Automates finis (non-)déterministes comme graphes orientés étiquetés

Feuille « 05 » (du 8 octobre 2024)

Pour *formaliser* plus simplement les calculs justifiés à la feuille « 04 », comme l'intersection ou la déterminisation de systèmes d'automate, nous généralisons ici la représentation des automates déterministes (cf. feuille « 04 ») à des systèmes d'automate quelconques. La représentation ainsi obtenue correspond aussi à interpréter les automates comme des **graphes orientés étiquetés**. On peut alors appliquer des méthodes et des algorithmes issus de la théorie des graphes (cf. les cours de RO), ce qui permet notamment un *algorithme de déterminisation* simple et efficace sur les automates « presque » déterministes. Cette représentation aide en outre à s'attaquer à de nouveaux problèmes, comme celui de l'équivalence d'automate.

1 Définitions et exemples

Définition générale On appelle **automate fini** un 5-uplet A de la forme $\langle Q, V, s, \delta, F\rangle$ où Q est un ensemble **fini** « d'états », V est le vocabulaire (fini), s l'état initial, $\delta \in Q\times(V\cup\{\varepsilon\}) \to \mathcal{P}(Q)$ est la *relation de transition*, et $F \subseteq Q$ est l'ensemble des *états finaux*. Étant donné un tel automate A, on note Δ_A le système d'équation d'automate sur le vocabulaire V, ayant Q comme ensemble de variables, s pour axiome, et telle que l'inéquation de chaque variable $q \in Q$ soit donnée par :

$$q \supseteq \left(\begin{array}{l} \varepsilon \text{ si } q \in F \\ \varnothing \text{ sinon} \end{array}\right) + \Sigma_{a\in V\cup\{\varepsilon\}}\Sigma_{q'\in\delta(q,a)}\ a.q'$$

Le langage « reconnu » (ou « engendré ») par l'automate est noté $\mathcal{L}(A) \stackrel{def}{=} \mathcal{L}(\Delta_A)$.
Plus généralement, pour $q \in Q$, on note $\mathcal{L}_A(q)$, le langage associé à la variable q dans la plus petite solution de Δ_A, qu'on peut aussi noter $\mathcal{L}_{\Delta_A}(q)$. Donc par définition, $\mathcal{L}(A) = \mathcal{L}_A(s)$.

Sous-catégorie d'automates On distingue les catégories d'automates (finis) listées ci-dessous avec leur propriété caractéristique :

Automate sans ε-transition $\forall\, q \in Q,\ \delta(q,\varepsilon) = \varnothing$.
Dans ce cas, on pourra se limiter à considérer simplement que $\delta \in Q\times V \to \mathcal{P}(Q)$.

Automate déterministe $\forall\, q \in Q,\ \delta(q,\varepsilon) = \varnothing$ et $\forall\, a \in V, \mathrm{card}(\delta(q,a)) \leq 1$.

Automate complet $\forall\, q \in Q\, \forall\, a \in V, \mathrm{card}(\delta(q,a)) \geq 1$.

Pour un automate déterministe et complet, on pourra « abusivement » considérer que $\delta \in Q\times V \to Q$ (comme à la feuille « 04 »).

Définitions équivalentes dans la littérature Voilà des définitions alternatives mais équivalentes dans la littérature :

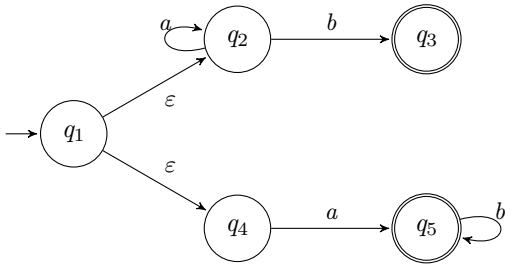
- δ est considéré comme un ensemble de triplets, avec $\delta \in \mathcal{P}(Q\times(V\cup\{\varepsilon\})\times Q)$. Mais comme $\mathcal{P}(Q\times(V\cup\{\varepsilon\})\times Q) \simeq Q\times(V\cup\{\varepsilon\}) \to \mathcal{P}(Q)$ (cf. compléments de la feuille « 02 »), cette représentation alternative change juste un peu les manipulations formelles, sans rien changer sur le fond ;
- au lieu de se limiter à un seul état initial s, on autorise un ensemble d'états initiaux I. Ça nécessite de changer la définition de Δ_A : celui-ci contient en plus une variable supplémentaire s pour l'axiome dont l'inéquation est $s \supseteq \Sigma_{q\in I}\, q$. Cette généralisation n'apporte pas grand chose en pratique et complexifie le reste des définitions. Par exemple, on doit considérer qu'un automate sans ε-transition (ou déterministe) doit avoir un seul état initial.

Représentation graphique Tout automate fini A défini comme ci-dessus correspond à un *graphe orienté étiqueté*[1] dont l'ensemble nœuds est Q ; chaque arc étiqueté correspond à une transition $q' \in \delta(q, a)$ ayant pour départ/source/origine $q \in Q$ et pour arrivée/destination/cible $q' \in Q$, l'étiquette de l'arc étant $a \in V \cup \{\varepsilon\}$. Sur la représentation graphique :

- chaque état est représenté par un « rond » ; une flèche pointe sur l'état initial ; les états finaux sont dans deux « ronds concentriques » ;
- les noms des états sont éventuellement représentés au centre du rond, mais ils ne sont pas obligatoires, puisque les noms des états n'ont aucune influence sur le langage représenté (l'intérêt de les mettre, c'est juste de rendre explicite le lien avec le système d'inéquations— pouvoir les ignorer participe sans doute à rendre la représentation graphique « intuitive ») ;
- chaque transition est représentée par une flèche, partant de l'état de départ à l'état d'arrivée, et surmontée du symbole de $V \cup \{\varepsilon\}$.

