sigma^*)$ et $Rat(\Sigma^*) \subseteq Rec(\Sigma^*)$.

- $Rec(\Sigma^*) \subseteq Rat(\Sigma^*)$: à chaque automate, on peut fournir une expression rationnelle qui décrit le langage qu'il reconnait. Par exemple l'algorithme Brzozowski - McCluskey permet celà.
- $Rat(\Sigma^*) \subseteq Rec(\Sigma^*)$: à chaque expression rationnelle, on peut fournir un automate qui reconnait le langage exprimé. L'algorithme de Thompson permet celà.

□

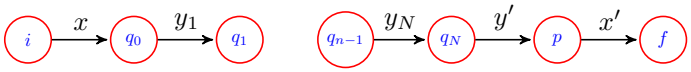
8.4 Le lemme de l'itération (ou lemme de l'étoile)

Dans ce qui précède, nous avons vu comment passer d'un automate à une expression régulière ou d'une expression régulière à un automate. Une question importante qui se pose est de savoir avant d'essayer de construire un automate qui reconnait un langage donné si un tel automate existe. Le lemme de l'étoile est une condition nécessaire mais, malheureusement, pas suffisante pour qu'un langage soit reconnaissable : si un langage $\mathcal{L}$ ne vérifie pas le lemme de l'étoile, alors on peut affirmer qu'il n'est pas reconnaissable ; s'il le vérifie, on ne peut rien conclure.

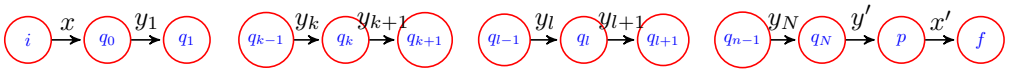
Lemme 4. *(Lemme de l'étoile). Si $\mathcal{L}$ est un langage régulier reconnu par un automate à N états, alors pour tout mot $m \in \mathcal{L}$ tel que $|m| \geq N$ et pour toute factorisation $m = xyx'$ telle que $|y| > N$, il existe une factorisation $y = uvw$ telle que :*

- $v \neq \varepsilon$
- $\forall k \geq 0, x(uv^k w)x' \in \mathcal{L}$

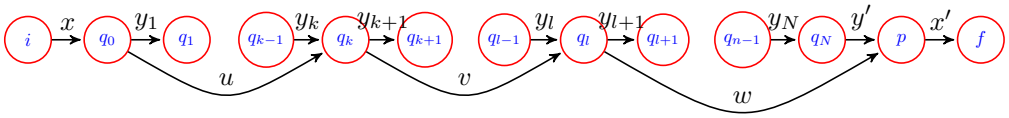
Démonstration. Soit $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$ un automate à N états qui reconnait le langage $\mathcal{L}$. Prenons un mot $m = xyx'$ de $\mathcal{L}$ avec $|m| > N$. Posons $y = y_1 y_2 ... y_N y'$ tel que $y_i \in \Sigma - \{\varepsilon\}$
$m \in \mathcal{L} \Rightarrow m$ est reconnu par $\mathcal{A}$. Il y a donc un chemin qui part d'un état initial i de I, qui arrive en un état final f de F et qui soit étiqueté par m (c'est à dire que la concaténation des lettres des arcs par lequel passe le chemin vaut m). Ce chemin peut se noter $i \xrightarrow{m} f$. Puisque $m = xyx' = xy_1y_2...y_Ny'x'$, ce chemin peut être représenté par :



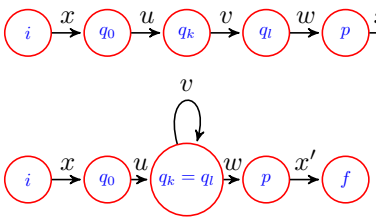
Pour aller de l'état q_0 et l'état q_N on passe par N+1 états. Or on sait que l'automate $\mathcal{A}$ possède N états. Donc parmi les N+1 états, il y en a forcéments 2 qui sont égaux. Nous appellerons q_k et q_l avec $q < l$ ces 2 états. Faisons apparaitre q_k et q_l dans le chemin qui mène de l'état initial i à l'état final f.



