Outils Mathématiques

Cours pour Ingénieur

Georges Edouard KOUAMOU

Motivation

- Besoin
  - écrire des programmes sous une forme plus naturelle, dans laquelle les éléments (fonctions, instructions, déclarations, expressions, etc) sont combinés selon des règles précises
- Préoccupations
  - comment écrire, à moindre frais, des compilateurs qui analysent le code source, et génèrent sa traduction ?
  - Inversement, peut-on identifier des familles de langages faciles à analyser ?
- Développements mathématiques
  - Au delà de ces motivations pratiques, la théorie des langages s'intéresse aux propriétés mathématiques des langages formels.

Domaines d’application

- Reconnaissances des expressions régulières
- Automatiser la fabrication des compilateurs, en fournissant une description du langage à reconnaitre à un méta-compilateur, qui produira un compilateur.

Positionnement

- L'informatique ne peut se réduire à la programmation. Elle ne peut pas non plus se limiter au génie logiciel
- L'informatique est la résolution des problèmes (P).
- Les mathématiques sont au cœur de la résolution de problème (Q).
  - $P \wedge Q \Rightarrow R$
  - Donner la forme littérale de R
- Définir un problème nécessite bien souvent la rigueur des mathématiques
- Le bon usage et l'analyse des modèles, des structures de données et des algorithmes nécessitent une base solide en mathématiques
- Edsger Dijkstra: « ***L'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes*** »

Sujets connexes

- Théorie des ensembles, Relations et fonctions:
  - Vu en MSP
- Logique prépositionnelle
  - Introduit en IA
  - L'application à la conception des circuits: algèbre de Boole/architecture des ordinateurs
- Science de l'information: fondements mathématiques de la sécurité informatique
  - Codage
  - Cryptographie et chiffrements

Contenu

- Mots et Langages
- Automates à états finis
  - Automate fini non déterministe
  - Automate fini déterministe
- Langages rationnels
  - Expression rationnelle
  - Langages rationnels
- Langage réguliers
  - Notion de résiduel d’un langage
  - Automate canonique (minimal)
- Grammaire algébrique et langages algébriques

Chapitre 1

Mots et langages

Définitions

- Définition (***alphabet***): Un alphabet est un ensemble fini non vide dont les éléments sont appelés des lettres.
- Exemple
  - B = {0, 1} l'alphabet binaire
  - A = {a, b, c}
  - L = {a · · · , z} l'alphabet français.
  - Le biologiste interessé par l'étude de l'ADN utilisera un alphabet à quatre lettres {A; C; G; T} pour les quatre constituants des gènes: Adénine, Cytosine, Guanine et Thymine

Mots

- Pour tout alphabet Σ, on pose Σ^* l'ensemble défini par:

$$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$$

- Un élément de Σ^* est appelé un **mot** sur l'alphabet Σ.
- Définition (***mot vide***): Le symbole ε dénote ce qu'on appelle le mot vide. Pour tout ensemble A, on considère par convention que $A^0 = \{\varepsilon\}$, l'ensemble contenant uniquement le mot vide.
  - Remarque: $\{\varepsilon\} \neq \phi$
- Un mot est donc soit le mot vide, soit un n-uplet d'éléments de Σ.
- Exemple
  - u = (a, a, b, c, a) est un mot de {a, b, c}∗
  - Par simplification, on va l'écrire u = aabca. C'est juste un changement d'écriture, les propriétés des mots sont les mêmes que celles des n-uplets

Définition des mots par recursion

- Terminologie
  - ε est le mot vide: “”
  - Σ est l’ensemble des lettres : { a, b, c, …, z }
    - Cet ensemble peut varier en function du problème
- Nous pouvons définir un ensemble de mots Σ^* comme il suit
  - Etape de base: $\varepsilon \in \Sigma^*$
  - Si $w \in \Sigma^*$ et $x \in \Sigma$, alors wx $\in \Sigma^*$
  - Ainsi, Σ^* est l’ensemble des mots possibles qui peuvent être générés avec l’alphabet
  - Est-ce dénombrablement infini ou indénombrablement infini?

Longueur d'un mot

- Définition (**longueur**): Soit u ∈ Σ*. La longueur de u est l'unique *l* tel que u ∈ Σ^l. On note $|u|$ la longueur de u.
- Pour tout n ∈ N , on note Σ^n l'ensemble des mots de longueur n.
  - Tout symbole (ou lettre) de l'alphabet Σ est un mot de longueur 1
  - Le mot vide est le mot de longueur 0
- Théorème (**égalité des mots**):
  - Deux mots $u = u_1 \dots u_n \; et \; v = v_1 \dots v_m$ sont égaux si et seulement si n = m et pour tout i ∈ {1, . . . , n}, $u_i = v_i$.
- Définition (**longueur en a**): Pour toute lettre a ∈ Σ et pour tout mot u∈ Σ*, on note $|u|_a$ la longueur en a de u, qui est le nombre d'occurrences de la lettre a dans le mot u.
- Exercice: Donner la définition récursive de la longueur d'un mot.

Longueur d'un mot par récurrence

- Définir récursivement la longueur d'un mot
  - Cas de base: $l(\varepsilon) = 0$
  - Recurrence: $l(wx) = l(w) + 1$ si $w \in \Sigma^*$ and $x \in \Sigma$
- Exemple: l("aaa")
  - l("aaa") = l("aa") + 1
  - l("aa") = l("a") + 1
  - l("a") = l("") + 1
  - l("") = 0
  - Resultat: 3

Opérations sur les mots

- Concaténation
  - Définition (**concaténation**): Soient $u = u_1 u_2 \dots u_n$ et v $= v_1 v_2 \dots v_m$ deux mots sur l'alphabet Σ. On note $u \cdot v$ (ou bien uv) la concaténation de u et de v, qui est le mot de taille n + m défini par : $uv = u_1 u_2 \dots u_n v_1 \dots v_m$
  - **Théorème**: La concaténation sur Σ^* est *associative* et admet pour *élément neutre* le mot vide ε. Elle n'est pas commutative.
  - **Théorème**: Pour tout u, v$\in \Sigma^*$ on a :
    - |uv| = |u| + |v| ; et $\forall a \in A, |uv|_a = |u|_a + |v|_a$.
- Simplification
  - Théorème (**simplification**): Si u, v, w sont trois mots tels que $uw = vw$ alors $u = v$. de même, si $wu = wv$ alors $u = v$. On dit qu'on peut simplifier à gauche et à droite
- Miroir
  - Définition (**miroir**): Le miroir d'un mot $u = u_1 \dots u_n$, noté $\tilde{u}$ est le mot $\tilde{u} = \tilde{u}_1 \dots \tilde{u}_n$ vérifiant: $\forall i \in \{1, \dots, n\}\ \tilde{u}_i = u_{n+1-i}$.
    - Exemple: $u = bonjour$ on a $\tilde{u} = ruojnob$.
  - Théorème: Pour tout u $\in \Sigma^*$, $|\tilde{u}| = |u|$ et $\tilde{\tilde{u}} = u$. Pour tout u, v $\in \Sigma^*$, $\widetilde{uv} = \tilde{v}\tilde{u}$.

Découpage des mots

- Définition (**préfixe**): On dit qu'un mot v est un préfixe d'un mot u s'il existe un mot $w \in \Sigma^*$ tel que $u = vw$.
- Définition (**suffixe**): On dit qu'un mot v est un suffixe d'un mot u sur Σ s'il existe un mot $w \in \Sigma^*$ tel que $u = wv$.
- Définition (**facteur**): On dit qu'un mot v est un facteur d'un mot u sur Σ s'il existe deux mots w, w' $\in \Sigma^*$ tel que $u = wvw'$.
- Attention
  - Dans les définitions ci-dessus, on peut très bien prendre $w = \varepsilon$ et/ou $w' = \varepsilon$.

Découpage des mots

- Définition (**sous-mot**): Soit $u = u_1 \dots u_n$ un mot de Σ^*. Un sous-mot v de u est un mot de longueur m tel qu'il existe une injection strictement croissante $\phi: \{1, \dots, m\} \to \{1, \dots, n\}$ tel que: $v = u_{\phi(1)} u_{\phi(2)} \dots u_{\phi(m)}$
- Informellement on obtient un sous-mot de u en supprimant des lettres de u et en ne changeant pas l'ordre de celles qui restent. Si u =informatique, v = faq est un sous-mot de u, mais pas w = fini.
- Théorème. Pour tout $u \in \Sigma^*$, u et ε sont toujours des préfixes, des suffixes, des facteurs et des sous-mots de u.

Ordre sur les mots

- Hypothèse: L'alphabet Σ est totalement ordonné (on donne un ordre sur les lettres).
- Définition (**ordre lexicographique**): Soient u et v deux mots de Σ, on dit que u est plus petit que v pour l'ordre lexicographique, noté $u \leq v$ ou encore $u \leq_{lex} v$ quand :
  - ou bien u est préfixe de v ;
  - ou bien il existe $a < b$ dans Σ (pour l'ordre sur l'alphabet), il existe des mots w, u', v' dans Σ^*, tels que u = wau' et v=wbv'.
- C'est l'ordre utilisé pour les dictionnaires.

Langages (1)

- Définition (**langage**): Un langage est un ensemble de mots. C'est donc un élément de $\mathcal{P}(\Sigma^*)$.
- Définition (**concaténation de langages**): Soient X et Y deux langages, la concaténation de X et de Y, notée XY est l'ensemble des concaténations des mots de X par les mots de Y:

$$XY = \{x.y | x \in X, y \in Y\}$$

- Théorème. La concaténation des langages est associative mais pas commutative. L'élément neutre est le langage réduit au mot vide {ε}.
- Définition (**miroir d'un langage**): Soit X un langage, le miroir noté $\tilde{X}$ est l'ensemble des miroirs des mots de X : $\tilde{X} = \{\tilde{x} | x \in X\}$.

Langages (2)

- Définition (**puissances**): Par convention, pour tout langage X on pose $X^0 = \{\varepsilon\}$. On définit les puissances d'un langage X, pour tout $n \geq 1$ par $X^n = X^{n-1}X$.
- Pour $n \geq 1$, $u \in X^n$ si et seulement si il peut s'écrire comme concaténation de n mots de X.
- Définition (**étoile**) Soit X un langage, l'étoile de X (on dit aussi l'étoile de Kleene de X) est le langage $X^* = \bigcup_{n \geq 0} X^n$.
- Le langage X^* contient toujours le mot vide.

Exercices

- **Exercice** 1 : Combien y a-t-il de mots de longueur p sur un alphabet à n lettres ?
  - 1. contenant au moins une occurrence d'une lettre donnée a ?
  - 2. contenant exactement une occurrence d'une lettre donnée a ?
  - 3. contenant exactement q occurrences d'une lettre donnée a ?
- **Exercice** 2. Soit l'alphabet V = {a, b} et les langages : L1={a, ab, ba} et L2={ε, b, ba}
  - 1. Donner les résultats des opérations suivantes : L1.L2, L2.L1 , L1.∅, ∅.L2, L1.{ε}, {ε}.L2, L2∩{ε}, L_1^2
  - 2. Si L3 et L4 sont deux langages tels que L3 . L4 = {ε}, que peut on dire de L3 et L4 ?
  - 3. Si L5 et L6 sont deux langages tels que L5 . L6 = ∅, que peut on dire de L5 et L6 ?
- **Exercice** 3. Soit Σ un alphabet. Montrer que Σ* est un ensemble infini dénombrable.