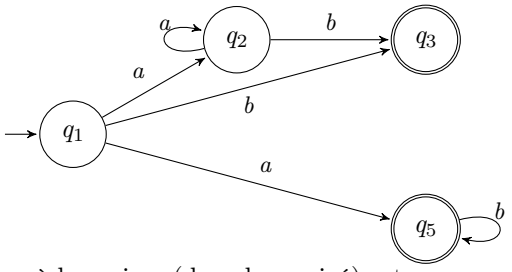
Exemples 1. Dessinons l'automate Δ_1 vu à l'exo 17 de la feuille « 04 » (du langage $a^*b + ab^*$)

$$\Delta_1 \stackrel{def}{=} \begin{cases} q_1 \supseteq q_2 + q_4 \\ q_2 \supseteq a.q_2 + b.q_3 \\ q_3 \supseteq \varepsilon \\ q_4 \supseteq a.q_5 \\ q_5 \supseteq \varepsilon + b.q_5 \end{cases}$$



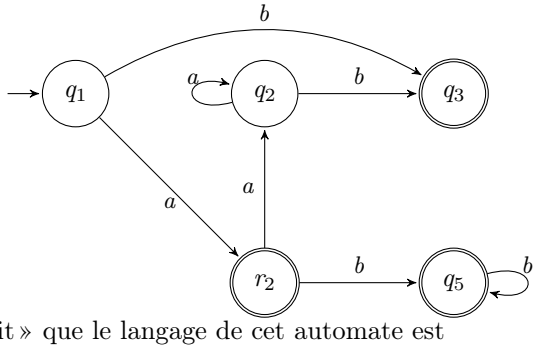
Après élimination de ses ε-transitions, on arrivait à l'automate non-déterministe et sans ε-transition :

$$\begin{cases} q_1 = a.q_2 + a.q_5 + b.q_3 \\ q_2 = a.q_2 + b.q_3 \\ q_3 = \varepsilon \\ q_5 = \varepsilon + b.q_5 \end{cases}$$



L'automate déterministe incomplet obtenu « à la main » (dans le corrigé) est :

$$\begin{cases} q_1 = a.r_2 + b.q_3 \\ r_2 = \varepsilon + a.q_2 + b.q_5 \\ q_2 = a.q_2 + b.q_3 \\ q_3 = \varepsilon \\ q_5 = \varepsilon + b.q_5 \end{cases}$$



Sur la représentation graphique, on « voit » que le langage de cet automate est

$$a.(\varepsilon + b.b^* + a.a^*.b) + b = ab^* + (aa^+)^?b.$$

Ce qui est bien le langage de l'automate de départ.

1. Voir https://fr.wikipedia.org/wiki/Graphe_orient%C3%A9 ou https://en.wikipedia.org/wiki/Directed_graph

Exercice 1. On considère l'automate Δ_4 de l'exo 17 de la feuille « 04 » :

$$\Delta_4 \stackrel{def}{=} \begin{cases} q_1 \supseteq q_2 + a.q_3 + q_5 \\ q_2 \supseteq a.q_1 + q_3 \\ q_3 \supseteq q_1 + b.q_2 + a.q_4 \\ q_4 \supseteq \varepsilon + c.q_4 \\ q_5 \supseteq a.q_5 + b.q_5 \end{cases}$$

Dessiner cet automate, puis celui obtenu après élimination des ε-transitions, puis celui obtenu après « fusion » des variables de même membre droit dans le système d'équations et élimination de l'état puits. Enfin, dessiner l'automate (incomplet) obtenu par déterminisation.

Exercice 2. L'ensemble des littéraux numériques en PYTHON forme un langage, formellement défini dans https://docs.python.org/3/reference/lexical_analysis.html#numeric-literals. Dans cet exercice, on considère un sous-ensemble des littéraux entiers et flottants écrits en base 10 (PYTHON supporte d'autres notations, comme la notation octale, qui est à l'origine des restrictions sur les zéros en tête, mais nous les ignorons pour simplifier). Ils sont composés d'une partie entière, d'une partie décimale optionnelle et d'un exposant optionnel ; ils sont définis sur le vocabulaire

$$V \stackrel{def}{=} \{\mathbf{0}, \ldots, \mathbf{9}, \mathbf{e}, \mathbf{E}, ., +, -\}.$$

On définit les huit ensembles suivants (suite sur la page suivante) :

$$\begin{aligned} \texttt{nonzerodigit} &\stackrel{def}{=} \{\mathbf{1}, \ldots, \mathbf{9}\} \\ \texttt{digit} &\stackrel{def}{=} \{\mathbf{0}\} \cup \texttt{nonzerodigit} \\ \texttt{integer} &\stackrel{def}{=} \texttt{nonzerodigit}\,(\texttt{digit}^*) \cup \{\mathbf{0}\}^+ \\ \texttt{dot} &\stackrel{def}{=} \{.\} \\ \texttt{pointfloat} &\stackrel{def}{=} (\texttt{digit}^*)\,\texttt{dot}\,(\texttt{digit}^+) \cup (\texttt{digit}^+)\,\texttt{dot} \end{aligned}$$

$$\begin{aligned} \texttt{exponent} &\stackrel{def}{=} \{\mathbf{e}, \mathbf{E}\}\,\{\varepsilon, +, -\}\,\texttt{digit}^+ \\ \texttt{exponentfloat} &\stackrel{def}{=} \left(\texttt{digit}^+ \cup \texttt{pointfloat}\right)\texttt{exponent} \\ \texttt{number} &\stackrel{def}{=} \texttt{integer} \cup \texttt{pointfloat} \cup \texttt{exponentfloat} \end{aligned}$$

1. Parmi les mots suivants, lesquels appartiennent à `number` ? Lesquelles n'y appartiennent pas ?
   .314, **.3E+4**, **0.5E−2**, **42**, **042**, **0000**, **E67**, **1E7e3**, **6E+1234**, **2E++3.4**
2. Dessiner un automate qui reconnaît le langage `integer`.
3. Dessiner un automate qui reconnaît le langage `number`.

2 Réinterprétation des transformations de la feuille « 04 »

Réinterpréter les constructions justifiées à la feuille « 04 » sur les systèmes d'automate, en utilisant la représentation des automates sous forme de 5-uplet, donne souvent des définitions formelles plus simples. Même, si *in fine*, la justification de la transformation à l'aide des systèmes d'équations reste pertinente. Nous considérons ici principalement les cas du calcul de l'intersection à l'exo 3 et de la déterminisation d'automate à l'exo 6. On revient aussi rapidement sur le calcul de complémentaire...

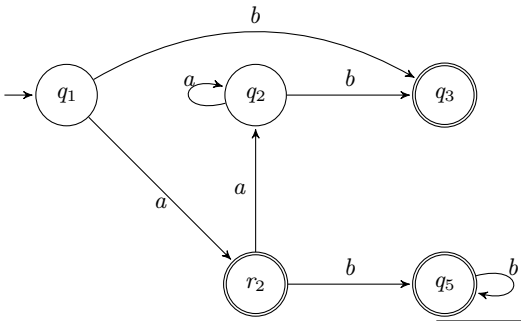
Exercice 3. Soit V un vocabulaire et soient $A_1 = (Q_1, V, s_1, \delta_1, F_1)$ et $A_2 = (Q_2, V, s_2, \delta_2, F_2)$ deux automates **sans ε-transition**. Définir l'automate A tel que $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$. Justifier votre réponse vis-à-vis de la construction donnée en feuille « 04 » (section 4).
NB dans la littérature, A est noté $A_1 \times A_2$, et on l'appelle *automate produit de A_1 et A_2*.