On a donc :



Et puisque q_k = q_l, les deux états n'en font qu'un (appelé $q_k = q_l$) et l'arc d'étiquette v boucle sur cet état. Ce qui se représente par :



□

9 Exemples

9.1 Digicodes

9.1.1 Un premier Digicode

Un premier digicode On imagine une sorte de digicode à deux touches (marquées a et b), qui débloque le bouton d'ouverture d'une porte lorsqu'on frappe n'importe quelle suite de lettres qui se termine par $abba$, et seulement l'une de celles-ci. Ainsi, les suites $aaabba$ ou $ababbba$ ou évidemment $abba$ vont convenir, mais pas les suites abb ou $aabbaba$. Par conséquent, on peut dire que, parmi toutes les suites de lettres a et b possibles, ce digicode a pour rôle de reconnaître celles qui se terminent par $abba$. Sur le plan fonctionnel, le bouton d'ouverture de la porte est bloqué lorsque l'automate se trouve dans les états 1 à 4 et débloqué dans l'état 5.

FIGURE 35 – Séquence de déblocage du digicode

Il reste à compléter les transitions correspondant aux cas où les lettres frappées ne forment pas la suite $abba$. Au départ, le système attend dans l'état 1 qu'on tape un a, qui est la première lettre de $abba$. C'est pourquoi on met une transition étiquetée b qui boucle de l'état initial sur lui-même : tant que l'utilisateur ne tape pas un a, l'état du système ne change pas (il attend toujours un a). Ensuite, dans l'état 2, on sait que la lettre a a déjà été tapée, mais tant que l'utilisateur ne tape que des a et pas un b, l'écriture de $abba$ n'avance pas : c'est la raison de la boucle étiquetée a sur cet état. Dans l'état 3, c'est la touche b qui est attendue, mais si l'utilisateur tape un a, le système retourne à l'état 2 : c'est comme si on avait recommencé à écrire $abba$ depuis le début. De même si on tape un b lorsque le système se trouve dans l'état 4 : il faut alors tout recommencer, y compris le a initial, donc on retourne à l'état 1. Enfin, on rend compte de ce qui se produit si l'utilisateur continue à taper des lettres alors que l'automate se trouve dans l'état 5 : un a envoie vers l'état 2 (la suite a a été tapée, et la suite bba est attendue), et un b envoie vers l'état 3 (la suite ab a été tapée, et la suite ba est attendue). Finalement, voici l'automate qui modélise le comportement de ce premier digicode :

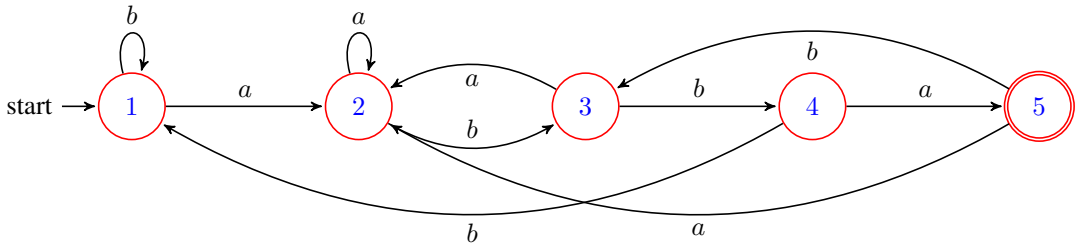


FIGURE 36 – Automate de déblocage du digicode

9.1.2 Un second Digicode

On s'intéresse maintenant à un digicode de même type, qui débloque le bouton d'ouverture de la porte pour toutes les suites de lettres a et b qui contiennent (au moins) deux a consécutifs, et seulement pour celles-ci. Ainsi, les suites $abaab$, ou $aaaaa$ ou $bbabaabaab$ vont convenir, mais pas les suites $ababab$ ou $bbba$. Tout d'abord, la suite aa doit être reconnue. On commence donc avec le schéma ci-dessous :

Puis on complète l'automate avec les autres transitions. Notons que, dès que la suite aa a été tapée une fois, l'utilisateur doit pouvoir continuer à taper d'autres lettres sans que le bouton d'ouverture de la porte

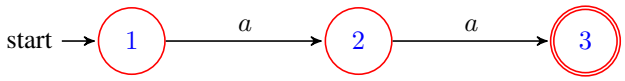


FIGURE 37 – Séquence de déblocage du digicode2

ne se bloque de nouveau : c'est le sens de la boucle étiquetée a, b sur l'état terminal de l'automate.

