Théorie de la complexité – Feuille de cours intégré n° 1
Automates finis et langages réguliers

Remarque générale pour les exercices de ce cours

Les exercices sont des résultats souvent importants. Retenez les techniques de preuve et les résultats principaux. Vous pourrez les réutiliser en devoir si besoin.

Une question centrale en informatique est de comprendre comment raisonner sur les programmes et les algorithmes. Pour cela, on a besoin de comprendre ce que sont les programmes et les algorithmes, pas seulement de façon intuitive, mais aussi de façon **mathématique**.

Un **modèle de calcul** définit des objets mathématiques qui permettent de représenter des programmes. Il y en a plusieurs sortes, des plus simples (les automates finis) aux plus réalistes (les machines de Turing, qui seront au cœur de ce cours). Entre ces deux extrêmes, il existe des modèles de calcul intermédiaires importants, comme les automates à pile.

Nous commencerons par étudier le modèle de calcul le plus simple : les automates finis, en détaillant d'abord leur syntaxe (c'est-à-dire la façon dont ils sont donnés), puis leur sémantique (c'est-à-dire l'objet mathématique qu'un automate définit). Auparavant, nous introduisons le vocabulaire dont nous aurons besoin tout au long de ce cours.

1 Vocabulaire

Mots.

Tout objet informatique peut se représenter comme une suite d'octets. Pour cette raison, la notion fondamentale dans ce cours est celle de **mot**. On se donne un ensemble fini de **lettres**, qu'on appelle un **alphabet**. Dans le cours, les alphabets seront notés Σ, Γ, ... Par exemple, on pourra choisir $\Sigma = \{a, b, c, \ldots, z\}$, ou $\Sigma = \{a, b, c, d, r\}$, ou bien encore l'alphabet binaire $\Sigma = \{0, 1\}$ suivant l'application souhaitée.

Une fois fixé un alphabet Σ, un **mot** sur Σ est simplement une suite finie de lettres de Σ. On note un mot simplement par la suite finie de ses lettres, sans séparer les lettres par des virgules. Par exemple, 11111100101 est un mot sur l'alphabet $\{0, 1\}$, et *abracadabra* est un mot sur l'alphabet $\{a, b, c, d, r\}$. Le **mot vide**, c'est-à-dire la suite de longueur 0, se note ε. La **longueur** d'un mot u est son nombre de lettres, notée $|u|$. Ainsi, $|\varepsilon| = 0$, et $|abcaa| = 5$. L'ensemble de tous les mots sur l'alphabet Σ se note Σ^*, et on note $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ l'ensemble de tous les mots non vides.

On peut concaténer deux mots. Cela consiste simplement à coller le second après le premier. Par exemple, si $u = abra$ et $v = cadabra$, alors la concaténation de u et v, que l'on note uv ou $u \cdot v$, est :

$$uv = abracadabra.$$

De plus, pour tout nombre entier $n \in \mathbb{N}$, on notera u^n la concaténation de n copies du mot u. Par exemple, $(ab)^3 = ababab$. Par convention, on a $u^0 = \varepsilon$ pour tout mot u.

Langages et problèmes

Un **langage** (sur l'alphabet Σ) est simplement un sous-ensemble de Σ^*. En informatique, un **problème de décision** est simplement un **langage**. Les deux notions sont identiques.

Plus précisément, la notion de langage est la formalisation mathématique de celle de problème de décision. En effet, un problème de décision pose une question sur des objets, à laquelle la réponse est « oui » ou « non ». Ces objets peuvent toujours être codés par des mots. Savoir répondre au problème revient alors à tester si un mot représente un objet pour lequel la réponse à la question est « oui ». L'ensemble de ces mots est un langage (par définition), qui représente donc l'ensemble de toutes les instances « oui » du problème (on parle d'**instances positives**).

Pour résumer, on associe à un problème l'ensemble des mots qui codent les instances positives du problème. Inversement, étant donné un langage $L \subseteq \Sigma^*$, le problème suivant est tel que ses instances positives sont exactement les mots de L :

Donnée Un mot sur l'alphabet Σ.