Exercice 4. Lors d'un changement de mot de passe, il est courant d'imposer des « règles de composition du mot de passe », c'est à dire des règles syntaxiques que doit satisfaire le mot de passe afin d'être accepté. Par exemple, on peut demander à ce qu'il y ait au moins un chiffre, une majuscule et un caractère « spécial » (dans une liste qui ne contient ni majuscule ni chiffre). Donner un automate (pas forcément déterministe) qui accepte l'ensemble des mots de passe vérifiant cette règle en exemple. On notera *ASCII* l'ensemble des caractère du code ASCII, A–Z l'ensemble des majuscules, 0–9 l'ensemble des chiffres et \$–# l'ensemble des caractères « spéciaux ».
Indication le langage à reconnaître correspond à

$$(ASCII^*.\text{A–Z}.ASCII^*) \cap (ASCII^*.\text{0–9}.ASCII^*) \cap (ASCII^*.\$–\#.ASCII^*)$$

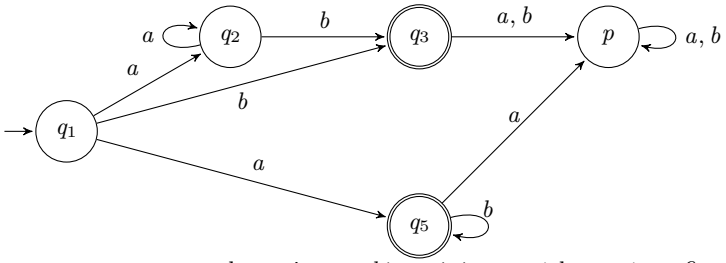
Exercice 5. On considère le calcul de complémentaire sur les automates donnés du langage $a^*b + ab^*$ en exemple 1.

1. En partant de l'automate déterministe incomplet suivant, dessiner l'automate du complémentaire :



   En déduire une expression régulière pour $\overline{a^*b + ab^*}$ par simple « lecture » de l'automate obtenu !

2. En partant de l'automate non-déterministe sans ε-transition, **mais rendu complet** avec un état puits, on obtient l'automate suivant :



   Sur cet automate complet mais *non-déterministe*, on échange états finaux et états non-finaux. Montrer que le langage de l'automate obtenu **n'est pas** $\overline{a^*b + ab^*}$ (il suffit de trouver un mot en trop ou un mot manquant).

Existence d'un automate déterministe et complet équivalent Soit $A = \langle Q, V, s, \delta, F\rangle$ un automate *sans ε-transition*. On pose la suite de définitions suivantes :

$$\begin{aligned}
Q_\Sigma &\stackrel{def}{=} \mathcal{P}(Q)\\
\delta_\Sigma &\stackrel{def}{=} ((P, a) \in Q_\Sigma \times V \mapsto \textstyle\bigcup_{q\in P} \delta(q, a))\\
F_\Sigma &\stackrel{def}{=} \{P \in Q_\Sigma \mid P \cap F \neq \varnothing\}\\
A_\Sigma &\stackrel{def}{=} \langle Q_\Sigma, V, \{s\}, \delta_\Sigma, F_\Sigma\rangle
\end{aligned}$$

A_Σ est un automate déterministe et complet (par abus de notation), car $\delta_\Sigma \in Q_\Sigma \times V \to Q_\Sigma$.
De plus, A_Σ accepte le même langage que A, c-à-d. plus formellement $\mathcal{L}(A_\Sigma) = \mathcal{L}(A)$.
NB la notation « A_Σ » souligne que comme vu à la feuille « 04 » et à l'exo suivant, chaque variable du système d'équations de A_Σ correspond à une somme/union de variables du système de A.

Exercice 6. (avancé) Démontrer la propriété $\mathcal{L}(A_\Sigma) = \mathcal{L}(A)$.
Indication : pour $P \in \mathcal{P}(Q)$, on pourra définir $L(P) \stackrel{def}{=} \bigcup_{q\in P} \mathcal{L}_A(q)$. Et prouver que $(L(P))_{P\in Q_\Sigma}$ est solution du système Δ_{A_Σ}. Par unicité de la solution, on obtient $\mathcal{L}_{A_\Sigma}(\{s\}) = L(\{s\}) = \mathcal{L}_A(s)$ qui implique l'égalité attendue.

3 Accessibilité et élagage des états inaccessibles

Remarquons que si A a n états, alors A_Σ en a 2^n (cf. compléments de la feuille « 02 » sur la cardinalité de $\mathcal{P}(E)$). Hormis dans des cas « pathologiques » comme celui de l'exo 13, on peut se convaincre que généralement A_Σ contient de très nombreux états inutiles. Par exemple, si A est déjà déterministe au départ, seul les $P \in \mathcal{P}(Q)$ tels que $\text{card}(P) \leq 1$ sont vraiment utiles. Plus généralement, si A est « proche » d'être déterministe, comme dans l'exemple 1, il n'y a pas forcément besoin de beaucoup plus d'états que pour A. Les concepts et les algorithmes issus de la théorie des graphes vont nous permettre de trouver comment éviter de « créer » tous ces états inutiles (cf. exo 12).

3.1 Vocabulaire sur les chemins

Étant donné un automate $A = \langle Q, V, s, \delta, F\rangle$, on appelle **chemin** de l'automate A une suite **finie** de triplets dans $Q\times(V\cup\{\varepsilon\})\times Q$ de la forme :

$$\text{« } (q_1, a_1, q_2)(q_2, a_2, q_3)\dots(q_n, a_n, q_{n+1}) \text{ » avec } \forall\, i \in \{1,\dots,n\},\, q_{i+1} \in \delta(q_i, a_i).$$

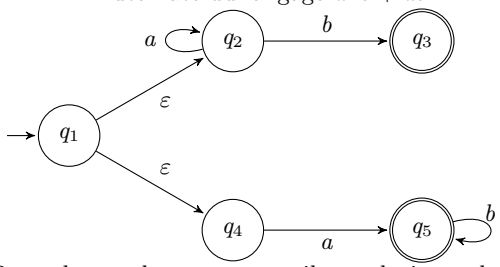
Sur un tel chemin (de A), on utilise le vocabulaire suivant :