Chapitre 2: automates et langages reconnaissables

Un langage est dit reconnaissable s'il est reconnu par un automate fini. On introduit la notion d'automate fini et celle d'automate généralisé. On montre que tout langage reconnaissable peut être reconnu par un automate fini déterministe mais que cette transformation peut, dans le pire cas, accroitre de façon exponentielle la taille de l'automate. On étudie les propriétés de clôture des langages reconnaissables : cette famille de langage est close par les opérations booléennes (union, intersection, complémentaire et donc aussi différence ensembliste) ainsi que par l'opération de concaténation et d'itération, elle est close également par image miroir et par images directes et inverses par substitutions (d'un mot à une lettre).

Automate fini

- Un automate fini $A = (Q, q_0, \Sigma, \delta, F)$ consiste en
  - un ensemble fini Q d'états, $q_0 \in Q$ est l'état initial,
  - Σ est un ensemble fini appelé alphabet de l'automate,
  - $\delta: Q \times \Sigma \rightarrow \wp(Q)$ est la fonction de transition. ($\wp(Q) = \{X \subseteq Q\}$ désigne l'ensemble des parties de Q.)
  - $F \subseteq Q$ est l'ensemble des états finals de l'automate.
- Représentation de la fonction de transition
  - Table de transition
  - Sagittale

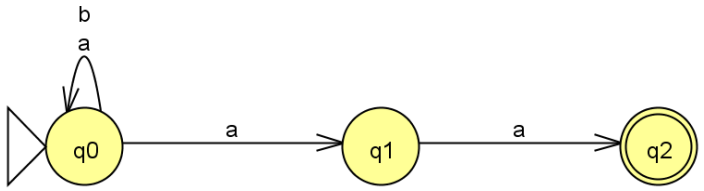
Automate fini

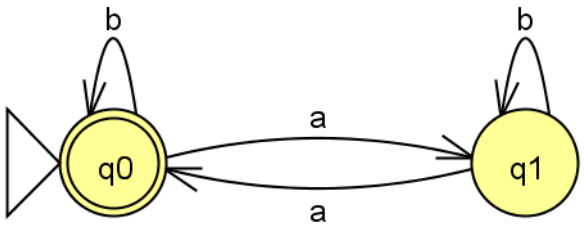
- Table de transition de l'automate, dont les lignes sont associées aux états de l'automate, les colonnes sont associées aux lettres de l'alphabet, et la case correspondant à l'état q et à la lettre a contient les états $q' \in \delta(q,a)$, c'est à dire les états atteints à partir de q par lecture de la lettre a $\in \Sigma$.
- Diagramme sagittale qui est un graphe dont les sommets sont les états et dans lequel on dispose d'un arc étiqueté a de q vers q' lorsque $q' \in \delta(q,a)$. Si $u \in \Sigma^*$ est un mot construit sur l'alphabet, on note q -- u $\rightarrow$ q′ s'il existe un chemin dans le graphe allant de q à q′ étiqueté u, ce qui est défini inductivement par:
  - q -- $\varepsilon \rightarrow$ q pour tout état q
  - q -- a $\rightarrow$ q′ lorsque $q' \in \delta(q,a)$
  - q -- au $\rightarrow$ q′ s'il existe un état q′′ tel que q -- a $\rightarrow$ q′′ et q′′ -- u $\rightarrow$ q′
- Un mot $u \in \Sigma^*$ est reconnu par l'automate s'il existe un chemin étiqueté u partant de l'état initial et conduisant à un état final, le langage reconnu par l'automate est donc défini par

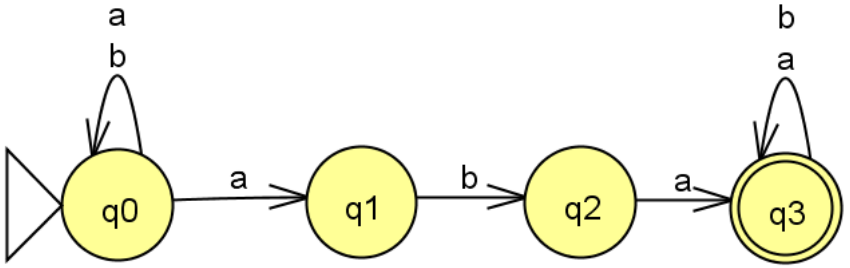
$$L(A)=\{u \in \Sigma^* | \exists q \in F \; q_0 \text{--} u \rightarrow q\}$$

Exemple

- *Construire un automate reconnaissant chacun des langages suivants sur l'alphabet Σ={a;b}*
  1. *L_1 = {u $\in\Sigma^*$ /$\exists$ v $\in\Sigma^*$ u = vaa}, c'est à dire les mots ayant pour suffixe (ou bien se terminant par) a^2.*
  2. *L_2 = { u $\in\Sigma^*$ / $\#_a(u)$ est paire} où $\#_a(u)$ désigne le nombre d'occurrences de la lettre a dans le mot u.*
  3. *L_3 = $\Sigma^*aba\Sigma^*$*, ie les mots ayant pour facteur aba







Automates déterministes

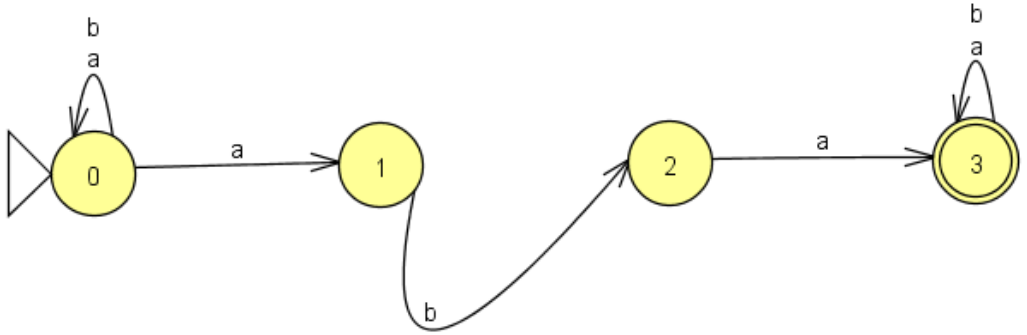
- Un automate est dit *déterministe* si les entrées de sa table de transition contiennent au plus un élément,
  - c'est à dire si pour tout état q et toute lettre a il existe au plus un état q' tel que $(q,a,q') \in \delta$
- Un automate est dit *complet* si les entrées de sa table de transition contiennent au moins un élément, c'est à dire si pour tout état q et toute lettre a il existe au moins un état q' tel que $(q,a,q') \in \delta$
- **Remarque**:
  - Il est toujours possible de rendre un automate complet par adjonction d'un état supplémentaire $\perp$, appelé *état puits*
  - pour tout état q et toute lettre a tels qu'il n' existe pas d'état q' tel que $(q,a,q') \in \delta$ on ajoute alors la transition $(q,a,\perp)$ à δ ainsi que les transitions $(\perp,a,\perp)$ pour chaque lettre $a \in \Sigma$

Rendre un automate fini déterministe

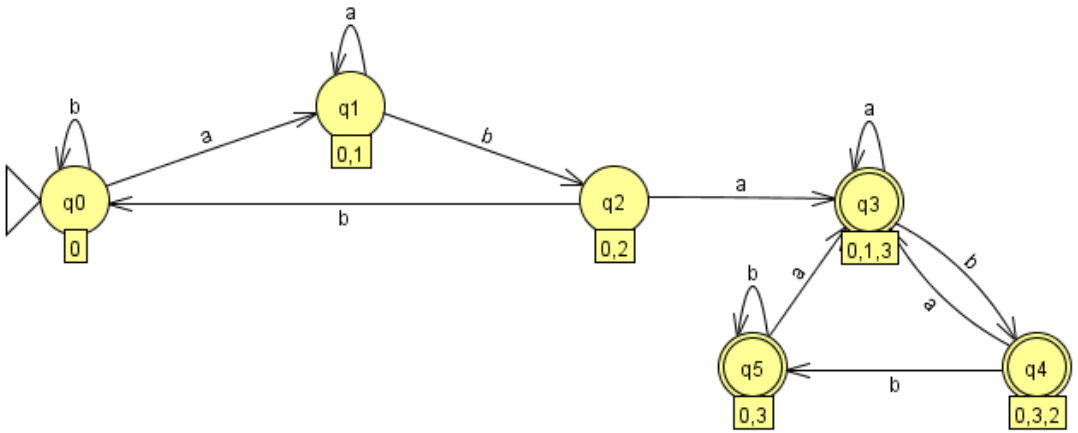
- **Proposition:** *Soit* $A = (Q, q_0, \Sigma, \delta, F)$ *un automate fini; l'automate* ***det****(A)* $= (\wp(Q), \{q_0\}, \delta', F')$ *dans lequel:*
  - $\wp(Q) = \{X \subseteq Q\}$ *est l'ensemble des parties de Q*
  - $Y \in \delta'(X,a) \Leftrightarrow Y = \{q \in Q \ / \ \exists p \in X \quad q \in \delta(p,a)\}$
  - $F' = \{X \subseteq Q \quad / \quad X \cap F \neq \emptyset\}$
  - *est déterministe, complet et reconnait le même langage que A.*
- Cas défavorable
  - Det(A) peut avoir jusqu'à $2^{card(Q)}$ états

Exemple

$L_3 = \Sigma^* aba \Sigma^*$, ie les mots ayant pour facteur aba



Automate fini non déterministe



Automate fini déterministe correspondant

Propriétés de clôture

- Un langage $L \subseteq \Sigma^*$ est dit *reconnaissable* s'il s'agit de l'ensemble des mots reconnus par un automate fini sur l'alphabet Σ.
- **Proposition**. L'ensemble des langages reconnaissables est clos par union, intersection, complémentaire, concaténation et étoile
- Preuve (Complémentaire)
  - Soit $A=(Q,q_0,\Sigma,\delta,F)$ un automate déterministe complet reconnaissant L le langage complémentaire $L^c=\Sigma^*\backslash L$ est reconnu par l'automate $A^c=(Q,q_0,\Sigma,\delta, Q\backslash F)$

Propriété de clôture

- Preuve (Intersection)
  - Soient L_1 et L_2 les langages reconnus par deux automates $A_1=(Q_1,q_{0,1},\Sigma,\delta_1,F_1)$ et $A_2=(Q_2,q_{0,2},\Sigma,\delta_2,F_2)$ alors l'automate $A=(Q,q_0,\Sigma,\delta,F)$ défini comme il suit reconnait $L_1 \cap L_2$
  - $Q = Q_1 \times Q_2 = \{(q_1,q_2) \mid q_1 \in Q_1 \text{ et } q_2 \in Q_2\}$
  - $q_0 = (q_{0,1},q_{0,2})$
  - $(q'_1,q'_2) \in \delta((q_1,q_2),a) \Leftrightarrow q'_1 \in \delta_1(q_1,a)$ et $q'_2 \in \delta_2(q_2,a)$
  - $F = F_1 \times F_2$
- Preuve (Union)
  - Approche 1: $L_1 \cup L_2 = (L^c_1 \cap L^c_2)^c$.
  - Approche 2: Une construction directe d'un automate (généralisé) reconnaissant $L_1 \cup L_2$ à partir d'automates reconnaissant L_1 et L_2 consiste à juxtaposer une copie de chacun de ces automates, puis à ajouter un nouvel état i qui devient l'état initial avec deux transitions à vide $(i,\varepsilon,q_{0,1})$ et $(i,\varepsilon,q_{0,2})$ de cet état vers chacun des états initiaux des deux automates

Propriétés de clôture

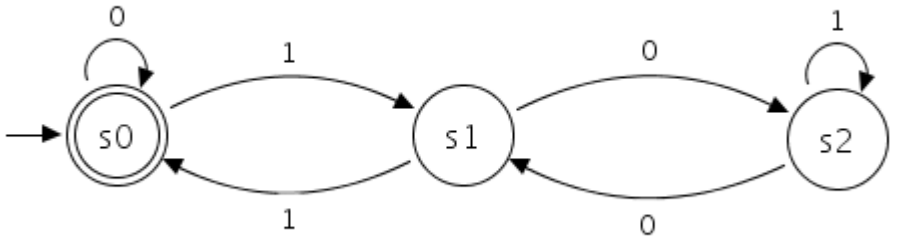
- Preuve (concaténation)
  - Si L_1 et L_2 sont des langages reconnus par des automates A_1 et A_2 leur concaténation $L_1 \cdot L_2$ est reconnu par l'automate (généralisé) obtenu par juxtaposition des automates A_1 et A_2 auxquels on ajoute les transitions à vide $(q,\varepsilon,q_{0,2})$ de chacun des états finals de A_1 vers l'état initial de A_2, avec comme état initial l'état initial $q_{0,1}$ de A_1 et comme états finals ceux de A_2.
- Preuve (Etoile)
  - Si L est un langage reconnu par un automate A, son étoile L^* est reconnu par l'automate obtenu à partir de A par ajout d'un nouvel état i et de transitions à vide (q,ε,i) de chacun des état finals q de A vers i et d'une transition à vide de i vers l'état initial de A, i est l'état initial de cet automate et ses états finals sont ceux de A auxquels on ajoute l'état i.