Question Ce mot appartient-il à L ?

Pour ces raisons, on identifie les deux notions. En particulier, on voit tout problème de décision comme un langage : celui des mots qui codent les instances positives du problème.

Remarque

Bien entendu, avant d'identifier un problème de décision avec un langage qui représente ses instances positives, il faut au préalable avoir fixé un codage des instances (positives et négatives) du problème, sur un certain alphabet. Cela ne pose jamais de difficulté : les codages sont toujours simples, en pratique.

Exercice 1

1. Proposer deux codages des graphes finis non orientés par des mots sur l'alphabet $\{0,1\}$. Y a-t-il des mots qui ne codent aucun graphe ?
2. Même question pour les graphes (finis, non orientés) pondérés sur $\mathbb{Z}$, c'est-à-dire dont les arêtes portent un entier. ▪

2 Automates finis et langages réguliers

Les automates finis sont un modèle de calcul : chaque automate représente un programme (on peut ainsi coder un automate en C, Python, OCaml ou tout autre langage). Par rapport aux programmes réalistes, les possibilités des automates sont cependant très limitées.

2.1 Définition

Chaque automate définit un ensemble de mots, autrement dit un langage, que l'on appelle **langage reconnu** par l'automate. Comme les problèmes de décision sont des langages, on peut, lorsqu'on a un problème de décision, se demander s'il existe un automate fini qui le résout, au sens où il accepte exactement les instances positives du problème.

Syntaxe. Un automate fini est un tuple $(\Sigma, Q, I, F, \delta)$ où :

— Σ est l'alphabet d'entrée, il contient les symboles avec lesquels on écrit l'entrée de l'automate.

— Q est un ensemble fini d'états.

— $I \subseteq Q$ est l'ensemble des états initiaux (ceux dans lesquels le calcul peut commencer).

— $F \subseteq Q$ est l'ensemble des états finaux (ceux dans lesquels le calcul peut s'arrêter).

— δ est la relation de transition. C'est elle qui constitue le « programme » de l'automate. On a :

$$\delta \subseteq Q \times \Sigma \times Q.$$

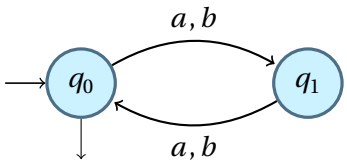
Remarque

Plutôt que d'écrire que $(p,a,q)\in\delta$, on écrit souvent $p \xrightarrow{a} q$.

De la même façon, on représente un automate graphiquement, par un graphe orienté : les sommets sont les états, et on indique les transitions entre états par des arcs étiquetés par la lettre impliquée dans la transition. On indique les états initiaux par une flèche entrante, et les états finaux par une flèche sortante.

Exercice 2

1. Donner la liste des transitions de l'automate suivant.



2. Dessiner l'automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ sur l'alphabet $\Sigma = \{a, b\}$, avec $Q = \{q_0, q_1, q_2, q_3\}$, $I = \{q_0\}$, $F = \{q_3\}$ et $\delta = \{(q_0, a, q_1), (q_1, b, q_2), (q_2, a, q_2), (q_2, b, q_2), (q_2, b, q_3)\}$. ▪

Sémantique. Considérons un automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$. Pour tout mot $w \in \Sigma^*$, on va définir l'exécution de $\mathscr{A}$ sur le mot d'entrée w (le mot *sémantique* fait référence à tout ce qui concerne l'exécution d'une machine, par opposition à sa *syntaxe* qui fait référence au texte du programme).