- q_1 est **l'origine** (ou *état de départ*) du chemin, et q_{n+1} est **la destination** (ou *état d'arrivée*) du chemin ;
- le mot sur V^* « $a_1.a_2.\dots.a_n$ » est **la trace** du chemin ;
- n est la **longueur** du chemin : attention, en cas d'ε-transition, la longueur de la trace peut être strictement inférieure à n ;
- en résumé, on dira que c'est un « chemin de q_1 à q_{n+1} ayant pour trace $a_1.a_2.\dots.a_n$ » ;

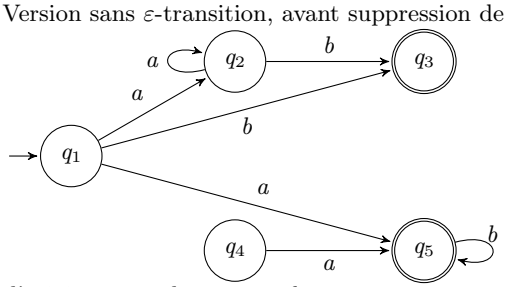
Par convention, on définit les chemins de longueur 0 en q (pour $q \in Q$ quelconque), comme ayant même origine et même destination q, et comme trace ε.

Exemples 2. Considérons les automates suivants :

Automate du langage $a^*b + ab^*$



Version sans ε-transition, avant suppression de q_4



Dans chacun des automates, il y a plusieurs chemins d'origine q_1 et de trace « ab » :

- sur l'automate de gauche, il y a (uniquement) deux chemins de longueur 3 :
  « $(q_1, \varepsilon, q_2)(q_2, a, q_2)(q_2, b, q_3)$ » et « $(q_1, \varepsilon, q_4)(q_4, a, q_5)(q_5, b, q_5)$ ».
- sur l'automate de droite, il y a (uniquement) deux chemins de longueur 2 :
  « $(q_1, a, q_2)(q_2, b, q_3)$ » et « $(q_1, a, q_5)(q_5, b, q_5)$ ».

Dans l'automate de gauche, il y a un unique chemin d'origine q_1 et de destination q_4 : le chemin de longueur 1 « (q_1, ε, q_4) ».
Dans l'automate de droite, il n'y a aucun chemin de q_1 à q_4. On dit que q_4 est **inaccessible** depuis q_1...

Ainsi, à partir de cette notion de chemin, on définit la terminologie suivante :

- Étant donné deux états q et q', on dit que q' est **accessible** à partir de q (dans A) si et seulement si il existe un chemin de q à q' (dans A).

— On dit qu'un état q est un **état puits** (de A) si et seulement si il n'existe aucun chemin de q à un état final. Ainsi, le langage d'un tel état puits vaut $\mathcal{L}_A(q) = \varnothing$.

— On dit que A est **initialement connecté** ssi tout état $q \in Q$ est accessible (dans A) depuis son état initial s.

Rem : on peut « supprimer » les états inaccessibles depuis l'état initial, ainsi que les états puits (avec toutes les transitions qui y arrivent ou qui en partent) sans changer le langage reconnu par l'automate. Attention, il peut être nécessaire d'avoir un état puits pour certaines transformations de l'automate comme le passage au complémentaire, mais dans ce cas, un seul état puits suffit (on peut « fusionner » les différents états puits) ; et cet état puits n'est pas nécessaire si l'automate est complet sans lui (l'état puits est dans ce cas inaccessible depuis l'état initial).

3.2 Formalisation de l'accessibilité sur les automates sans ε-transition

Intuitivement, cette notion de chemin est reliée à la sémantique des automates à cause de la propriété : $\forall\, q \in Q,\quad \mathcal{L}_A(q) = \{w \in V^* \mid \text{« il existe dans } A \text{ un chemin de } q \text{ à un état de } F \text{ ayant pour trace } w \text{ »}\}$.

Il est bien sûr possible de formaliser [2] la notion de chemin en toute généralité, mais pour simplifier, on se limite ici au cas des automates sans ε-transition (sachant qu'on a donné un algorithme pour les éliminer à la feuille « 04 »). Pour les automates sans ε-transition, la situation est en effet un peu plus simple, car la longueur d'un chemin est exactement donnée par la longueur de la trace (ce qui permet une simple définition par récurrence sur la longueur de la trace). Et on peut alors remplacer la notion de chemin par une notion un peu plus abstraite : « l'accessibilité de q' à partir de q par une trace w » (peu importe le chemin exact par lequel on y arrive), ce qu'on va noter formellement $q' \in \delta_A^*(q, w)$.

Accessibilité par une trace Soit $A = \langle Q, V, s, \delta, F\rangle$ un automate sans ε-transition. On définit une fonction $\delta_A^* : Q \times V^* \to \mathcal{P}(Q)$ où $\delta^*(q, w)$ est définie par récurrence sur la taille de w, pour tout $q \in Q$:

— pour tout $q \in Q$, $\delta_A^*(q, \varepsilon) \overset{def}{=} \{q\}$;

— pour $a \in V$ et $w \in V^*$, pour tout $q \in Q$, $\delta_A^*(q, a.w) \overset{def}{=} \bigcup_{q' \in \delta(q,a)} \delta_A^*(q', w)$;

Notons en particulier que pour $a \in V$, $\delta_A^*(q, a) = \bigcup_{q' \in \delta(q,a)} \delta_A^*(q', \varepsilon) = \bigcup_{q' \in \delta(q,a)} \{q'\} = \delta(q, a)$.

Propriétés de δ_A^* On peut montrer formellement les propriétés suivantes :

1. $\forall\, q \in Q,\quad \mathcal{L}_A(q) = \{w \in V^* \mid \delta_A^*(q, w) \cap F \neq \varnothing\}$;
2. si A est déterministe, alors $\forall\, w \in V^*, \forall\, q \in Q, \mathrm{card}(\delta_A^*(q, w)) \leq 1$ et si A est de plus complet, alors $\forall\, q \in Q, \forall\, w \in V^*, \mathrm{card}(\delta_A^*(q, w)) = 1$;
3. $\forall\, w \in V^*, \forall\, w' \in V^*, \forall\, q \in Q, \delta_A^*(q, w.w') = \bigcup_{q' \in \delta_A^*(q,w)} \delta_A^*(q', w')$;

Exercice 7. (avancé) Prouver la propriété 1 de δ_A^*.

Indication : on pourra définir $\hat{L}_A(q) \overset{def}{=} \{w \in V^* \mid \delta_A^*(q, w) \cap F \neq \varnothing\}$. Et prouver que $(\hat{L}_A(q))_{q \in Q}$ est solution du système Δ_A. Par unicité de la solution, on obtient le résultat annoncé.