Exercices

1. Donner un automate a états finis qui reconnait les mots sur l'alphabet {a,b} qui se terminent par « aab ».
   - Rendre l'automate obtenu déterministe
2. Donner un automate à états finis qui accepte les mots sur l'alphabet {a,b} qui commence par b, se termine par b et ayant a* en facteur i.e les mots de la forme *b(a)*b*
   - *Rendre l'automate obtenu déterministe s'il ne l'est pas*
3. Donner un automate à états finis déterministe acceptant les nombres binaires qui contiennent la sous-chaîne « 011 »
4. Donner un automate à états finis déterministe acceptant les nombres binaires qui ne contiennent pas la sous-chaîne « 011 »

Exercices

- *Construire un automate reconnaissant chacun des langages suivants sur l'alphabet Σ={a;b}*
  1. *$L_1 = \{u \in \Sigma^* / \exists v \in \Sigma^*$ $u = vaaa$ ou $u = vbb\}$, c'est à dire les mots se terminant par a^3 ou b^2.*
  2. *$L_2 = \{ u \in \Sigma^* /$ $\#_a(u)$ est paire et $\#_b(u)$ est impaire} où $\#_a(u)$ désigne le nombre d'occurrences de la lettre a dans le mot u.*
  3. *$L_3 = \Sigma^* aba \Sigma^*$*
- Construire un automate fini qui accepte les suites binaires qui représentent des nombres divisibles par 3



Exercices

- Construire un automate déterministe complet équivalent à chacun des automates suivants sur l'alphabet {a; b}

Q = {p,q,r}, l'état initial est p, r est final et la table de transition est

	a	b
p	p,q	p
q		r
r		

	a	b
p	q, r	p
q	s	
r		r, s
s	s	r

Q = {p,q,r,s}, l'état initial est p, s est final et la table de transition est

Langages rationnels

On introduit la notion d'expression rationnelle et donne la preuve du théorème de Kleene qui indique qu'un langage est rationnel, c'est à dire représentable par une expression rationnelle, si, et seulement si, il est reconnaissable. Ce théorème est constructif ce qui signifie que l'on dispose d'algorithmes permettant de construire un automate à partir d'une expression rationnelle, et inversement d'extraire une expression rationnelle représentant le langage d'un automate fini. On introduit ensuite les expressions rationnelles étendues qui sont des notations abrégées pour des expressions rationnelles.

Les expressions rationnelles

- Soient
  - Σ un alphabet
  - 0 et 1 deux constantes n'appartenant pas à l'alphabet
  - un opérateur unaire * et un opérateur binaire + qui seront respectivement associés aux opérations d'itération et d'union sur les langages
- Définition inductive (**ER(Σ)** ensemble des expressions rationnelles associées à un alphabet)
  - $\forall a \in \Sigma \quad a \in ER(\Sigma)$
  - $0,1 \in ER(\Sigma)$
  - $\forall e_1, \ e_2 \in ER(\Sigma) \quad e_1 + e_2 \in ER(\Sigma)$
  - $\forall e_1, \ e_2 \in ER(\Sigma) \quad e_1 \cdot e_2 \in ER(\Sigma)$
  - $\forall e \in ER(\Sigma) \quad e^* \in ER(\Sigma)$

Langages rationnels

- Définition inductive (Le langage $L(e) \subseteq \Sigma^*$ associé à une expression rationnelle $e \in ER(\Sigma)$)
  - $\forall a \in \Sigma \quad L(a) = \{a\}$
  - $L(0) = \varnothing$ et $L(1) = \{ \varepsilon\}$
  - $\forall e_1, \quad e_2 \in ER(\Sigma) \quad L(e_1 + e_2) = L(e_1) \cup L(e_2)$
  - $\forall e_1, \quad e_2 \in ER(\Sigma) \quad L(e_1 \cdot e_2) = L(e_1) \cdot L(e_2)$
  - $\forall e \in ER(\Sigma) \quad L(e^*) = L(e)^*$
- Un langage est dit *rationnel* s'il peut être représenté par une expression rationnelle
  - $L \in \mathbf{Rat}(\Sigma^*) \Leftrightarrow \exists e \in ER(\Sigma) \quad L = L(e)$

Exemples

- Sur l'alphabet $\Sigma = \{a, b\}$, voici quelques exemples d'expressions régulières
- $e_1 = \big(1 + (a.b)\big)$
- $e_2 = \big((a.b).a + b^*\big)^*$
- $e_3 = \big((a+b)^*.(a.b)\big)$

- Les langages associés à chacune des ER
- $L(e_1) = \{\epsilon, ab\}$
- $L(e_2) = (\{aba\} \cup b^*)^*$
- $L(e_3) = \{a, b\}^*\{ab\}$
- Question: Donner l'expression littérale de chaque langage

Remarque:

- ✓ *Dans la suite, on s'autorisera a confondre une expression régulière et le langage qu'elle représente.*
- ✓ *Si aucune confusion n'est possible, on s'autorisera également a enlever les parenthèses ou autres symboles superflus.*

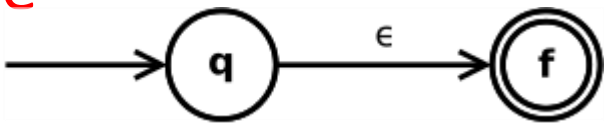
Langages rationnels et langages reconnaissables

- Théorème (dit de Kleene)
  - Un langage $L \subseteq \Sigma^*$ est rationnel si, et seulement si, il est reconnaissable.
- Preuve
  - Comme $\varnothing$, $\{\varepsilon\}$ et $\{a\}$ pour toute lettre $a \in \Sigma$ sont des langages reconnaissables et que par ailleurs la classe des langages reconnaissables est close par union, concaténation et étoile on déduit que tout langage rationnel est reconnaissable. Inversement supposons que $L \subseteq \Sigma^*$ soit un langage reconnaissable
    - **L'algorithme de Thompson permet d'aller de l'expression rationnelle à l'automate**
  - Inversement, construire un automate fini à partir d'une ER. La preuve est constructive, mais nous verrons plutôt la méthode directe

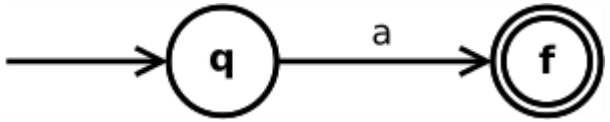
Construction inductive de l'automate

Cas de base

Expression ε

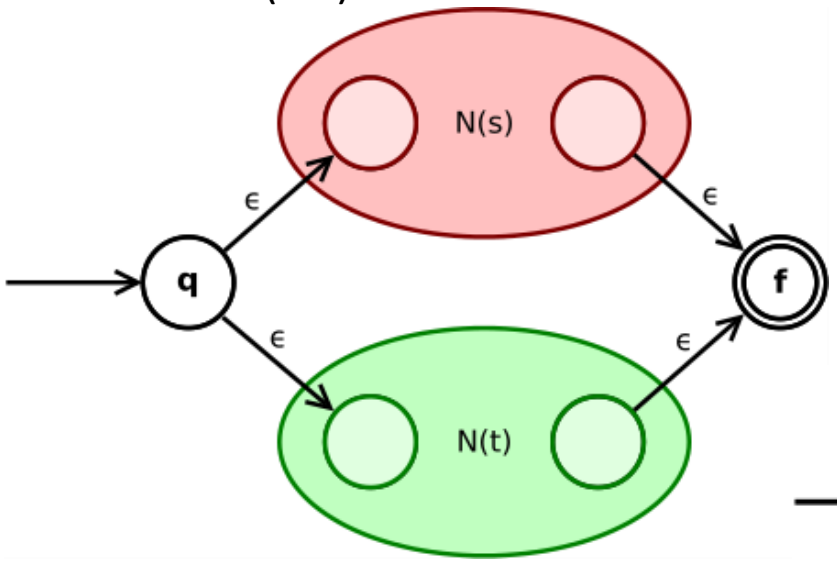


Expression a où a est une lettre

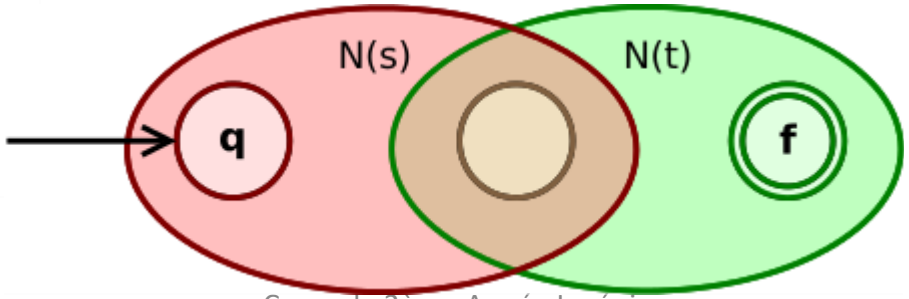


Cas inductifs

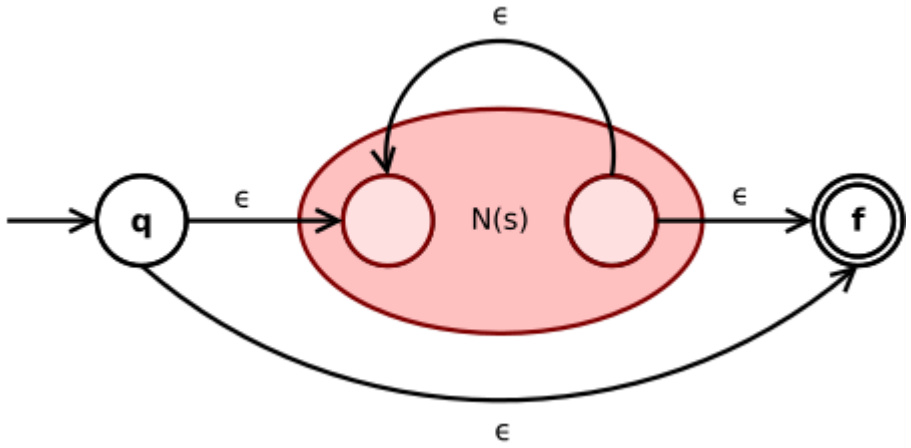
Union (s+t)