Soit $w = a_1 \cdots a_n \in \Sigma^*$ avec $a_1, \ldots, a_n \in \Sigma$. Le mot w est donc de longueur n. Une **exécution partielle** (ou un **calcul partiel**) sur w dans l'automate $\mathscr{A}$ est une suite de $n+1$ états $q_0, \ldots, q_n \in Q$ telle que pour tout $i \geqslant 1$, on a $(q_{i-1}, a_i, q_i) \in \delta$. Avec la notation précédente, on représente une telle exécution partielle de la façon suivante :

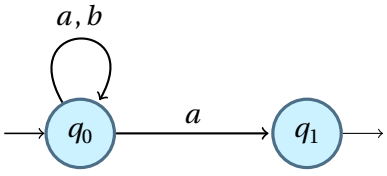
$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} q_n.$$

On dit que $w = a_1 \cdots a_n$ est **l'étiquette** de l'exécution partielle.

On dit que q_0 est le *premier état* de cette exécution partielle et que q_n est son *dernier état*. Si $q, r \in Q$ sont des états et v est un mot de A^*, on note $q \xrightarrow{v} r$ s'il existe une exécution partielle sur v dans $\mathscr{A}$ allant de q à r, c'est-à-dire dont le premier état est q et le dernier état est r.

Une **exécution** (ou un **calcul**) est une exécution partielle dont le premier état est initial, c'est-à-dire dans I. Une exécution (ou un calcul) est **acceptante** si son dernier état est final, c'est-à-dire dans F.

Exercice 3 On considère l'automate suivant.



1. Donner un calcul sur le mot $abba$.
2. Le calcul donné à la question précédente est-il unique ?
3. Décrire en français l'ensemble de tous les mots qui étiquettent un calcul acceptant dans cet automate. ▪

On dit qu'un mot est **accepté**, ou **reconnu** par l'automate $\mathscr{A}$ s'il existe un calcul acceptant sur ce mot dans $\mathscr{A}$. Notez qu'il peut aussi y avoir d'autres calculs qui ne sont pas acceptants. Le **langage accepté**, ou **reconnu** par l'automate $\mathscr{A}$ est l'ensemble des mots acceptés par $\mathscr{A}$. Notez qu'un même langage peut être reconnu par plusieurs automates distincts. Enfin, on dira qu'un langage est **régulier** si il existe un automate qui le reconnait.

Exercice 4 Pour chacun des automates des exercices précédents, dire quel est le langage reconnu. ▪

Exercice 5 Pour chacun des langages suivants, construire un automate qui le reconnaît.

1. Le langage des mots sur l'alphabet $\{a,b,c\}$ de longueur impaire.
2. Le langage des mots sur l'alphabet $\{a,b,c\}$ qui ont un nombre pair de c.
3. Le langage des mots sur l'alphabet $\{a,b,c\}$ qui se terminent par $abba$.
4. Le langage des mots sur l'alphabet $\{a,b,c\}$ qui contiennent $abba$.
5. Le langage des mots sur l'alphabet $\{0,1\}$ qui représentent un entier divisible par 3. ▪

Exercice 6 Montrer que les langages suivants sur l'alphabet $\Sigma=\{a,b\}$ ne sont pas réguliers :

1. le langage $\{a^n b^n \mid n\in\mathbb{N}\}$.
2. le langage $\{ww \mid w\in\Sigma^*\}$. ▪

Exercice 7 — ♢,♠. Soit $\Sigma=\{a,b,c\}$. Pour tout mot $w\in\Sigma^*$ et toute lettre $\ell\in\Sigma$, on note $\#_\ell(w)\in\mathbb{N}$, le nombre de copies de la lettre "ℓ" dans w. Le langage suivant est-il régulier ?

$$L=\Big\{uv\in\Sigma^* \mid \#_a(u)+\#_b(u)=\#_b(v)+\#_c(v)\Big\}.$$ ▪

Exercice 8 — ♢,♠. Soit $\Sigma=\{a,b\}$. Pour tout mot $w\in\Sigma^*$ et tout mot $u\in\Sigma^+$, on note $\#_u(w)\in\mathbb{N}$, le nombre de copies du mot u dans w, c'est-à-dire le nombre de décompositions de w de la forme $w=w_1uw_2$ (où w_1 et w_2 sont deux mots éventuellement vides). Les deux langages suivants sont-ils réguliers ?