Exercice 8. Prouver les propriété 2 et 3 de δ_A^*.

Exercice 9. Formaliser les concepts « intuitifs » suivants en utilisant δ_A^* :

1. « l'état q' est *accessible* à partir de l'état q » ;
2. « l'état q est un *état puits* » ;
3. « l'automate A est *initialement connecté* ».

Exercice 10. En conséquence de la propriété 2, si A est déterministe et complet, on pourra abusivement considérer que $\delta_A^* \in Q \times V^* \to Q$. En particulier, on considère maintenant que $\delta_{A_\Sigma}^* \in \mathcal{P}(Q) \times V^* \to \mathcal{P}(Q)$. Avec cette convention [3], montrer : $\forall\, w \in V^*, \delta_{A_\Sigma}^*(\{s\}, w) = \delta_A^*(s, w)$.
Ça constitue une preuve alternative que $\mathcal{L}(A_\Sigma) = \mathcal{L}(A)$.

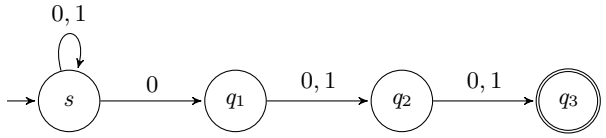
2. Pour voir une formalisation en toute généralité, on pourra se référer aux diapos des cours 2 et 3 de TL1 pour les étudiants 1A. Voir :https://chamilo.grenoble-inp.fr/courses/ENSIMAG3MMTL1/.

3. Sans la convention, il faudrait montrer à la place $\forall\, w \in V^*, \delta_{A_\Sigma}^*(\{s\}, w) = \{\delta_A^*(s, w)\}$—se passer de la convention est juste bureaucratique !

Exercice 11. Grâce au résultat de l'exo 10, on souhaite généraliser la fonction `accepte(A, w)` implémentée en Python à la feuille « 04 » sur les automates déterministes et complets, pour qu'elle fonctionne sur les automates sans ε-transition quelconques (e.g. ni forcément déterministes ni forcément complets). Chaque automates sans ε-transition sera juste donné comme un 5-uplet `(Q, V, s, delta, F)` où :

- `Q`, `V`, `F` sont des « `set` » Python qui représentent respectivement Q, V et F ;
- `s` correspond à s ;
- `delta` est un dictionnaire de dictionnaires de « `set` » tel que `delta[q][a]` représente $\delta(q, a)$.

On impose que pour l'ensemble des « `q in Q` », il y ait un dictionnaire « `delta[q]` ». Par contre, on n'impose pas la présence d'un « `delta[q][a]` » pour tout « `a in V` » (ça simplifie un peu l'écriture pour les automates incomplets). Par exemple, sur le vocabulaire $V \stackrel{def}{=} \{0, 1\}$, l'automate suivant (non-déterministe en s et incomplet en q_3) est représenté en Python par le 5-uplet `a_L3` ci-dessous :




```
a_L3 = ({'s', 'q1', 'q2', 'q3'},
        {'0', '1'},
        's',
        { 's': { '0': { 's', 'q1'}, '1': {'s'} },
          'q1': { '0': { 'q2'}, '1': {'q2'} },
          'q2': { '0': { 'q3'}, '1': {'q3'} },
          'q3': {} },
        { 'q3' })
```

Adapter la fonction `accepte(A, w)`, en s'inspirant de δ^*_A, notamment l'égalité ci-dessous (un cas particulier de sa propriété 3) :

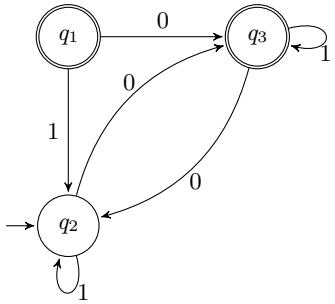
$$\delta^*_A(s, w.a) = \bigcup_{q \in \delta^*_A(s,w)} \delta(q, a)$$

Quel le lien entre `accepte(A, w)` et A_Σ ?

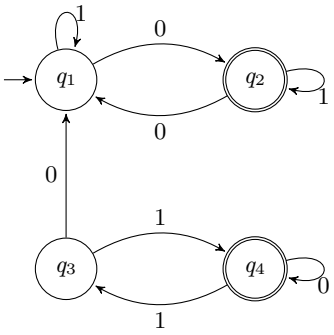
3.3 Élagage et renommage « canonique » des automates déterminisés

On considère les deux codes d'automate déterministe (et complet) `A0` et `A1` (accompagnés de leur représentation graphique) ci-dessous qui reconnaissent le langage « $(1 + 01^*0)^*01^*$ » :

```
A0=({'q1','q2','q3'},
    {'0','1'},
    'q2',
    { 'q1':{'0':'q3', '1':'q2'},
      'q2':{'0':'q3', '1':'q2'},
      'q3':{'0':'q2', '1':'q3'} },
    { 'q1','q3' })
```




```
A1=({'q1','q2','q3','q4'},
    {'1','0'},
    'q1',
    { 'q1':{'0':'q2', '1':'q1'},
      'q2':{'0':'q1', '1':'q2'},
      'q3':{'0':'q1', '1':'q4'},
      'q4':{'0':'q4', '1':'q3'} },
    { 'q2', 'q4' })
```

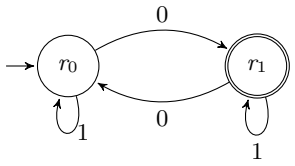


Ces deux automates sont en fait équivalents modulo :

1. élagage des états inaccessibles depuis l'état initial (`q1` pour `A0` ; `q3` et `q4` pour `A1`) ;
2. renommage des états.