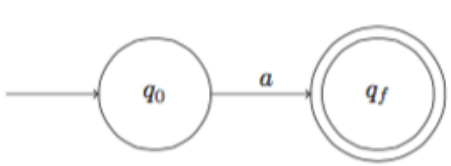
Concaténation s.t

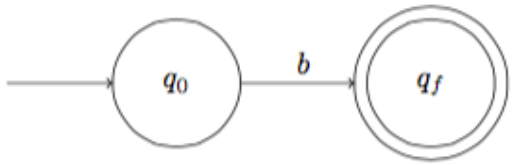


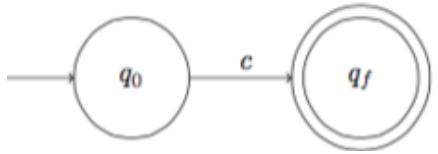
Etoile de Kleene s*

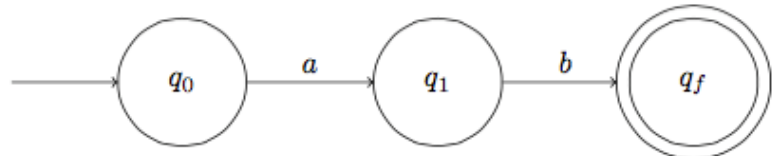


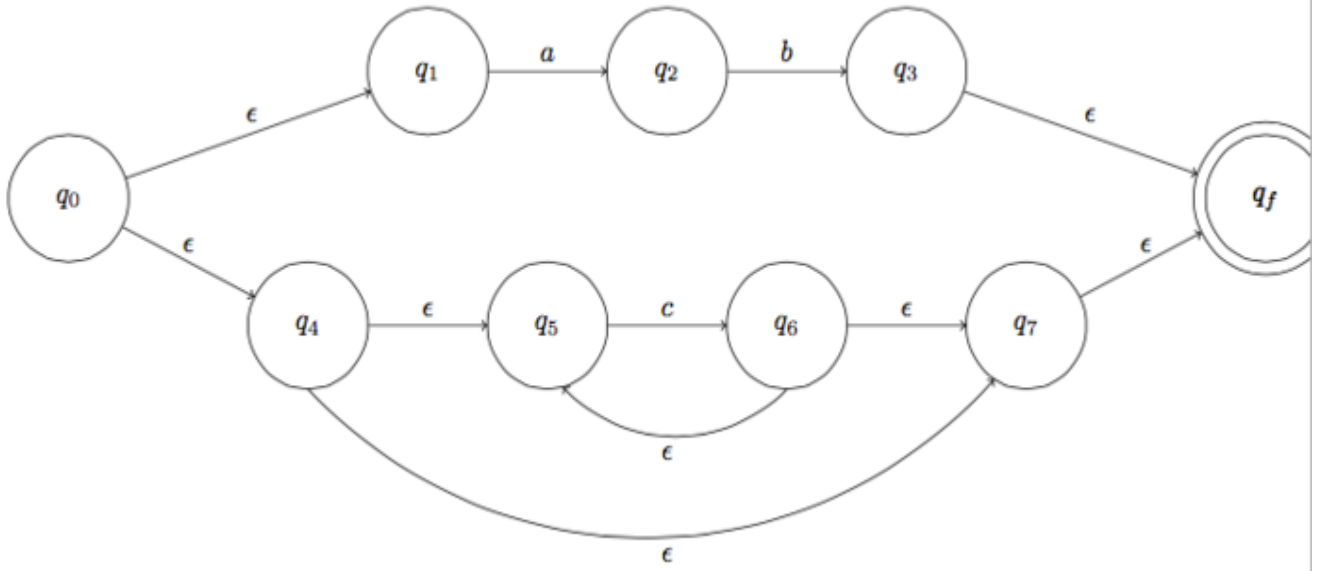
Exemple $e = (ab + c^*)$

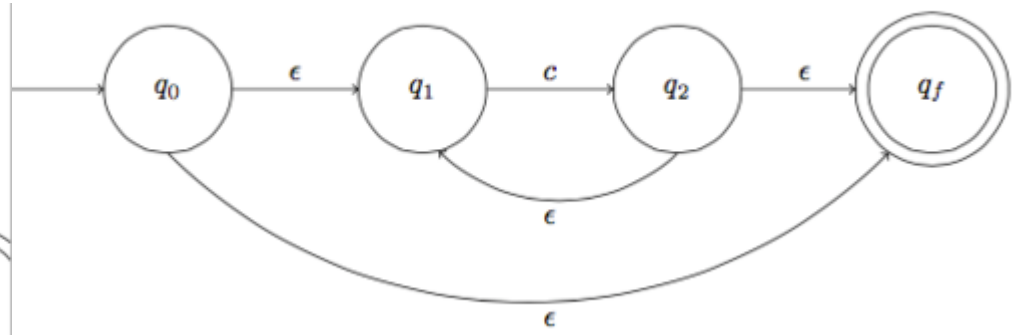












Etape 1. on construit l'automate qui reconnait {a}
Etape 2. on construit l'automate qui reconnait {b}
Etape 3. on construit l'automate qui reconnait {c}
Etape 4. Par concaténation de 1 et 2, on construit l'automate qui reconnait {ab}
*Etape 5. A partir de l'automate 3, on construit un automate reconnaissant c**
Etape 6. e est reconnu par l'union de 4 et 5

Exercice. $e' = (a + b)^* b$

Système d'équations caractéristiques

- Si $A = (Q,q_0,\Sigma,\delta,F)$ est un automate fini, on lui associe un système d'équations
  - une variable X_i pour chaque état q_i de l'automate.
  - Cette variable X_i est associée à une équation ayant l'une des deux formes suivantes :

  1. $X_i = a_1 \cdot X_{j1} + \ldots + a_k \cdot X_{jk}$ ou

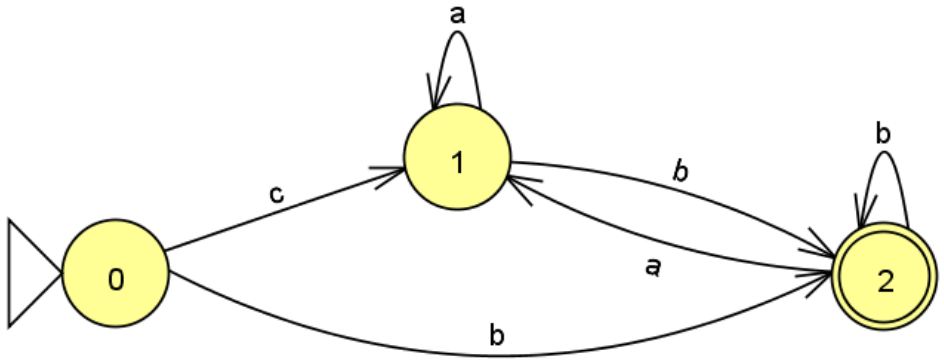
  2. $X_i = 1 + a_1 \cdot X_{j1} + \ldots + a_k \cdot X_{jk}$ si q_i est final
- Une solution du système est un ensemble de langages $L_1, \ldots L_n \subseteq \Sigma^*$ tels que
  - $L_i = a_1 \cdot L_{j1} \cup \ldots \cup a_k \cdot L_{jk}$ si $X_i = a_1 \cdot X_{j1} + \ldots + a_k \cdot X_{jk}$
  - $L_i = \{\varepsilon\} \cup a_1 \cdot L_{j1} \cup \ldots \cup a_k \cdot L_{jk}$ si $X_i = 1 + a_1 \cdot X_{j1} + \ldots + a_k \cdot X_{jk}$
- *L_i est le langage reconnu à partir de l'état q_i, en particulier L_0 est le langage de l'automate*

Equation en langages

- **Lemme 1 (Arden).** Soient A et B deux langages. **L'équation X = AX + B** a une plus petite solution, qui est **X = A*B**. Cette solution est unique si $\mathcal{E}$ n'appartient pas au langage A.
- Démonstration
  - **validité** : A*B est bien solution : $A(A^*B) + B = (AA^* + 1)B = A^*B$
  - **minimalité** : Soit L une solution de l'équation. Alors $\mathrm{L} = \mathrm{AL} + \mathrm{B}$. En remplaçant L par AL + B à droite, on a aussi $L = A(AL + B) + B = A^2L + AB + B$. En continuant ainsi, on montre (par récurrence) que $L = A^{n+1}L + \sum_{i=0}^{n} A^iB$. Ainsi, pour tout $n \geq 0, A^nB \subseteq L, et\ donc\ A^*B \subseteq L$.
  - **unicité** : si $\mathcal{E}$ n'est pas élément de A, alors soit u un mot d'un langage solution L de l'équation. Si $|u| = 0$, alors le mot vide appartient à B. Sinon, si $|u| = n$, comme on a $L = A^{n+1}L + \sum_{i=0}^{n} A^iB$, u est nécessairement dans $\sum_{i=0}^{n} A^iB$. donc $L = A^*B$.

Exemple

Soit l'automate ci-après



$$\begin{cases} X_0 = cX_1 + bX_2 \\ X_1 = aX_1 + bX_2 \\ X_2 = aX_1 + bX_2 + 1 \end{cases}$$

$$Résolution\ de\ l'équation\ 2:$$

$$X_1 = a^*bX_2$$

Substitution dans l'équation 3:

$$X_2 = 1 + (aa^*b + b)X_2$$

$$qui\ a\ pour\ solution\ X_2 = (aa^*b + b)^*$$

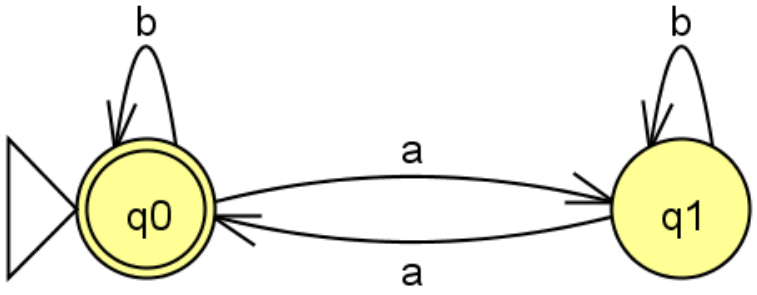
Substitution dans les 2 autres:

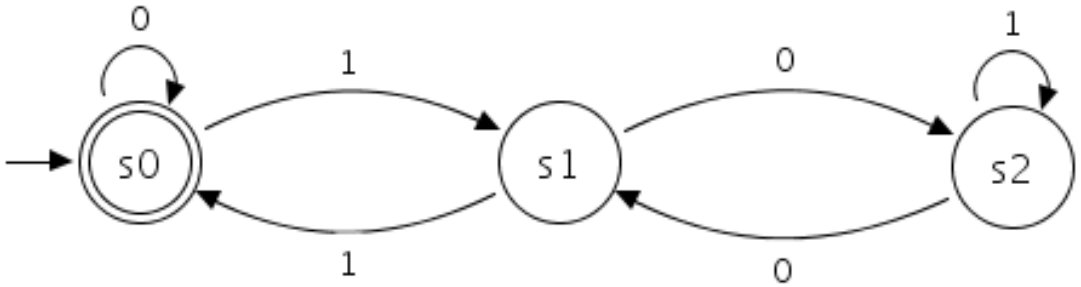
$$X_1 = a^*b(aa^*b + b)^*\ et$$

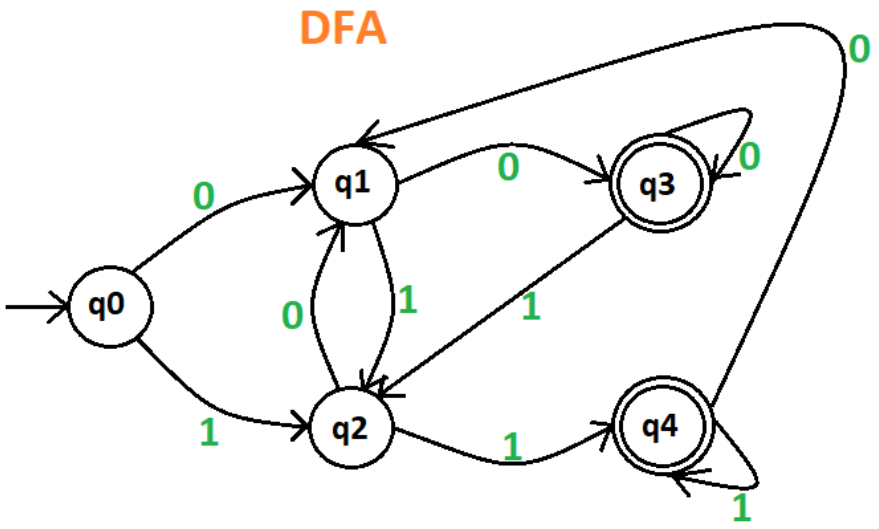
$$X_0 = (ca^*b + b)\ (aa^*b + b)^*$$

Exercices

Donner une expression rationnelle pour les automates ci-après





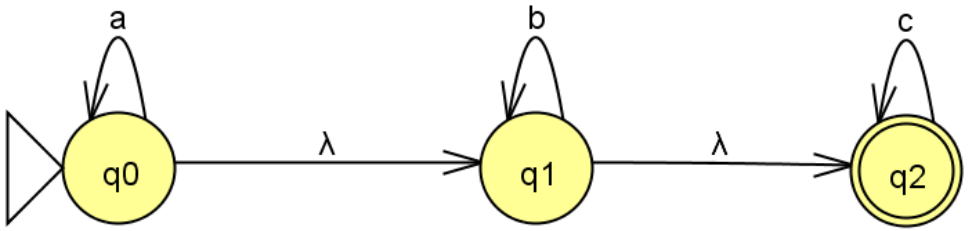


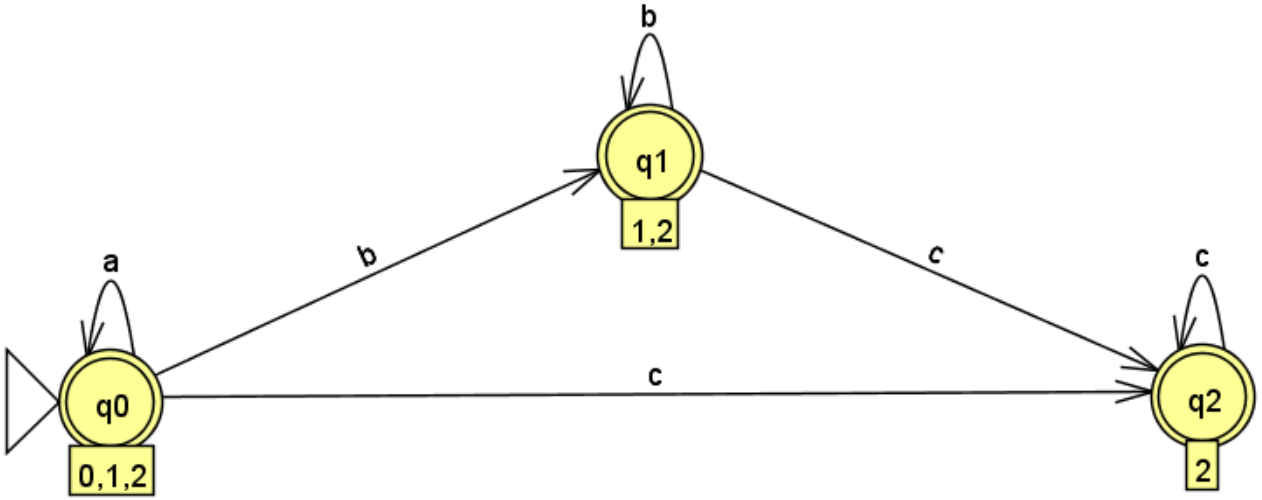
TD: Elimination des transitions spontanées

- **Définition**: Les *transitions étiquetées par le mot vide* dans un automate fini sont appelées les *transitions spontanées*.
- Un Automate fini à transitions spontanées est un automate fini (non-déterministe) noté $\varepsilon - NFA$
- **Définition** ($\varepsilon - fermeture$)
  - $\varepsilon - fermeture(p) = \{q \in Q | \delta^*(p, \varepsilon) = q\}$, i.e l'ensemble des états qu'on peut atteindre en suivant des transitions vides
  - Par construction, p $\in \varepsilon - fermeture(p)$
- **Théorème**. Pour tout $\varepsilon - NFA$ A, il existe un DFA A' qui reconnait le même langage $L(A) = L(A')$
- Démonstration
  - En posant $A = (\Sigma, Q, d, F, \delta)$ un $\varepsilon - NFA$, on définit A' comme suit: $A' = (\Sigma, Q, d', F', \delta')$ avec
  - $F' = \{q \in Q | \varepsilon - fermeture(q) \cap F \neq \emptyset\}$
  - $\delta'(p, a) = \bigcup_{q \in \varepsilon - fermeture(p)} \delta(q, a)$

Exemple

- Construire l'automate fini pour l'expression rationnelle a*b*c*
- Eliminer les transitions spontanées





Exercice.

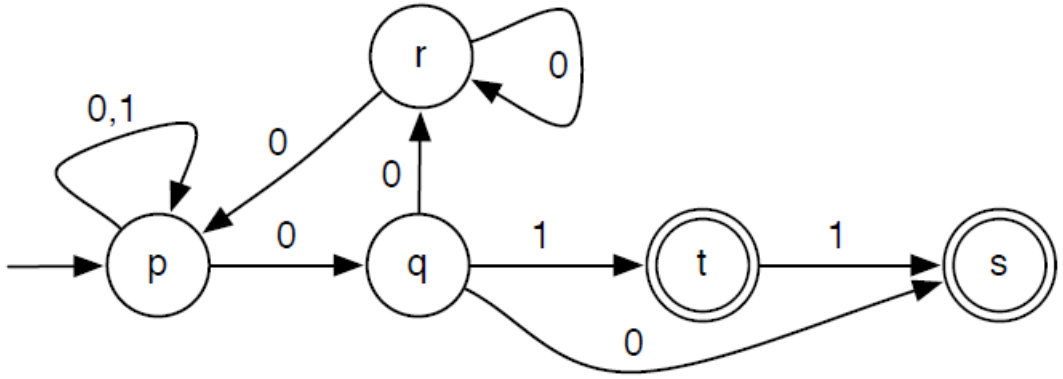
- Convertissez les expressions rationnelles suivantes en $\varepsilon - NFA$:
  1. 01*
  2. (0 + 1)01
  3. 00(0 + 1)*
- Puis éliminer les transitions spontanées

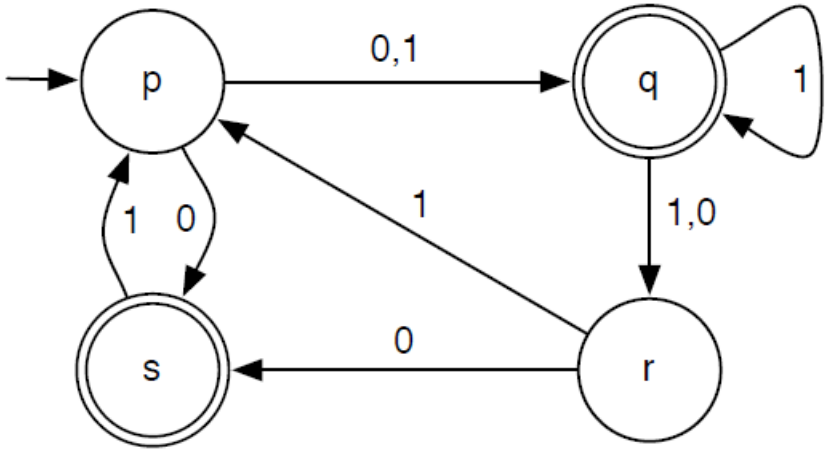
Exercices

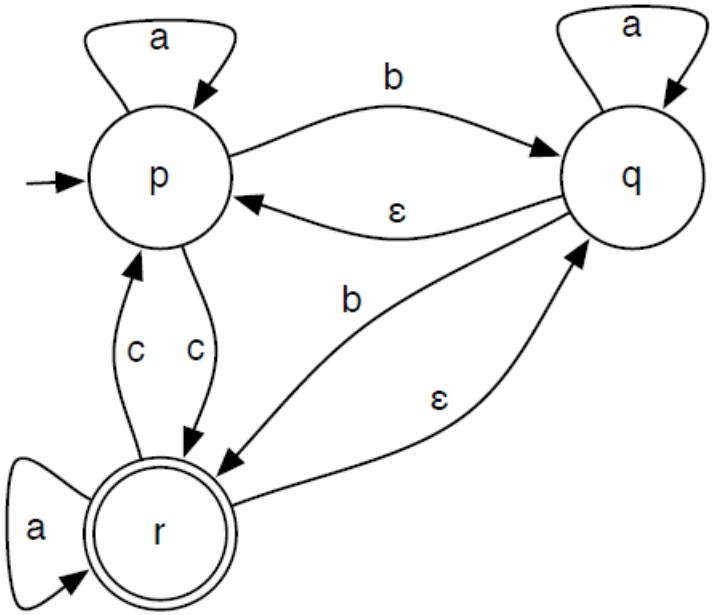
- Construire un automate fini qui accepte les suites binaires qui représentent :
  1. Des nombres pairs
  2. Des nombres multiples de 4
     - Construire le NFA, le rendre déterministe, puis minimiser
  3. Ensemble de toutes les chaînes ne contenant pas 101
- *Construire un automate sur l'alphabet Σ={a;b} reconnaissant l*e langage des mots admettant aba pour facteur
  - Construire l'automate du langage miroir
  - Donner une expression rationnelle

Travaux dirigés

Rendre déterministe les automates suivants







Langages réguliers, Automate minimal

On introduit la notion de résidu d'un langage par un mot et l'équivalence de Nerode. Un langage est dit régulier s'il n'admet qu'un nombre fini de résidus ce qui revient à dire que son équivalence de Nerode est de type fini. On montre qu'un langage est reconnaissable si, et seulement si, il est régulier. L'automate associé à l'équivalence de Nerode d'un langage reconnaissable possède une propriété remarquable intéressante : il s'agit de l'automate déterministe complet reconnaissant le langage ayant un nombre minimal d'états. Cette propriété caractérise, à isomorphisme près, cet automate qui est alors appelé automate canonique associé au langage. On définit une équivalence, dite aussi de Nerode, sur les états d'un automate fini de telle sorte que son automate canonique est obtenu en quotientant par cette équivalence. Cela nous procure un moyen pour décider si deux automates finis sont équivalents, c'est à dire s'ils reconnaissent le même langage. Par ailleurs on introduit un calcul (calcul de Brzozowski) pour les résidus d'une expression rationnelle qui permet de calculer directement l'automate canonique associé à une expression rationnelle

Notion de residuel

- **Définition**: Le résidu d'un langage $L \subseteq \Sigma^*$ par un mot $u \in \Sigma^*$ est le langage $u^{-1}L = \{v \in \Sigma^* | uv \in L\}$
  - c'est à dire qu'il s'agit de l'ensemble des mots qui peuvent être concaténés à u pour former un mot du langage.
- **Définition**. Un langage est dit régulier s'il possède un nombre fini de résidus.
- **Proposition**. Tout langage reconnaissable est régulier

Langage régulier

- *Preuve:*
  - Soit $A = (Q,q_0,\Sigma,\delta,F)$ un automate fini déterministe complet reconnaissant le langage L considéré. Pour tout mot $u \in \Sigma^*$ posons q_u l'état atteint à partir de l'état initial par lecture du mot u, i.e. q_0 -- $u \rightarrow q_u$. Un tel état existe car l'automate est complet, et il est caractérisé par cette propriété car l'automate est déterministe. Il est alors clair que $u^{-1}L = L(q_u)$ où $L_q = \{v \in \Sigma \mid \exists q' \in F \quad q \text{ -- } v \rightarrow q'\}$ désigne le langage reconnu à partir de l'état q. Comme il n'y a qu'un nombre fini détats on en déduit que L n'admet qu'un nombre fini de résidus et donc est régulier.
- **Corollaire.** *Tout résidu d'un langage reconnaissable est reconnaissable*.
  - *Preuve :* Avec les mêmes notations que dans la preuve précédente; le résidu $u^{-1}L = L(q_u)$ est reconnu par l'automate $A_u = (Q,q_u,\Sigma,\delta,F)$ obtenu à partir de l'automate de départ en remplaçant l'état initial par q_u.