$$L_1=\Big\{w\in\Sigma^* \mid \#_{ab}(w)=\#_{ba}(w)+1\Big\} \quad\text{et}\quad L_2=\Big\{w\in\Sigma^* \mid \#_{aba}(w)=\#_{bab}(w)\Big\}.$$ ▪

2.2 Déterminisme

On considère un automate $\mathscr{A}=(\Sigma,Q,I,F,\delta)$. On dit que $\mathscr{A}$ est **complet** si et seulement si la propriété suivante est satisfaite :

$$|I|\geqslant 1 \quad\text{et}\quad \text{pour tout } (p,a)\in Q\times\Sigma, \text{ on a } \Big|\Big\{q\in Q \mid (p,a,q)\in\delta\Big\}\Big|\geqslant 1.$$

De plus, on dit que $\mathscr{A}$ est **déterministe** si et seulement si la propriété suivante est satisfaite :

$$|I|\leqslant 1 \quad\text{et}\quad \text{pour tout } (p,a)\in Q\times\Sigma, \text{ on a } \Big|\Big\{q\in Q \mid (p,a,q)\in\delta\Big\}\Big|\leqslant 1.$$

Exercice 9

1. Exprimer en français la propriété d'être complet pour un automate.
2. Même question pour la propriété d'être déterministe.

▪

Exercice 10 Soit $\mathscr{A}=(\Sigma,Q,I,F,\delta)$ un automate complet *et* déterministe. Montrer que pour tout mot $w\in A^*$, il existe une *unique* exécution de l'automate $\mathscr{A}$ sur w. ▪

Exercice 11 On considère un alphabet Σ et un mot $w\in\Sigma^*$.

1. Montrer que le langage Σ^*w est reconnu par un automate déterministe contenant $|w|+1$ états.
2. Montrer que tout automate déterministe qui reconnaît Σ^*w contient au moins $|w|+1$ états.
3. Est-il vrai que tout automate (pas nécessairement déterministe) qui reconnaît Σ^*w contient au moins $|w|+1$ états ? ▪

Exercice 12 On dit que deux automates sont **équivalents** s'ils reconnaissent le même langage.

1. Démontrer que pour tout automate, il existe un automate équivalent qui est complet.
2. Démontrer que pour tout automate, il existe un automate équivalent qui est déterministe et complet. ▪

Exercice 13 — ♢. 1. Dans votre construction de l'automate déterministe reconnaissant le même langage qu'un automate quelconque, quel est le nombre d'états, au maximum, que peut avoir l'automate déterministe (en fonction du nombre d'états de l'automate d'origine) ?

Soit n un entier naturel. On considère le langage L_n des mots sur l'alphabet $\{a,b\}$ dont la n-ième lettre avant la fin du mot est un a. Par exemple, L_1 est l'ensemble des mots qui se terminent par a, et L_2 est l'ensemble des mots dont l'avant-dernière lettre est un a.

2. Donner un automate à $n+1$ états qui reconnaît L_n.
3. Démontrer que tout automate déterministe qui reconnaît L_n a au moins 2^n états. ▪

2.3 Clôtures

On montre que les langages réguliers sont préservés par les opérations naturelles. On commence par les opérations Booléennes (union, intersection, complément).

Exercice 14 Soit Σ un alphabet et $K, L \subseteq \Sigma^*$ deux langages réguliers.

1. Montrer que $K \cup L$ est régulier.
2. Montrer que $\Sigma^* \setminus L$ est régulier.
3. Que peut-on en conclure sur $K \cap L$? ▪

On considère maintenant deux opérations spécifiques aux langages. On peut tout d'abord étendre la concaténation aux langages. Soit Σ un alphabet et $K, L \subseteq \Sigma^*$ deux langages. La concaténation de K et L, notée KL, est définie comme le langage suivant :

$$KL = \{uv \mid u \in K \text{ et } v \in L\}.$$

On généralise également la puissance à un langage : on note L^n la concaténation de n copies de L, c'est-à-dire $L^n = \underbrace{LL\cdots L}_{n \text{ fois}}$. Par convention, $L^0 = \{\varepsilon\}$ pour tout langage L.