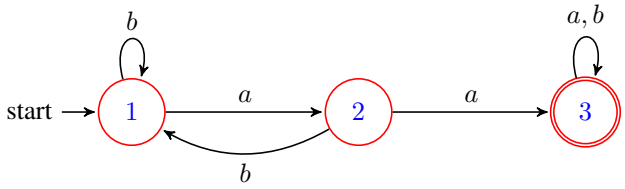


FIGURE 38 – Automate de déblocage du digicode2

9.1.3 Un troisième Digicode

Pour terminer cette série d'exemples introductifs, on modélise le comportement d'un digicode à deux touches qui débloque le bouton d'ouverture de la porte pour toutes les suites de lettres a et b qui commencent par ab, et seulement pour celles-ci. Ainsi, les suites $abaab$, ou abb vont convenir, mais pas les suites baa ou $aaba$. Pour commencer, la suite ab doit évidemment être reconnue. De plus, dès que la suite ab a été tapée, l'utilisateur doit pouvoir continuer à taper d'autres lettres sans que le bouton d'ouverture de la porte ne se bloque de nouveau : comme dans le cas précédent, on modélise ce phénomène par une boucle étiquetée par a, b sur l'état terminal de l'automate. On obtient à ce stade le schéma ci-dessous :

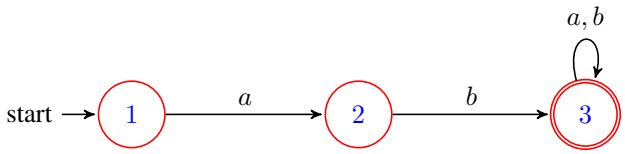


FIGURE 39 – Automate de déblocage du digicode3

On observe que deux événements n'apparaissent pas sur ce schéma : frapper un b lorsque le système se trouve dans l'état 1 et frapper un a dans l'état 2. Ces deux événements mènent à une impasse, puisqu'aucune suite commençant par b ou par aa ne permet de débloquer le bouton d'ouverture de la porte. Il y a deux façons d'en tenir compte sur l'automate. La première consiste simplement à ne pas dessiner les transitions interdites : on se contente alors du schéma ci-dessus. Dans ce cas, par exemple, la suite de lettres aab n'est pas reconnue, puisqu'elle « se bloque » après la première lettre, et ne permet donc pas d'atteindre l'état terminal. La deuxième méthode pour tenir compte de ces impasses consiste à introduire un puits, dont on ne peut pas s'échapper, et vers lequel on envoie les transitions interdites. On obtient alors l'automate ci-dessous :

9.2 Vérificateur de format d'horaire

Un des domaines où la modélisation à l'aide d'automates finis est très utilisée est ce qu'on appelle l'algorithmique du texte au sens large : recherche documentaire, traitement de texte, génomique, compilation, etc. On parle alors de recherche de motifs ou d'analyse syntaxique selon les domaines d'application. On détaille ci-dessous un exemple simple utilisé dans les activités proposées pour les élèves. Dans ce paragraphe, les noms des états sont des lettres, pour ne pas créer de confusion avec les étiquettes dont certaines sont des chiffres.