Langage régulier

- **Proposition**. Tout langage régulier est reconnaissable
- Preuve: Soient L un langage régulier. Par hypothèse, il a un ensemble un ensemble fini de résidu. Notons le $Q=\{L_0, \ldots, L_n\}$
- Supposons que $L_0=\mathcal{E}^{-1}L$ i.e L lui-même; alors l'automate $\mathbf{can}(L) = (Q, L_0, \Delta, F)$ où
  - F ensemble des états finaux sont les résidus de L contenant le mot vide
  - Le relation de transition est donnée par $L_i \in \Delta(L_j, a)$ si et seulement si $L_i = a^{-1}L_j$
  - Cet automate reconnait L
- On montre en effet aisément par récurrence sur la longueur du mot u que L_i -- $u \rightarrow L_i$ si, et seulement si, $L_i = u^{-1}L_i$ une fois qu'on a observé les identités $\varepsilon^{-1}L = L$ et $(u\cdot a)^{-1}L = a^{-1}(u^{-1}L)$. Le langage de $\mathbf{can}(L)$ est alors l'ensemble des mots u tels que $\varepsilon \in u^{-1}L$ or cette dernière condition équivaut au fait que $u \in L$.
- can(L), appelé automate canonique de L
  - Il est par définition déterministe et complet.
  - Il s'agit du plus petit automate déterministe complet reconnaissant L

Equivalence de Nerode

- **Donnée**: $A = (Q, q_0, \Sigma, \delta, F)$ est un automate déterministe complet
- **Définition** (langage reconnu depuis un état q)
  - $L_q = \{u \in \Sigma^* | \delta^*(q,u) \in F\}$
  - en particulier $L(A) = L_{q0}$
- **Définition** (congruence de Nerode)
  - Deux états q et q' de A sont discernés par un mot u si $\delta^*(q,u) \in F$ et $\delta^*(q',u) \notin F$ ou le contraire; i.e le mot u est « reconnu » à partir de l'un d'entre eux mais par les deux.
  - Deux états sont dits indiscernables s'il n'y a aucun mot permettant de les discerner.
- Cette ***relation d'indiscernabilité*** est une relation d'équivalence appelée équivalence de Nerode.
- Ainsi **deux états q et q' sont équivalents** pour l'équivalence de Nerode s'ils reconnaissent le même langage: $L_q = L_{q'}$

Etats équivalents

- On pose $q \equiv_n q'$ lorsque q et q' ne sont discernés par aucun mot de longueur inférieure ou égale à n
  - $q \equiv_n q' \Leftrightarrow L_q \cap \Sigma^{\leq n} = L_{q'} \cap \Sigma^{\leq n}$
- On les identités suivantes qui nous permettent de calculer ces relations de proche en proche
  1. $q \equiv_0 q' \Leftrightarrow$ les deux états sont finaux ou non
  2. $q \equiv_{n+1} q' \Leftrightarrow \forall a \in \Sigma \quad q \cdot a \equiv_n q' \cdot a$
- si $\equiv_n = \equiv_{n+1}$ alors $\forall m \geq n \quad \equiv_n = \equiv_m$ et l'équivalence de Nerode est obtenue à cette n-ième étape

Automate quotient

- Le quotient de A par son équivalence de Nerode est obtenu en identifiant les états indiscernables.
- Il s'agit de l'automate **Nerode**(A) = ($Q_{\equiv}$, $[q_0]$, N(δ), N(F)) dans lequel:
  - $Q_{\equiv}$ = {[q] | q ∈ Q} est l'ensemble des classes d'équivalence des états pour l'équivalence de Nerode,
  - $[q_0]$ est la classe de l'état initial,
  - Y ∈ N(δ)(X,a) ⇔ ∃q ∈ X, ∃q′ ∈ Y tel que q′ ∈ δ(q,a),
  - N(F) = {[q] | q ∈ F} est l'ensemble des classes d'équivalence des états finals.
- **Théorème**: Le quotient d'un automate déterministe complet A par son équivalence de Nerode est un automate déterministe complet qui reconnait le même langage que A
- **Proposition**: $\boldsymbol{Nerode(A) \approx can(L(A))}$
  - *Le quotient d'un automate déterministe complet accessible (tous les états sont accessibles à partir de l' état initial) par son équivalence de Nerode est isomorphe à l'automate canonique du langage reconnu par cet automate.*

Problème de minimisation

- **Donnée** : A un A.F.D. complet dont chaque état est accessible depuis l'état initial
- **Problème** : construire l'automate minimal *(déterministe et complet)* qui reconnaît le même langage que A
- **Idée** : faire en sorte que les états équivalents soient fusionnés
- En pratique, l'algo. est fondé sur le principe de «séparation des états»
  - on commence par séparer les états finals des états non finals
  - dans chaque classe, on sépare les états non équivalents
  - on renouvelle cette opération jusqu'à la stabilisation ...

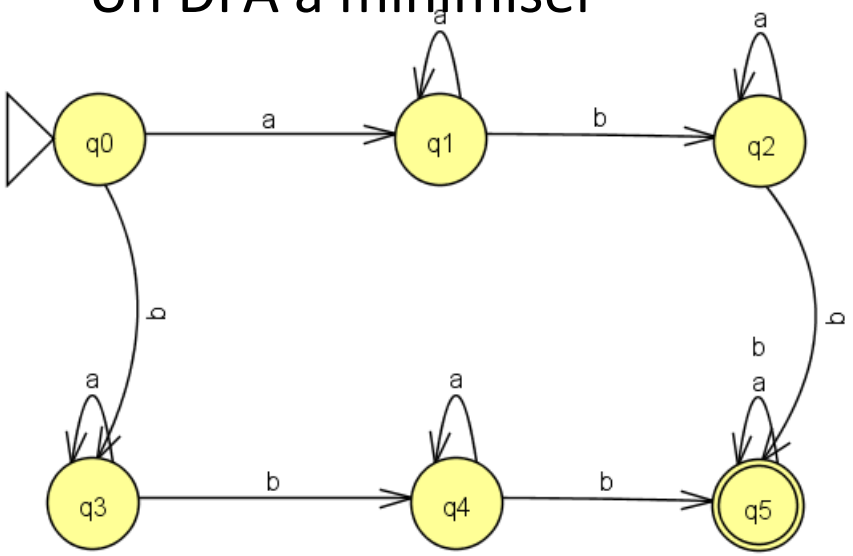
Minimisation (algorithme de Moore)

- **Idée**: identifier les classes d'équivalence à partir d'un automate fini déterministe $A = (Q, q_0, \Sigma, \delta, F)$
  - la partition correspondant aux classes d'équivalence est construite par raffinement de la partition initiale Π_0 qui distingue simplement états finaux et non-finaux
- Algorithme (itératif)

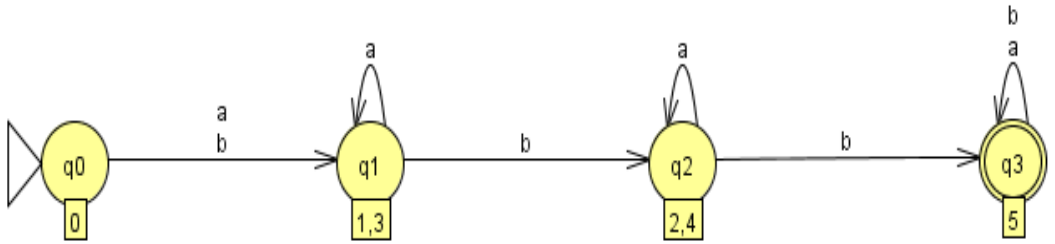
1. Initialiser avec deux classes d'équivalence : F et Q\ F
2. Itérer jusqu'à stabilisation :
   - pour toute paire d'états q et p dans la même classe de la partition Π_k, s'il existe $a \in \Sigma$ tel que $\delta(q, a)$ et $\delta(p, a)$ ne sont pas dans la même classe pour Π_k, alors ils sont dans deux classes différentes pour Π_{k+1}.

Exemple

Un DFA à minimiser



L'automate minimal



— $\Pi_0 = \{\{0, 1, 2, 3, 4\}, \{5\}\}$ (car 5 est le seul état final)

— $\Pi_1 = \{\{0, 1, 3\}, \{2, 4\}, \{5\}\}$ (car 2 et 4, sur le symbole b, atteignent 5).

— $\Pi_2 = \{\{0\}, \{1, 3\}, \{2, 4\}, \{5\}\}$ (car 1 et 3, sur le symbole b, atteignent respectivement 2 et 4).

— $\Pi_3 = \Pi_2$ fin de la procédure.

Corolaire

- L'automate canonique associé à un langage reconnaissable est l'automate déterministe complet ayant le minimum d’états qui reconnaisse ce langage.
- On peut **décider si deux automates finis reconnaissent le même langage** :
  - On les rend déterministes, complets et accessibles en passant aux parties,
  - on quotiente les automates obtenus par leurs équivalence de Nerode,
  - on vérifie que les automates réduits ainsi obtenus sont isomorphes.
- On peut **décider si deux expressions rationnelles** représentent le même langage
  - construire un automate associé à chacun d'entre eux et leur appliquer la procédure précédente.

Résidus d'une expression rationnelle

- On peut directement calculer l'automate canonique du langage associé à une expression rationnelle en définissant une opération de résidu sur les expressions rationnelles telle que : $L(u^{-1}e) = u^{-1}L(e)$.
- Pour cela on part des identités suivantes :
  - $a^{-1}(L_1 \cup L_2) = a^{-1}L_1 \cup a^{-1}L_2$,
  - $a^{-1}(L_1 \cdot L_2) = (a^{-1}L_1)\cdot L_2 \cup \rho(L_1)\cdot a^{-1}L_2$,
  - $a^{-1}L^* = (a^{-1}L)\cdot L^*$
  - où $\rho(L) = \{\varepsilon\}$ si $\varepsilon \in L$ et $\rho(L) = \varnothing$ sinon.