De plus on considère un second opérateur : l'*étoile de Kleene*. Soit Σ un alphabet et $L \subseteq \Sigma^*$ un langage. On définit L^* comme le langage suivant :

$$L^* = \bigcup_{n \in \mathbb{N}} L^n.$$

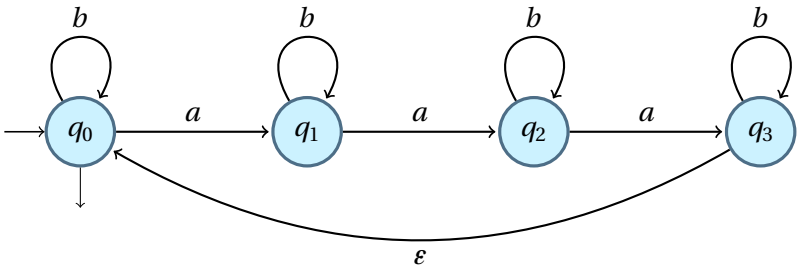
Enfin, on notera $L^+ = LL^*$.

Exercice 15 Montrer que pour tous langages K, L, on a $(K \cup L)^* = (K^*L^*)^*$. ▪

On cherche maintenant à prouver que les langages réguliers sont préservés par ces deux opérateurs. On va commencer par un résultat préliminaire.

Exercice 16 — ε-automates. Les ε-automates généralisent les automates classiques. Il peuvent contenir un nouveau type de transitions qui sont étiquetées par le mot vide "ε". On parle de "ε-transition". Formellement, dans un ε-automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$, l'ensemble de transition δ est un sous-ensemble de $Q \times (\Sigma \cup \{\varepsilon\}) \times Q$. La sémantique est étendue aux ε-automates de façon naturelle : une ε-transition s'utilise sans consommer une lettre du mot pris en entrée.

1. Quel est le langage reconnu par le ε-automate suivant :



2. Montrer que pour tout ε-automate, il existe un automate classique équivalent. ▪

Exercice 17 Soit Σ un alphabet et $K, L \subseteq \Sigma^*$ deux langages réguliers.

1. Montrer que KL est régulier.
2. Montrer que L^* est régulier.
3. Que peut-on en conclure sur L^+ ? ▪

Exercice 18 — ♢,Conjugués. On considère un alphabet Σ.On dit que deux mots $w_1, w_2 \in \Sigma^*$ sont *conjugués* si il existe $u, v \in \Sigma^*$ tels que $w_1 = uv$ et $w_2 = vu$. Par exemple, aab et baa sont conjugués. Pour tout mot $w \in \Sigma^*$, on notera $C(w) \subseteq \Sigma^*$ le langage de tous les mots qui sont un conjugué de w. De la même façon, pour tout langage $L \subseteq \Sigma^*$, on note,

$$C(L) \stackrel{\text{def}}{=} \bigcup_{w \in L} C(w) \qquad \text{(qui est un sous-ensemble de } \Sigma^*\text{)}.$$

Montrer que si L est régulier, alors $C(L)$ l'est aussi. ▪

Exercice 19 — ♢. Étant donnés deux alphabets A et B, un morphisme de A^* vers B^* est une application $\alpha : A^* \to B^*$ telle que $\alpha(\varepsilon) = \varepsilon$ et $\alpha(uv) = \alpha(u)\alpha(v)$ pour tous mots $u, v \in A^*$. Montrer que pour tout morphisme $\alpha : A^* \to B^*$ et tout langage régulier $L \subseteq A^*$, le langage $\alpha(L) \subseteq B^*$ est aussi régulier. ▪

Exercice 20 — ♢,♠. On considère un alphabet Σ et un langage régulier $L \subseteq \Sigma^*$. On définit le langage $L_{\frac{1}{2}}$ suivant :

$$L_{\frac{1}{2}} = \left\{ u \in \Sigma^* \mid \text{il existe } v \in \Sigma^* \text{ tel que } |u| = |v| \text{ et } uv \in L \right\}.$$