La fonction `rename` en figure 1 prend en entrée un automate `A` **déterministe** (pas forcément complet) et retourne un automate déterministe qui reconnaît le même langage que `A`, qui est *initialement connecté* et qui est issu d'un renommage « canonique » (sous l'hypothèse que `V` est muni d'un ordre total). « Renommage canonique », cela signifie que tous les automates équivalents à `A` modulo renommage sont renommés exactement de la même façon par `rename` (en particulier, `rename` est idempotent). Très concrètement, `rename(A0)` et `rename(A1)` produisent exactement le même 5-uplet donné ci-dessous :

```
({'r1','r0'},
 {'0','1'},
 'r0',
 { 'r0':{'0':'r1', '1':'r0'},
   'r1':{'0':'r0', '1':'r1'} },
 { 'r1' })
```



On peut donc *réduire* l'équivalence de `A0` et `A1` modulo élagage des états inaccessibles et renommage au simple test Python « `rename(A0)==rename(A1)` ».

```python
def rename(A):
    (Q, V, s, delta, F) = A
    rs = 'r'+repr(0) # renommage de l'axiome
    acc = [s]        # liste (dans l'ordre) des anciens noms (renommés)
    ren = { s: rs }  # dict des anciens noms vers les nouveaux
    rdelta = dict()
    todo = 0  # prochain état renommé dont il faut générer l'équation
    while todo < len(acc):
        q = acc[todo]
        tq = delta[q]
        rtodo = ren[q]
        rdelta[rtodo] = dict()
        tr = rdelta[rtodo]
        for a in sorted(V):
            if not a in tq:
                continue # automate incomplet
            q0 = tq[a]
            if q0 in ren:
                rq0 = ren[q0]  # q0 déjà renommé
            else:
                rq0 = 'r'+repr(len(acc)) # nouveau nom pour q0
                acc.append(q0)
                ren[q0] = rq0  # renommage q0 effectué
            tr[a] = rq0 # renommage de la transition
        todo += 1  # prochaine équation à visiter
    rQ = set(ren[q] for q in acc)
    rF = set(ren[q] for q in acc if q in F)
    return (rQ, V, rs, rdelta, rF)
```

FIGURE 1 – Élagage et renommage « canonique » des automates déterministes

Pour calculer l'ensemble des états de `Q` accessibles depuis l'état initial `s` (c-a-d. $\{\delta^*(s,w) \mid w \in V^*\}$), `rename` procède de façon classique par un *parcours en largeur sur le graphe*[4] de l'automate déterministe. Concrètement, `rename` calcule cet ensemble dans la *liste* `acc`, l'ordre des états dans `acc` correspondant à l'ordre de « *marquage* » dans le parcours en largeur. Voilà le principe de ce *marquage* : on va *visiter* une et

4. https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur

une seule fois chaque état q *marqué* dans `acc` de manière à ce que tous les états de $\bigcup_{a\in V}\delta^*(s,a)$ soient eux-mêmes *marqués* (dans `acc`). Autrement dit, quand on visite q : les $q' \in \bigcup_{a\in V}\delta^*(s,a)$ qui ne sont pas déjà marqués doivent être ajoutés à `acc` ; et ils devront à leur tour être visités. Sur le code de la figure 1, chaque tour de boucle effectue la visite de l'état `acc[todo]`, les états de `acc` d'indice strictement inférieur à `todo` ayant déjà été visités. Initialement, avant la boucle, `acc` contient un seul état marqué qui est `s` (d'indice 0) et c'est le premier état à visiter.

La fonction `rename` effectue le renommage des équations à la volée, pendant le parcours en largeur : l'indice i d'un état q dans `acc` correspond à renommer q en « `r`i », l'association $q \mapsto$ « `r`i » étant enregistrée dans `ren`. Ainsi, le renommage est induit par l'ordre de marquage. L'ordre de marquage lui est uniquement déterminé à partir de l'ordre de parcours de $\bigcup_{a\in V}\delta^*(s,a)$ pour chaque état q. Ici, l'automate est déterministe : à chaque $a \in V$ est engendre au plus un état de $\delta^*(q,a)$ et l'ordre de parcours des $a \in V$ est déterminé par l'ordre total sur V (cf. le **sorted**`(V)` ligne 14). C'est ce qui rend le renommage canonique. Au final, `rename` effectue l'élagage et le renommage en une seule passe sur l'automate. Remarquons que si l'automate `A` est complet, alors `rename(A)` aussi, puisqu'il préserve toutes les transitions des états accessibles.

Exercice 12. En vous inspirant de `rename` de la figure 1 et de `accepte` à l'exo 11, écrire une fonction `determ(A)` qui prend en entrée un automate A sans ε-transition (codé en PYTHON comme à l'exo 11), et calcule directement `rename(`A_Σ`)`, sans bien sûr calculer entièrement A_Σ (pour éviter d'exploser si on peut). L'idée sous-jacente est la même : la fonction `determ` peut appliquer exactement le même algorithme que `rename` en remplaçant $\delta^*_{A_\Sigma}(\{s\},w)$ (qui apparaîtrait dans les calculs de par `rename(`A_Σ`)`) par $\delta^*_A(s,w)$. La fonction `determ` ainsi obtenue est idempotente (et sur un automate déterministe complet, elle est équivalente à `rename`, y compris en temps d'exécution - à un facteur constant près) [5].

Exercice 13. (avancé) Cet exercice consiste à montrer que pour tout $k > 0$, il existe un automate non-déterministe à $k+1$ états tel que tout automate déterministe complet équivalent contient au moins 2^k états (la déterminisation des automates « explose » dans le pire cas).
Pour $k > 0$, soit L_k le langage constitué des mots sur $\{0,1\}$ de longueur au moins k, et dont le $k^{\text{ième}}$ symbole **en partant de la fin** est un 0. Par exemple, 00101 et 100110111 sont dans L_3. Formellement,

$$L_k \stackrel{def}{=} \{0,1\}^*.\{0\}.\{0,1\}^{k-1}$$

1. Construire un automate (non-déterministe) à $k+1$ états qui reconnaît L_k.
2. Donner un automate déterministe et complet qui calcule L_3.
3. On cherche à borner la taille minimale d'un automate déterministe complet reconnaissant L_k. Soit $A = (Q, \{0,1\}, \delta, \{q_0\}, F)$ un automate déterministe complet reconnaissant L_k. On définit $f : \{0,1\}^k \to Q$, qui à tout mot u de longueur k associe $\delta^*(q_0,u)$. Autrement dit, $f(u)$ est l'état atteint par le chemin de trace u dans A, partant de q_0. Montrer que f est injective. En déduire une borne inférieure de la taille de A.

4 Minimisation

Le calcul de `determ` (de l'exo 12) sur l'automate sans ε-transition non-déterministe de l'exemple 2 donne un automate déterministe complet (sur le vocabulaire $\{a,b\}$) ayant pour système d'équations ci-contre.

$$\begin{cases} r_0 = a.r_1 + b.r_2 \\ r_1 = \varepsilon + a.r_3 + b.r_4 \\ r_2 = \varepsilon + a.r_5 + b.r_5 \\ r_3 = a.r_3 + b.r_2 \\ r_4 = \varepsilon + a.r_5 + b.r_6 \\ r_5 = a.r_5 + b.r_5 \\ r_6 = \varepsilon + a.r_5 + b.r_6 \end{cases}$$

Trivialement, on peut simplifier ce système d'équations, en remarquant que $r_6 = r_4$ et en éliminant r_6 : cela impacte uniquement l'équation de r_4 qui devient « $r_4 = \varepsilon + a.r_5 + b.r_4$ » ; et l'équation de r_6 qui est supprimée. Modulo l'état puits r_5 et au renommage près des états, on obtient alors le même système d'automate déterministe que celui donné à l'exemple 1.