Résidus d'une ER

- la fonction $\boldsymbol{\rho}$ ci-dessus sur les expressions rationnelles est définie inductivement par
  - $\rho(0) = \rho(a) = 0$,
  - $\rho(1) = \rho(e^*) = 1$,
  - $\rho(e_1 + e_2) = \rho(e_1) + \rho(e_2)$ et
  - $\rho(e_1 \cdot e_2) = \rho(e_1) \cdot \rho(e_2)$
- de sorte que $L(\rho(e)) = 1$ si $\varepsilon \in L(e)$ et $L(\rho(e)) = 0$ sinon.
- Le résidu d'une expression rationnelle par une lettre est alors donné par :
  - $a^{-1}1 = 0, \quad a^{-1}a = 1, \quad a^{-1}b = 0 \quad$ si $a \neq b$
  - $a^{-1}(e_1 + e_2) = a^{-1}e_1 + a^{-1}e_2$
  - $a^{-1}(e_1 \cdot e_2) = (a^{-1}e_1) \cdot e_2 + \rho(e_1) \cdot a^{-1}e_2$
  - $a^{-1}(e^*) = (a^{-1}e) \cdot e^*$

Calcul de **Brzozowski**

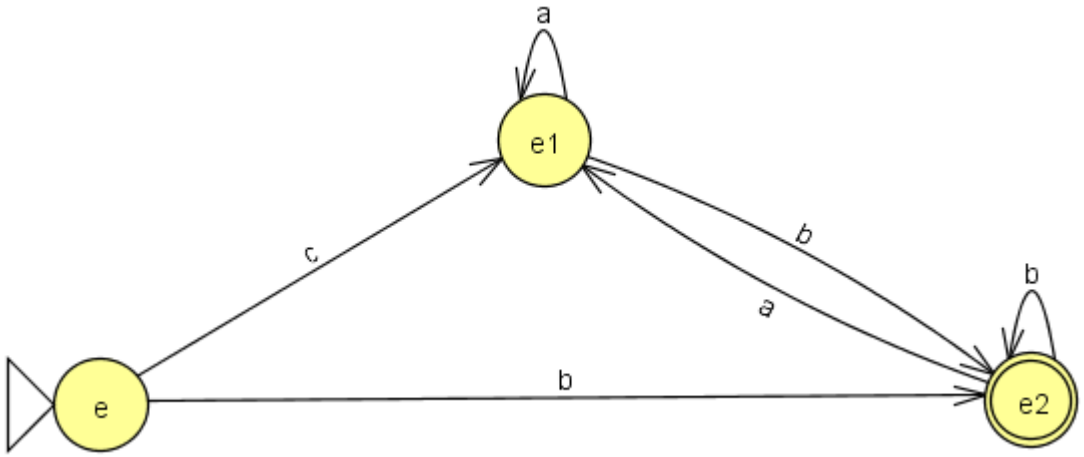
- 1. initialiser l'ensemble des expressions à traiter par l'expression rationnelle dont on cherche à construire l'automate canonique.
- 2. tant qu'il reste des expressions à traiter faire le traitement suivant
  a. Choisir une expression parmi l'ensemble des expressions non traitées et la retirer de cet ensemble.
  b. En utilisant le fait que la somme est associative, commutative, idempotente, et admet 0 comme neutre, que le produit est associatif, admet 1 comme neutre et distribue par rapport à la somme, et la règle d'expansion $\mathbf{e^* = 1+e \cdot e^*}$ écrire e sous l'une des équations suivantes :
     i. $e = a_1 \cdot e_1 + \ldots + a_k \cdot e_k$
     ii. $e = 1 + a_1 \cdot e_1 + \ldots + a_k \cdot e_k$
  c. Parmi les expressions e_i identifier celles qui ont déjà été rencontrées et ajouter les autres à l'ensemble des expressions à traiter.

Construction de l’automate minimal

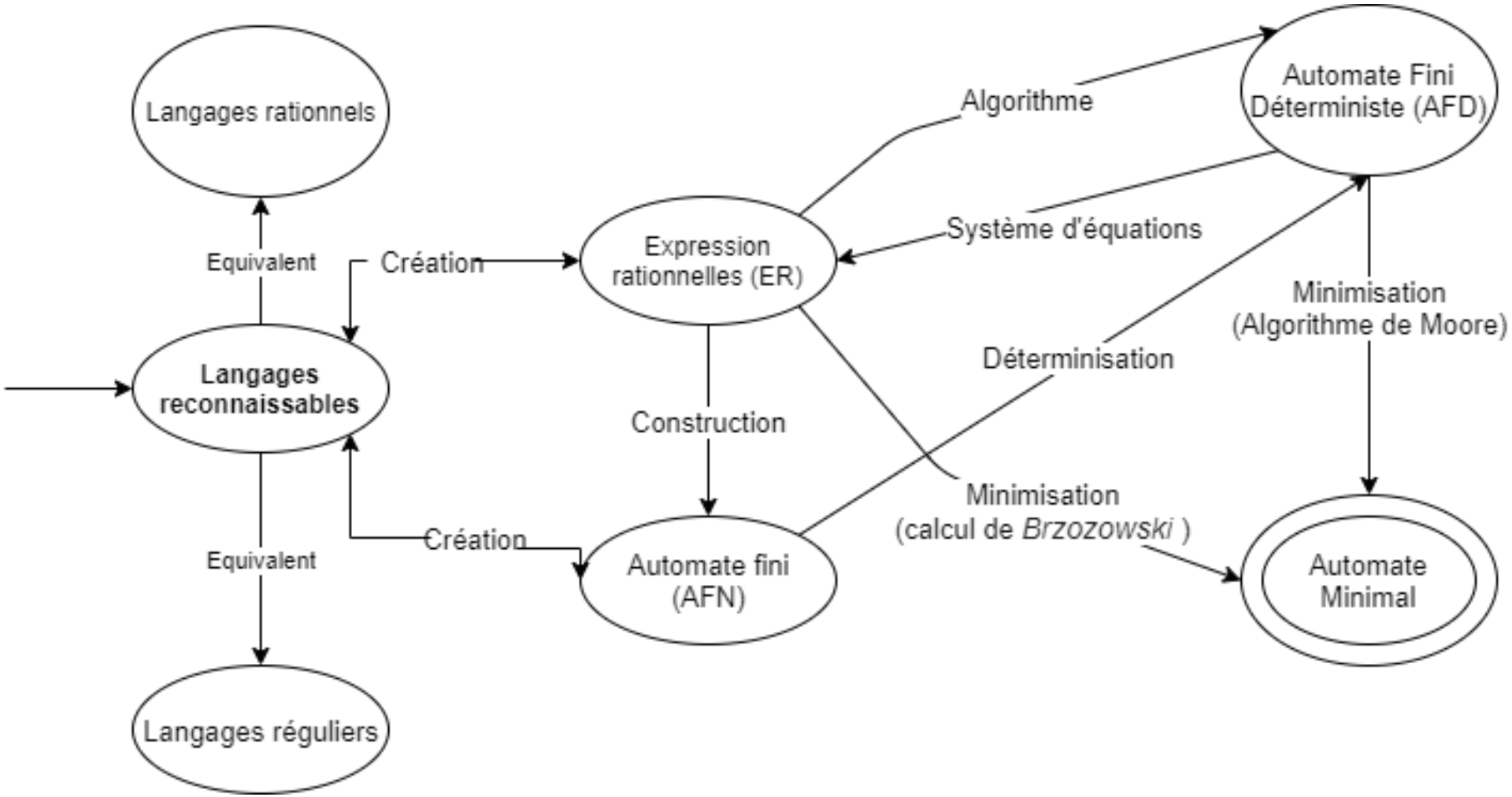
- 3. L'ensemble des équations obtenues nous donne une description de l'automate canonique, plus précisément
  a. Les états de l'automate sont les expressions obtenues, l'état initial est l'expression de départ et une expression correspond à ***un état final si 1 apparait en partie droite de l'équation*** qui la définit.
  b. On a une transition de e_i vers e_j étiquetée a si, et seulement si, le terme $a \cdot e_j$ apparait en partie droite de l'équation définissant e_i.

Exemple

- $e = (ca^*b+b)\cdot(aa^*b+b)^*$
- $e = ca^*b(aa^*b+b)^* + b(aa^*b+b)^*$ distributivité de la concaténation sur +
- $e = c\cdot e_1+b\cdot e_2$ avec $e_1 = a^*b(aa^*b+b)^*$ et $e_2 = (aa^*b+b)^*$
- $e_1 = a^*b(aa^*b+b)^* = (1+aa^*)b(aa^*b+b)^*$ on sait que $e^*=1+ee^*$
- $e_1 = b\cdot e_2+a\cdot e_1$
- $e_2 = (aa^*b+b)^* = 1+(aa^*b+b)(aa^*b+b)^*$
- $e_2 = 1+a\cdot e_1+b\cdot e_2$



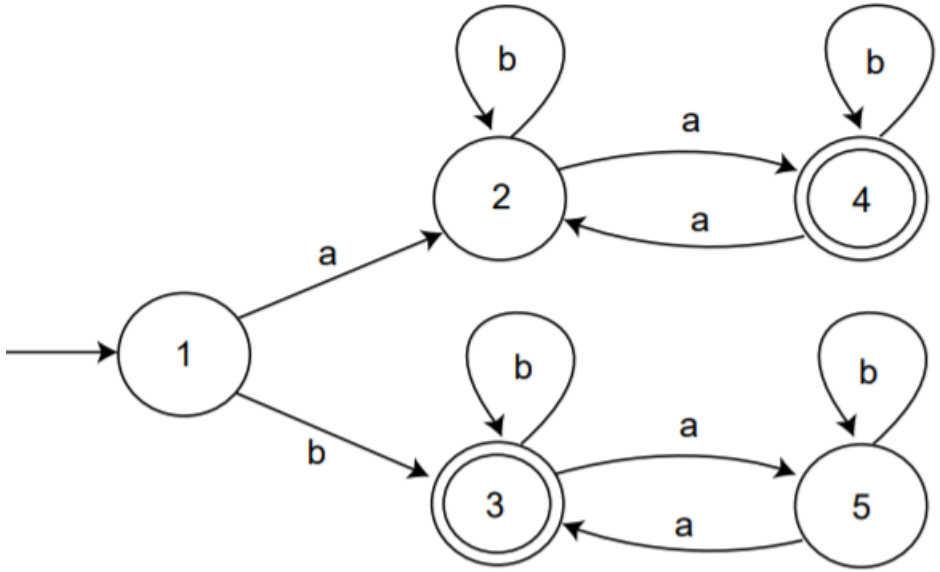
Synthèse (sous forme d'un automate)

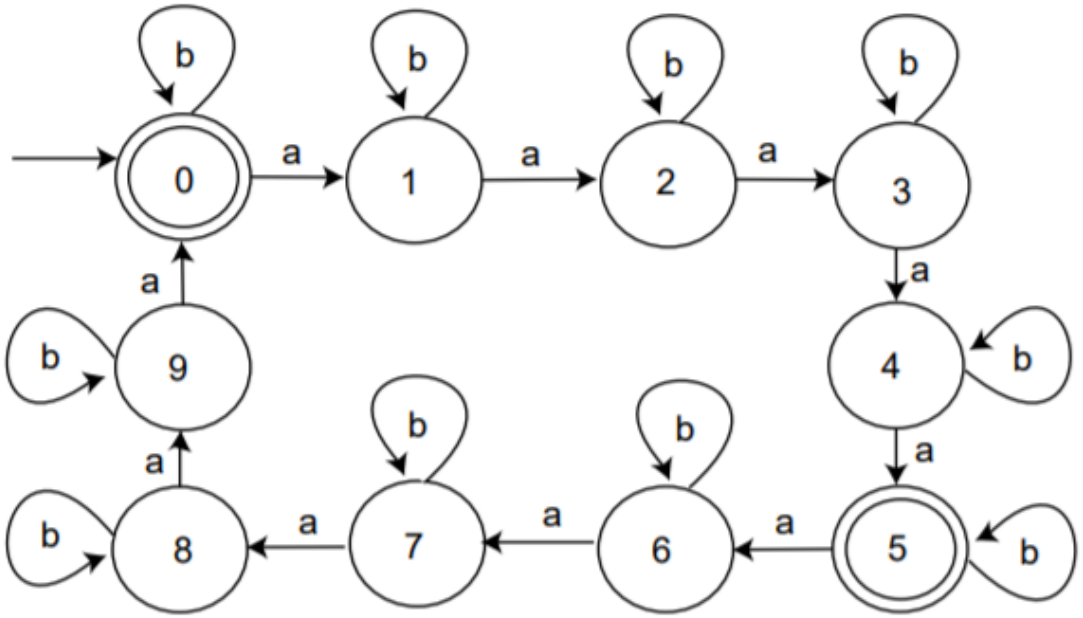


Exercices

- *Montrer que:* $\forall u \in \Sigma^*, a \in \Sigma\ (ua)^{-1}L = a^{-1}u^{-1}L$
- *Généralisation: Montrer que* $\forall u, v \in \Sigma^*, (uv)^{-1}L = v^{-1}u^{-1}L$
- *Caractériser les résidus du langage* $\{a^n b^n / n \in IN\}$ *et en déduire que ce langage n'est pas reconnaissable*
- *Utiliser la méthode des résidus pour construire l'automate canonique associé à chacune des expressions suivantes :*
  1. $(a+b)^*aba(a+b)^*$
  2. $(a+b)^*(ab+ba)(a+b)^*$
  3. $(a+b)^*a(a+b)(a+b)$
- *Montrer que* $(a+bd^*c)^* = a^*+a^*b(ca^*b+d)^*ca^*$.