Montrer que $L_{\frac{1}{2}}$ est régulier. Plus généralement pour tout $i, j \in \mathbb{N} \setminus \{0\}$, on définit le langage $L_{\frac{i}{j}}$ suivant :

$$L_{\frac{i}{j}} = \left\{ u \in \Sigma^* \mid \text{il existe } v \in \Sigma^* \text{ tel que } j \times |u| = i \times |uv| \text{ et } uv \in L \right\}.$$

Montrer que $L_{\frac{i}{j}}$ est régulier. ▪

Exercice 21 — ♢,♠. On considère un alphabet Σ et un langage régulier $L \subseteq \Sigma^*$. On définit le langage $\sqrt{L}$ suivant :

$$\sqrt{L} = \{ w \in \Sigma^* \mid ww \in L \}.$$

Montrer que $\sqrt{L}$ est régulier. ▪

Exercice 22 — ♢,♠,♠,♠. On considère un alphabet Σ et un langage régulier $L \subseteq \Sigma^*$. Pour tout mot $w \in \Sigma^*$, on note $|w| \in \mathbb{N}$ sa longueur (le nombre de lettres qu'il contient). Montrer que le langage suivant est également régulier :

$$\{ w \in \Sigma^* \mid w^{|w|} \in L \}.$$ ▪

2.4 Problèmes de décision

On va maintenant montrer qu'il existe des algorithmes qui résolvent les problèmes de décision naturels qui ont pour entrée un ou plusieurs automates.

Exercice 23 — Problème de l'appartenance. Donner un algorithme qui prend en entrée un automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ ainsi qu'un un mot d'entrée $w \in \Sigma^*$ et teste si w est accepté par $\mathscr{A}$. ▪

Exercice 24 — Problème du vide. Donner un algorithme qui prend en entrée un automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ et teste si le langage reconnu par $\mathscr{A}$ est *vide*. ▪

Exercice 25 — Problème de l'universalité. Donner un algorithme qui prend en entrée un automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ et teste si le langage reconnu par $\mathscr{A}$ est le *langage universel* Σ^*. ▪

Exercice 26 — Égalité sémantique. Donner un algorithme qui prend en entrée deux automates $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ et $\mathscr{B} = (\Sigma, R, J, G, \gamma)$ sur le même alphabet Σ, teste si $\mathscr{A}$ et $\mathscr{B}$ reconnaissent le *même* langage. ▪

3 Expressions régulières

On va maintenant introduire une syntaxe alternative qui permet de définir les langages réguliers : les *expressions régulières*. Elle fonctionnent de façon *descriptive* : elle décrivent les mots contenus dans le langage qu'elles représentent.

On fixe un alphabet Σ quelconque pour la définition. Les expressions régulières sont définies récursivement : elles se construisent à partir d'expressions de base en utilisant des règles. Une expression de base peut avoir les deux formes distinctes suivantes :

— "$\emptyset$", qui définit le langage vide.

— "a" pour tout $a \in \Sigma$, qui définit le langage singleton $\{a\}$.

L'ensemble de toutes les expressions régulières est maintenant construit à partir de celles de base en utilisant les trois règles suivantes :

— si exp_1 et exp_2 sont des expressions régulières, alors $(exp_1 + exp_2)$ en est une aussi. Elle définit le langage $L_1 \cup L_2$, où L_1 et L_2 sont les langages définis par exp_1 et exp_2 respectivement.

— si exp_1 et exp_2 sont des expressions régulières, alors $(exp_1 \cdot exp_2)$ en est une aussi. Elle définit le langage $L_1 L_2$, où L_1 et L_2 sont les langages définis par exp_1 et exp_2 respectivement.

— si exp est une expression régulière, alors $(exp)^*$ en est une aussi. Elle définit le langage L^*, où L est le langage défini par exp.