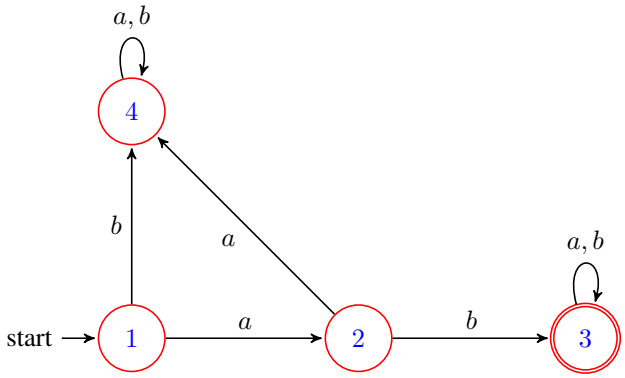


FIGURE 40 – Automate de déblocage du digicode3

On s'intéresse à un formulaire sur un site internet qui demande d'entrer un horaire sous la forme HH :MM et vérifie si la valeur donnée est correcte avant de donner accès à la touche Entrée. Les horaires acceptables vont de 00 : 00 à 23 : 59, par conséquent des expressions comme 15 : 44 ou 08 : 00 sont correctes, mais pas 5 : 12, ni 42 : 05, ni 06 : 75, ni 04 : 255. Le comportement du programme de vérification de la valeur donnée par l'internaute peut être modélisé par l'automate ci-dessous.

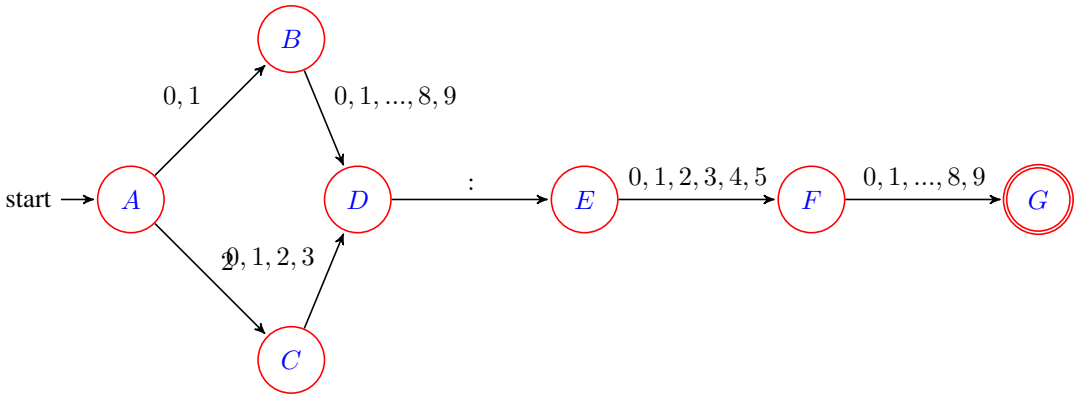


FIGURE 41 – Automate des horaires (version 1)

Détaillons le schéma : il faut impérativement deux chiffres, suivis du symbole :, puis encore deux chiffres. Le premier chiffre (celui des dizaines des heures) ne peut prendre que les valeurs 0, 1 ou 2. Si c'est un 0 ou un 1, alors le chiffre des unités correspondant peut prendre n'importe quelle valeur entre 0 et 9, mais si c'est un 2, alors les seuls chiffres des unités possibles sont 0, 1, 2 et 3. Ensuite, le symbole : est obligatoire. Pour les minutes, les seuls chiffres des dizaines possibles vont de 0 à 5, tandis que le chiffre des unités est quelconque.

Pour aboutir à un modèle plus réaliste, il nous reste à tenir compte du fait que le clavier de l'ordinateur possède d'autres touches que les chiffres et le symbole :, d'une part, et que le formulaire propose probablement une possibilité d'effacer en cas de fausse manoeuvre, d'autre part. On introduit un état supplémentaire symbolisant le fait qu'une touche non autorisée vient d'être frappée, et dont on ne peut sortir qu'en cliquant sur effacer, ce qui ramène à l'état initial du système (il faut donc recommencer à frapper son horaire). Pour ne pas trop compliquer le schéma, on a simplement étiqueté les transitions avec tout pour indiquer que toutes les touches ont le même effet et avec autre pour indiquer que toutes les touches autres que celles indiquées ont le même effet. On aboutit alors à l'automate suivant :