Minimiser les automates

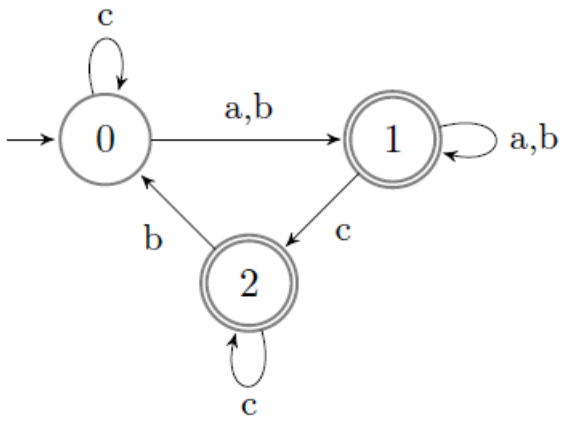




Exercice

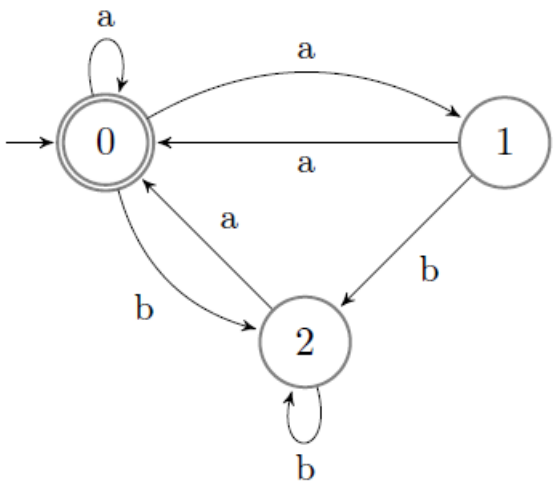
On considère l'automate ci-après

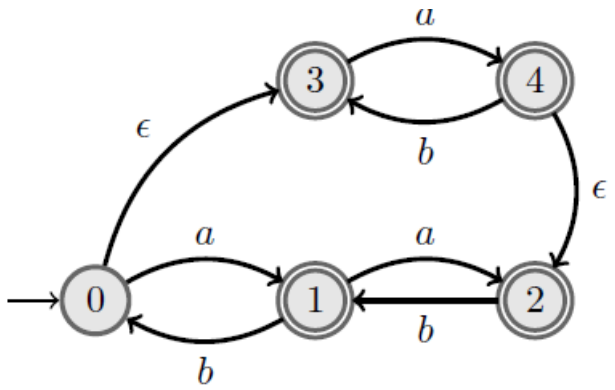
1. Est-ce que cet automate accepte le mot ccabab ? Et le mot acbccccc ?
2. Quel est l'ensemble des états finaux de cet automate ?
3. Cet automate est-il complet ? Est-il déterministe ?



Calculer l'automate minimal correspondant à l'expression $(a(ab)^*)^* + (ab)^*$

Determiniser chacun des automates suivants, puis minimiser l'automate obtenu





Exercices pratiques

- **Exercice 1.** Monsieur Berger **B** emmène un Loup **L**, une chèvre **C**, et un choux **X** près d'une rivière et souhaite traverser avec un petit bateau. Le bateau est tellement petit que B peut entrer dans le bateau avec au plus un passager. Sans surveillance de B, L mange C et C mange X. Comment B peut faire traverser la rivière à la compagnie ?
  - 1. Construire un automate qui modélise la situation.
  - 2. Utiliser cet automate pour trouver comment le problème peut être résolu.
- **Exercice 2.** Si U est un alphabet fini, on appelle liste de U une séquence débutant par le symbole '(', finissant par le symbole ')', et contenant des éléments de U séparés par des ':'. Ainsi par exemple: (1 : 2 : 3 : 2 : 1) est une liste d'éléments de U = {1, 2, 3}. Dans tout l'exercice, on considérera qu'une liste contient toujours au moins un élément: il n'y a pas de liste vide.
  - Est-il possible de reconnaître l'ensemble des listes d'éléments de U = {1, 2, 3} avec un automate fini ? Si votre réponse est positive, donnez une représentation graphique d'un automate reconnaissant ce langage, en indiquant précisément l'alphabet, l'état initial et le ou les états finaux. Les automates les plus simples seront préférés.

Exercices pratiques

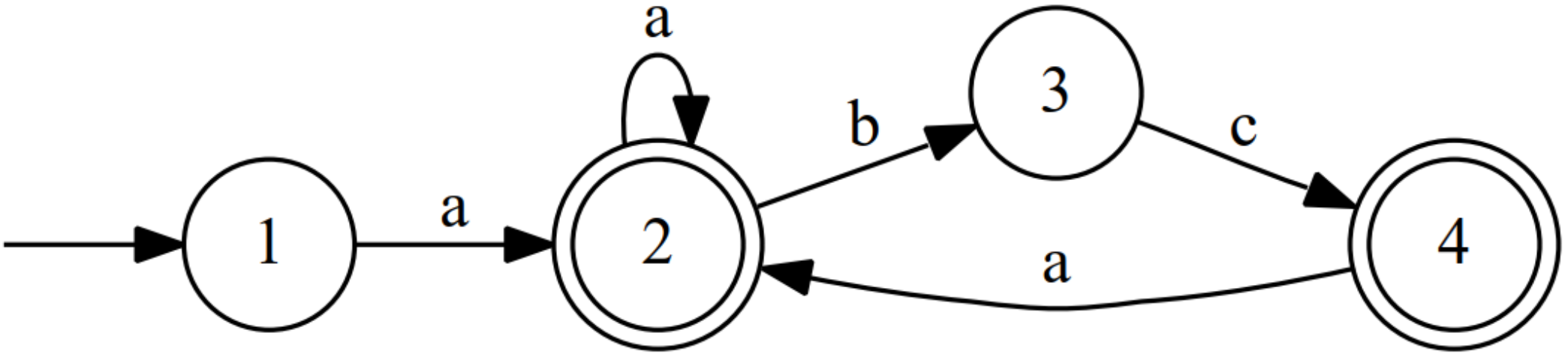
- **Exercice 2** – Distributeur de boissons
- On modélise un distributeur de boissons par un automate fini. L'alphabet d'entrée Σ = {d, v, c} correspond aux pièces de 25, 50 et 100 francs acceptées par le distributeur. Une boisson coûte 100 francs.
  - Construisez un automate A déterministe reconnaissant toutes les séquences de pièces dont le total est 100 francs. Vous pourrez éventuellement commencer par proposer un automate non-déterministe reconnaissant toutes ces séquences.

Travaux Pratiques

- **Problème**:
  - Ecrire un programme qui teste si une chaîne donnée (sous forme d'une chaîne de caractères) appartient au langage d'un automate fini déterministe donné.
- **Solution 1: Automate code dans les instructions du programme**
  - Les transitions sont transcrites par des branchements conditionnels
- **Solution 2: Implémentation d'un automate comme une donnée**
  - La représentation de l'automate est dans des variables. Un programme pour interprète ces données et produit une exécution.
  - Une représentation classique se fait au moyen de tableaux.
    - Un tableau à deux dimensions pour les transitions
    - et un tableau de booléens pour désigner si un état est final ou non

Exemple

- Donner deux versions de l'implémentation de l'automate ci-dessous
- On utilisera le langage Java



Grammaires et langages algébriques

Un langage est dit algébrique s'il peut être engendrée par une grammaire algébrique. On introduit les notions de grammaire algébrique, d'arbre de dérivation, de trace de dérivations et d'ambiguïté.

Grammaires algébriques

- **Définition:** *Une grammaire algébrique $G = (\Sigma, V, R, S)$ consiste en*
  - *un alphabet Σ dont les symboles sont appelés terminaux*
  - *un ensemble V de variables ou non-terminaux*
  - *un ensemble $R \subseteq V \times (V \cup \Sigma)^*$ de règles de productions*
  - *un axiome $S \in V$*
- Note
  - réserver les lettres minuscules du début de l'alphabet latin a,b,c … pour désigner les symboles terminaux (éléments de Σ)
  - les lettres de la fin de l'alphabet u,v,w,…u′,v′,w′…u_1, u′$_1$… pour désigner des mots constitués de symboles terminaux u $\in \Sigma^*$
  - les lettres de l'alphabet grec $\alpha,\beta,\gamma,\ldots\alpha',\beta_1,\gamma'_2\ldots$ pour désigner des suites de symboles grammatiaux i.e des éléments de $(\Sigma \cup V)^*$.

Notations des productions

- Une production $(A,\alpha) \in V\times(V\cup\Sigma)^*$ sera écrite sous la forme $A \rightarrow \alpha$
- On pourra regrouper ensemble les productions $A \rightarrow \alpha_1$, $A \rightarrow \alpha_2 \ldots$ $A \rightarrow \alpha_n$ concernant un même non-terminal en utilisant la notation factorisée
  - $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ ou encore
  - $A \rightarrow \alpha_1 + \alpha_2 + \ldots + \alpha_n$.
  - Lorsque cette dernière notation est utilisée on conviendra de noter le mot vide par 1 plutôt que sous la forme ε
- Exemple: La grammaire $G = (\Sigma,V,R,S)$ donnée par
  - $\Sigma = \{a;b\}$, $V = \{S,A,B\}$, et
  - $R = \{S \rightarrow aAbS;\ S \rightarrow bBaS;\ S \rightarrow \varepsilon;\ A \rightarrow aAbA;\ A \rightarrow \varepsilon;\ B \rightarrow bBaB;\ B \rightarrow \varepsilon\}$ sera représentée par
    - $S \rightarrow aAbS \mid bBaS \mid \varepsilon$ $A \rightarrow aAbA \mid \varepsilon$ B -- $\rightarrow bBaB \mid \varepsilon$ ou encore
    - $S \rightarrow 1 + aAbS + bBaS$ $A \rightarrow 1 + aAbA$ $B \rightarrow 1 + bBaB$