Quand il n'y a pas d'ambiguïté, on simplifiera la lecture en omettant les parenthèses et les symboles "$\cdot$" dans nos expressions régulières. Par convention, l'étoile de Kleene est prioritaire sur la concaténation. Par exemple, on écrira $a\,b^*$ pour l'expression $(a(b^*))$. De la même façon, la concaténation est prioritaire sur l'union. Par exemple, on écrira $a + b \cdot c$ pour l'expression $(a + (b \cdot c))$.

Exercice 27 On considère l'alphabet $\Sigma = \{a, b, c\}$. Décrire en français les langages définis par les expressions régulières suivantes :

— $(\emptyset)^*$.

— $(a + b + c)^* a (a + b + c)^*$.

— $(a\,b + b + c)^*$.

— $(a\,a)^*$. ▪

Exercice 28 — ♢. On considère un alphabet Σ. Une expression régulière sur l'alphabet Σ peut être vue comme un mot sur l'alphabet $\Sigma \cup \{\emptyset, (,), +, \cdot, *\}$. Montrer que le langage de toutes les expressions régulières n'est pas un langage régulier. ∎

Remarque

Un abus de langage courant quand on utilise les expressions régulières est d'identifier une expression avec le langage qu'elle définit. Par exemple, on parle du "langage $(a\,b)^*$" alors qu'on devrait parler du "langage défini par $(a\,b)^*$". Bien que nous utiliserons cet abus nous-mêmes, il est important de garder en tête que les expressions et les langages sont des objets de nature différente : une expression est un élément de syntaxe et un langage est un objet mathématique qui peut être la sémantique d'une expression.

On cherche maintenant à montrer que l'ensemble de tous les langages définis par une expression régulière est exactement l'ensemble des langages réguliers. On prouve d'abord que tout langage défini par une expression régulière est nécessairement régulier. Commençons par un exemple.

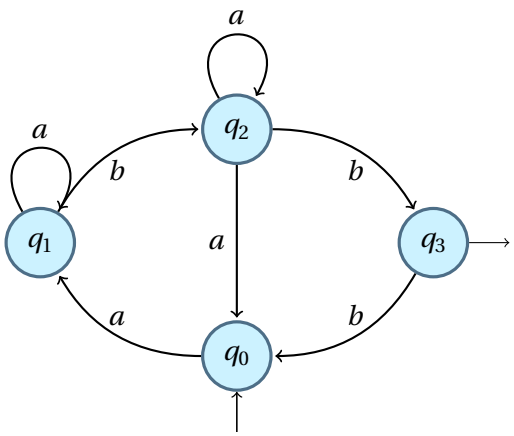
Exercice 29 On considère l'alphabet $\Sigma = \{a, b\}$. Montrer que le langage $(a(a\,b)^*b)^*$ est régulier. ∎

Exercice 30 — Des expressions régulières aux automates. Soit Σ un alphabet et *exp* une expression régulière sur l'alphabet Σ. Montrer que le langage défini par *exp* est régulier. ∎

On va maintenant montrer que tout langage régulier est défini par une expression régulière. On commence par un résultat préliminaire.

Exercice 31 — Lemme d'Arden. Soit Σ un alphabet et $K, L \subseteq \Sigma^*$ deux langages tels que $\varepsilon \notin K$. Montrer que le langage $X = LK^*$ est l'*unique* solution de l'équation $X = XK + L$. ∎

Exercice 32 On considère l'automate $\mathscr{A}$ suivant :



En utilisant l'exercice précédent, construire une expression régulière qui définit le langage reconnu par $\mathscr{A}$. ∎

Exercice 33 — Des automates aux expressions régulières. Montrer que tout langage régulier peut être défini par une expression régulière. ∎

Exercice 34 — ♢,♠. On considère un alphabet Σ. On note $\mathscr{C}$ la plus petite classe de langages sur l'alphabet Σ qui contient tous les langages finis et est close par union, complément et concaténation. Est-il vrai que $\mathscr{C}$ contient tous les langages réguliers ? ∎