On décrit ici un algorithme général pour calculer les états *équivalents* d'un automate déterministe complet (comme r_4 et r_6), et les « fusionner » pour arriver à un **automate déterministe complet minimal**.

5. A ceci près qu'il faut passer de l'encodage des automates déterministes à celui des automates non-déterministes via une fonction `lift`. L'idempotence s'écrit alors **assert**`(determ(lift(determ(A)))==determ(A))`. Lorsque `A` est complet dans l'encodage des automates déterministes, on a **assert**`(determ(lift(A)))==rename(A)`.

Définition de l'automate minimal Soit A un automate **déterministe et complet** (δ_A noté avec abus de notation).
On pose la suite de définitions suivante :

$q \sim_A q' \stackrel{def}{=} (L_A(q) = L_A(q'))$ ($\sim_A$ est une *relation d'équivalence*) [6]
$[q]_A \stackrel{def}{=} \{q' \in Q \mid q \sim_A q'\}$ (on l'appelle *classe d'équivalence* de q) [7]
$Q_{/\sim_A} \stackrel{def}{=} \{[q]_A \mid q \in Q\}$ (on l'appelle *ensemble quotient* de Q par $\sim_A$) [8].
$\delta_{/\sim_A} \stackrel{def}{=} ([q], a) \in Q_{/\sim_A} \times V \mapsto [\delta_A(q, a)]_A$
$F_{/\sim_A} \stackrel{def}{=} \{[q]_A \mid q \in F\}$ (c'est *ensemble quotient* de F par $\sim_A$)
$A_{/\sim} \stackrel{def}{=} \langle Q_{/\sim_A}, V, [s]_A, \delta_{/\sim_A}, F_{/\sim_A}\rangle$ (c'est *l'automate minimal de* A, cf. paragraphe suivant).

NB : ci-dessus, pour que la définition de $\delta_{/\sim_A}$ en tant que fonction de $Q_{/\sim_A} \times V \to Q_{/\sim_A}$ soit valide [9], il faut montrer « $[q]_A = [q']_A \Rightarrow [\delta_A(q, a)]_A = [\delta_A(q', a)]_A$ » c-à-d. le « résultat » retourné par $\delta_{/\sim_A}([q], a)$ ne dépend pas du choix de q dans $[q]_A$. Ou de manière équivalente,

$$q \sim_A q' \Rightarrow \delta_A(q, a) \sim_A \delta_A(q', a)$$

Pour cela, on remarque que $q_1 \sim_A q_2 \Leftrightarrow (\forall\, w,\, \delta^*_A(q_1, w) \in F \Leftrightarrow \delta^*_A(q_2, w) \in F)$.
Et donc, cette dernière implication découle du fait que $\forall\, q_0, \delta^*_A(\delta_A(q_0, a), w) = \delta^*_A(q_0, a.w)$.

Propriétés de $A_{/\sim}$ L'automate $A_{/\sim}$ s'appelle « l'automate déterministe complet minimal » de $\mathcal{L}(A)$, à cause des propriétés suivantes

1. $A_{/\sim}$ est déterministe et complet, et si A est initialement connecté alors $A_{/\sim}$ aussi ;
2. $\mathcal{L}(A_{/\sim}) = \mathcal{L}(A)$;
3. Si A_1 et A_2 sont deux automates déterministes complets et *initialement connectés* qui vérifient $\mathcal{L}(A_1) = \mathcal{L}(A_2)$, alors $A_{1/\sim}$ et $A_{2/\sim}$ sont égaux à renommage près des états. Autrement dit, après renommage canonique on a égalité syntaxique des 5-uplets.

Exercice 14. Démontrer les propriétés 1 et 2. Pour la propriété 3, voir la preuve « simple » mais « conceptuelle » de https://en.wikipedia.org/wiki/Myhill%E2%80%93Nerode_theorem.

Calcul de l'automate minimal Il existe plusieurs algorithmes pour calculer l'automate minimal [10], l'algorithme que nous présentons ici correspond à celui de Moore (ce n'est pas le plus efficace, mais le plus simple à comprendre). Soit A un automate déterministe complet fixé. Pour simplifier, on notera ci-dessous « $\sim$ » au lieu de « $\sim_A$ », et « $[q]$ » au lieu de « $[q]_A$ », et « $Q_{/\sim}$ » au lieu de « $Q_{/\sim_A}$ ». Le principe de l'algorithme est de calculer $Q_{/\sim}$ par *approximations successives* de $\sim$. Ainsi, pour $k \in \mathbb{N}$, on pose

$$q \sim_k q' \stackrel{def}{=} \forall\, w \in V^*, |w| \leq k \Rightarrow (\delta^*_A(q, w) \in F \Leftrightarrow \delta^*_A(q', w) \in F)$$
$$[q]_k \stackrel{def}{=} \{q' \in Q \mid q \sim_k q'\} \qquad Q_{/\sim_k} \stackrel{def}{=} \{[q]_k \mid q \in Q\}$$

On dérive alors facilement les propriétés suivantes :

$$q \sim q' \Leftrightarrow \forall\, k \in \mathbb{N}, q \sim_k q' \qquad [q] = \bigcap_{k\in\mathbb{N}} [q]_k$$
$$q \sim_{k+1} q' \Rightarrow q \sim_k q' \qquad [q]_{k+1} \subseteq [q]_k$$

En fait, on établit ci-dessous que $Q_{/\sim} = Q_{/\sim_{\mathrm{card}(Q)-1}}$. On commence par remarquer que $\sim_k$ est calculable par récurrence sur k :