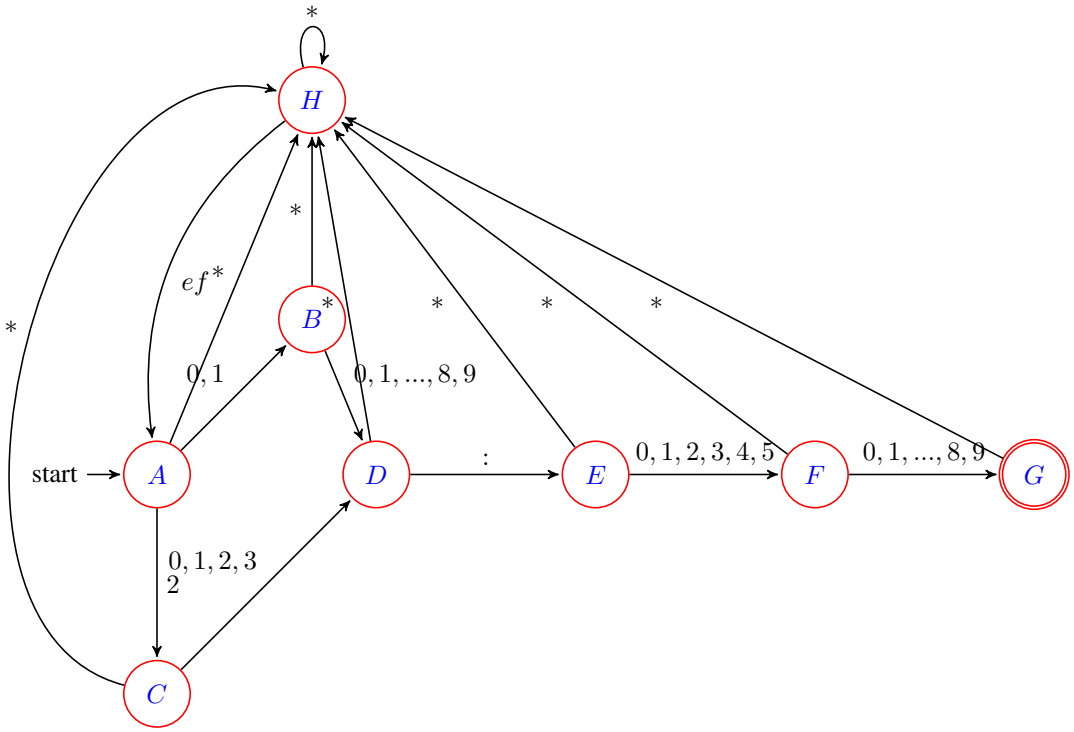


FIGURE 42 – Automate des horaires (version 2)

9.3 Vérificateur de format de dates

On obtient bien sûr un automate très similaire au précédent si on s'intéresse à la vérification d'une date (au format JJ/MM/AA par exemple) au lieu d'un horaire. Si la nature était coopérative et si tous les mois avaient 31 jours, on obtiendrait le schéma ci-dessous, et il ne serait pas nécessaire de s'attarder.

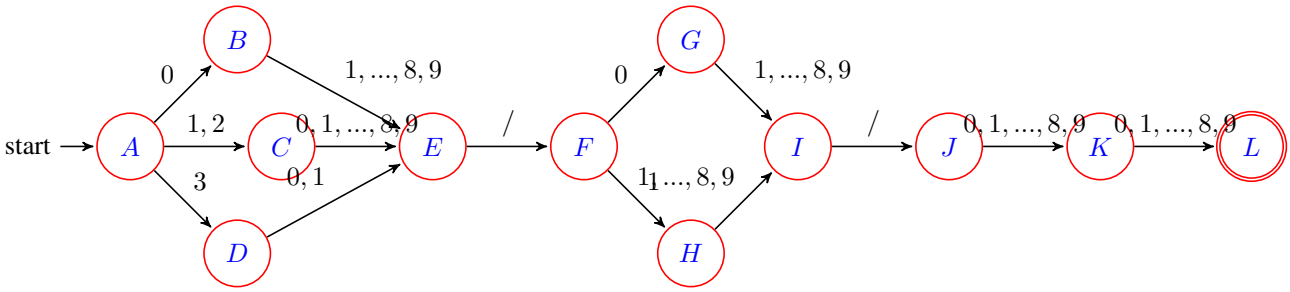


FIGURE 43 – Automate des horaires (version 2)

Cependant, le calendrier réel est bien plus compliqué. Il faut tenir compte des mois qui ont 30 jours, du mois de février qui n'a que 28 jours, et 29 jours les années bissextiles. L'année 2000 étant bissextile, contrairement aux autres années multiples de 100 (1900 et 2100 par exemple), on va se limiter à un calendrier valable du 1er janvier 2000 au 31 décembre 2099.

Pour comprendre ce schéma, on repère les passages obligés par les signes / qui correspondent à la séparation entre les jours et les mois d'une part (entre les états F et J, G et K, H et L et I et M) et entre les mois et les années d'autre part (entre les états V et X et W et Y). La partie la plus simple de l'automate est celle qui va de l'état A à l'état G, puis de l'état K à l'état W et enfin de l'état Y à l'état AC. Il s'agit du cas le plus général : les jours 01 à 28 se produisent pour les mois 01 à 12 et les années 00 à 99. Tout

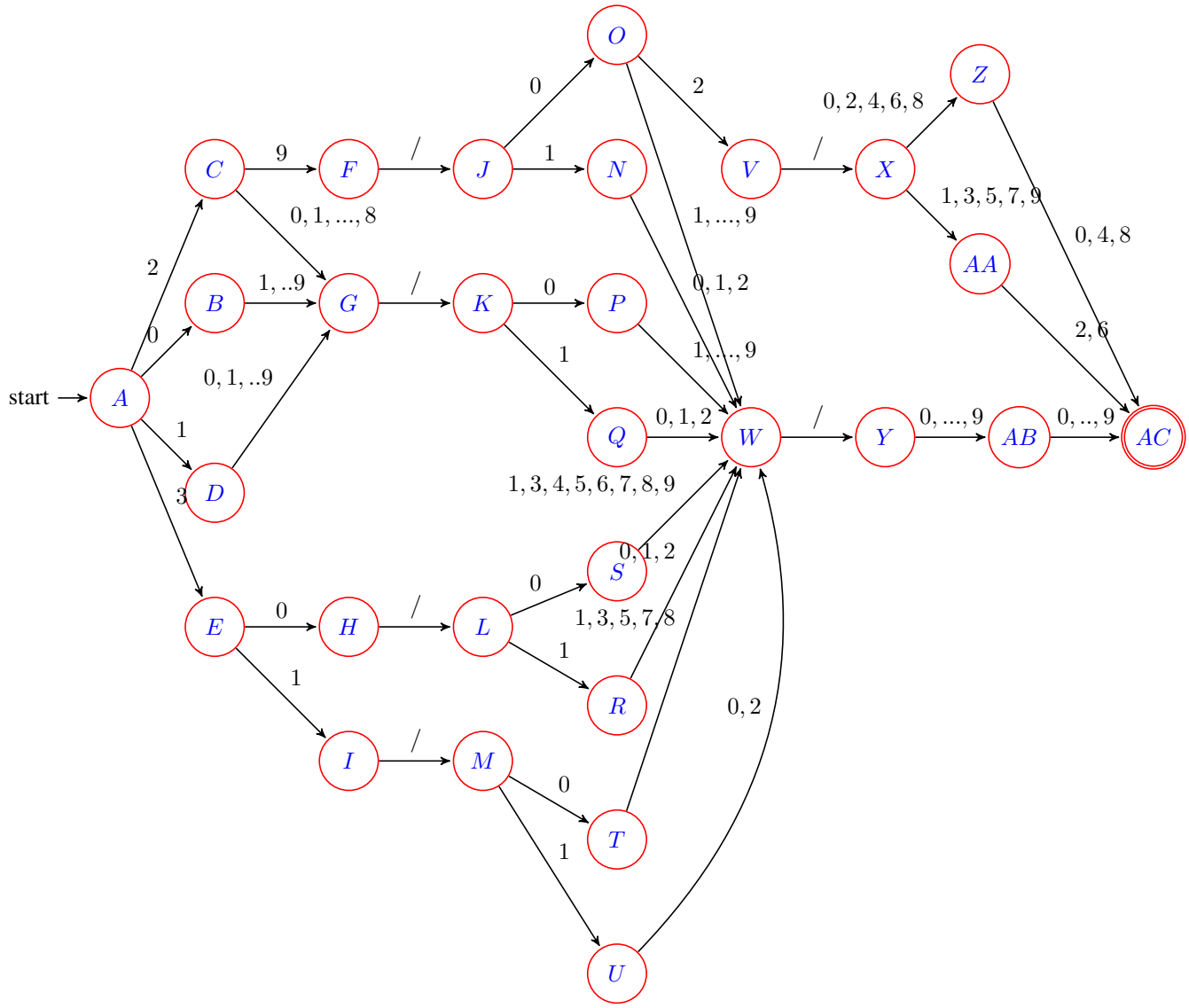


FIGURE 44 – Automate des horaires (version 2)

le reste de l'automate correspond au traitement des cas particuliers. Le numéro de jour 30 se produit pour tous les mois sauf celui de numéro 02, et ceci pour n'importe quelle année : c'est la partie qui part de l'état A, et atteint l'état W en passant par l'état H, puis rejoint l'état AC. Par contre, le numéro de jour 31 ne se produit que certains mois (passage par l'état I), et ceci pour n'importe quel numéro d'année (passage par l'état W). Il reste à traiter le numéro de jour 29 (passage par l'état F) : pour n'importe quel mois sauf celui de numéro 02, il se produit pour n'importe quelle année (état W). Pour le mois 02, on passe par l'état X, et il reste à restreindre les numéros d'années permettant d'atteindre l'état terminal AC à ceux des années bissextiles (c'est-à-dire ceux qui sont des multiples de 4).

9.4 Machine à café

De nombreuses machines mécaniques ou électroniques peuvent être représentées par des automates finis. La plupart des automates obtenus dans la vie réelle sont évidemment beaucoup plus compliqués que ceux qu'il est possible de présenter ici. À titre d'exemple supplémentaire, on donne ci-dessous un

automate correspondant à une machine à café simplifiée. Elle n'accepte que les pièces de 10 centimes et 20 centimes. Un café coûte 30 centimes. Quand le montant est suffisant (état F), elle ferme la fente permettant d'introduire des pièces, débloque un bouton permettant d'obtenir le café, rend éventuellement la monnaie et retourne dans l'état initial si on appuie sur le bouton café. Si on glisse une pièce d'un autre type que 10 ou 20 centimes (transitions autres), la machine passe dans l'état B, dont on ne peut sortir qu'en appuyant sur la touche annuler, qui déclenche le rendu des pièces et le retour à l'état initial.

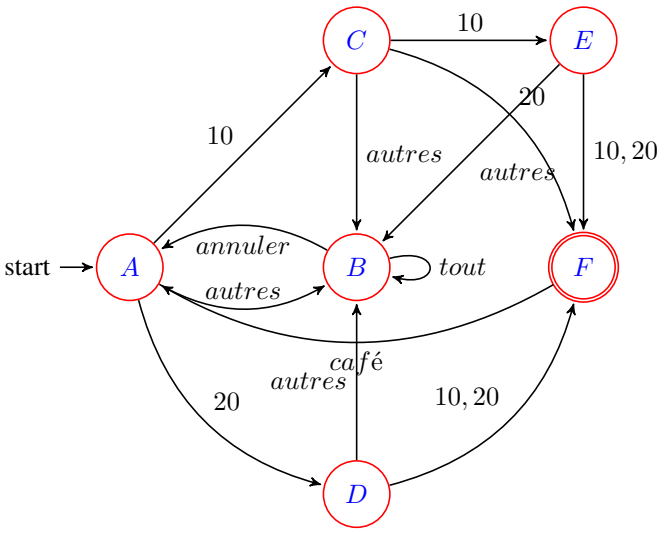


FIGURE 45 – Automate de la machine à café

9.5 Le tennis

Un autre type d'exemples de modélisation d'un système à l'aide d'un automate fini concerne des processus complètement abstraits, comme les différentes étapes du scénario d'un jeu. On illustre ce domaine par un automate modélisant l'évolution du score au cours d'un jeu pendant une partie de tennis entre Alice et Bob. Les états correspondent aux différents scores possibles (comme 40-15, ou avantage Alice), et les évènements Alice ou Bob représentent un point marqué par le joueur Alice ou Bob respectivement. On obtient le schéma ci-dessous.

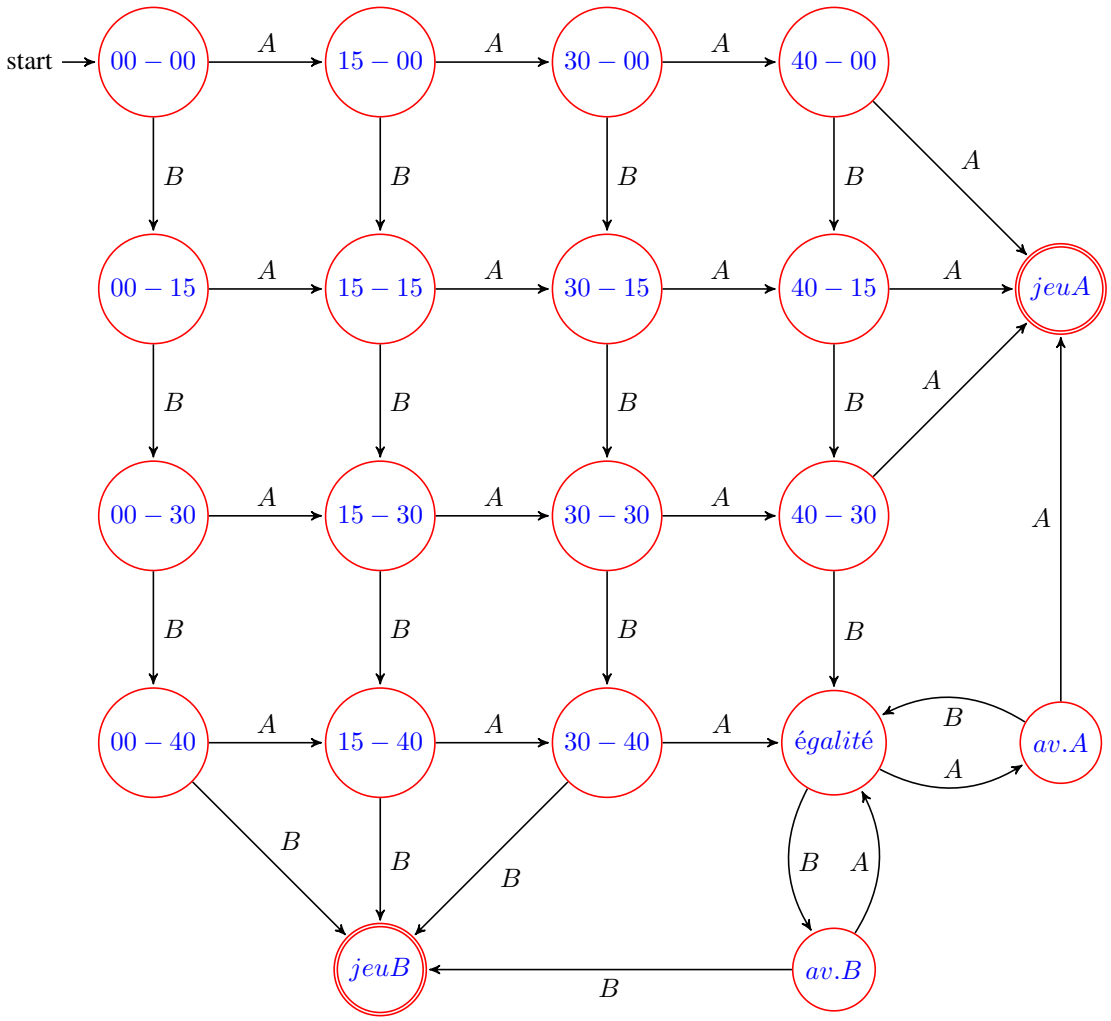


FIGURE 46 – Automate des scores d'un tennis

Table des figures

Liste des tableaux