6. https://fr.wikipedia.org/wiki/Relation_d%27%C3%A9quivalence
7. https://fr.wikipedia.org/wiki/Relation_d%27%C3%A9quivalence#Classe_d'%C3%A9quivalence
8. https://fr.wikipedia.org/wiki/Relation_d%27%C3%A9quivalence#Ensemble_quotient
9. Comme expliqué dans les compléments de la feuille « 02 », la notation « $\ldots \in \ldots \mapsto \ldots$ » est une abréviation. Dans la définition ci-dessus, la définition de $\delta_{/\sim_A}$ empile des « abus de notation » très fréquents en maths mais un peu dangereux. En toute rigueur, il faut « dé-sucrer » sa définition en :

$$\delta_{/\sim_A} \stackrel{def}{=} \{\ (([q]_A, a), [\delta_A(q, a)]_A)\ \mid\ q \in Q \wedge a \in V\ \}$$

ce qui définit $\delta_{/\sim_A} \in \mathcal{P}((Q_{/\sim_A} \times V) \times Q_{/\sim_A})$. Pour montrer que $\delta_{/\sim_A}$ est une fonction, il faut montrer

$$\forall\, q \in Q, \forall\, a \in V, \delta_{/\sim_A} \cap (\{([q]_A, a)\} \times Q_{/\sim_A}) = \{(([q]_A, a), [\delta_A(q, a)]_A)\}$$

et ce qui se ramène à l'implication entre guillemets ci-dessus.
10. cf. https://en.wikipedia.org/wiki/DFA_minimization

— $q \sim_0 q'$ ssi $(\delta_A^*(q,\varepsilon) \in F \Leftrightarrow \delta_A^*(q',\varepsilon) \in F)$
ssi $(q \in F \Leftrightarrow q' \in F)$

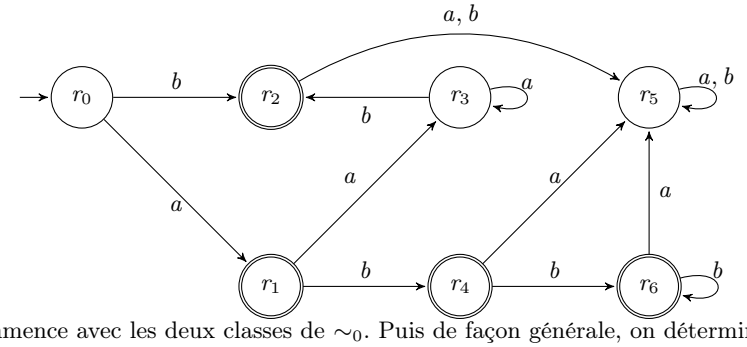
— $q \sim_{k+1} q'$ ssi $\forall\, w \in V^*, |w| \leq k \Rightarrow \begin{pmatrix} \delta_A^*(q,w) \in F \Leftrightarrow \delta_A^*(q',w) \in F \\ \wedge\, \forall\, a \in V, \delta_A^*(q,a.w) \in F \Leftrightarrow \delta_A^*(q',a.w) \in F \end{pmatrix}$
ssi $q \sim_k q' \wedge \forall\, a \in V, \forall\, w \in V^*, |w| \leq k \Rightarrow (\delta_A^*(\delta(q,a),w) \in F \Leftrightarrow \delta_A^*(\delta(q',a),w) \in F)$
ssi $q \sim_k q' \wedge \forall\, a \in V, \delta(q,a) \sim_k \delta(q',a)$

Autrement dit :

— On commence le calcul avec $Q_{/\sim_0} = \{\overline{F}_{/\sim_0}, F_{/\sim_0}\}$, c-à-d. les deux classes de $\sim_0$.
— Puis, les classes de $Q_{/\sim_{k+1}}$ s'obtiennent en subdivisant chaque classe de $Q_{/\sim_k}$. Concrètement, étant donné deux états q et q' vérifiant $[q]_k = [q']_k$, on a $[q]_{k+1} = [q']_{k+1}$ ssi $\forall\, a \in V, [\delta(q,a)]_k = [\delta(q',a)]_k$.
— Lorsque $Q_{/\sim_{k+1}} = Q_{/\sim_k}$ (c-à-d. $\forall\, q, [q]_{k+1} = [q]_k$), on a $Q_{/\sim} = Q_{/\sim_k}$.

Remarquons que si $\text{card}([q]_k) = 1$, alors nécessairement $[q]_{k+1} = [q]_k$, car aucune subdivision n'est alors possible ! On en déduit que la convergence $Q_{/\sim_{k+1}} = Q_{/\sim_k}$ s'obtient pour $k \leq \text{card}(Q) - 1$.

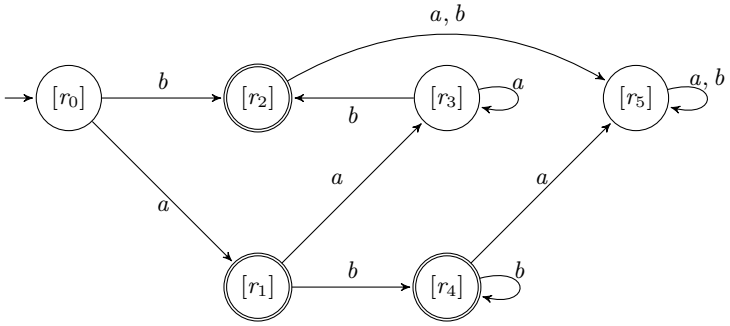
Exemple 3. Illustrons ce calcul sur l'automate déterministe et complet de notre exemple



On commence avec les deux classes de $\sim_0$. Puis de façon générale, on détermine la classe $\sim_{k+1}$ en fonction de la classe $\sim_k$ et de la « signature » de chaque état, c.-à-d. des classes vers lesquelles on arrive après une transition par chacun des symboles du vocabulaire. Par exemple, sur le vocabulaire $\{a, b\}$ avec $a < b$, la signature $[q]_k[q']_k$ signifie qu'en lisant un a, on arrive dans la classe $[q]_k$ et qu'en lisant un b, on arrive dans la classe $[q']_k$. Les états d'une même classe de $\sim_k$ qui ont même signature reste ensemble dans la même classe de $\sim_{k+1}$. Sauf que, quand une classe ne contient qu'un élément, on ne calcule pas non sa signature (puisqu'elle ne peut pas se subdiviser) et on la note directement $[q]$ au lieu de $[q]_k$. Par convention, on choisit comme représentant de la classe l'état de plus petit numéro dans cette classe.

$Q_{/\sim_0}$	$\{r_0,$	$r_3,$	$r_5,\}$	$\{r_1,$	$r_2,$	$r_4,$	$r_6\}$
noms des classes		$[r_0]_0$				$[r_1]_0$	
« signatures »	$[r_1]_0[r_1]_0$	$[r_0]_0[r_1]_0$	$[r_0]_0[r_0]_0$	$[r_0]_0[r_1]_0$	$[r_0]_0[r_0]_0$	$[r_0]_0[r_1]_0$	$[r_0]_0[r_1]_0$
$Q_{/\sim_1}$	$\{r_0\}$	$\{r_3\}$	$\{r_5\}$	$\{r_2\}$	$\{r_1,$	$r_4,$	$r_6\}$
noms des classes	$[r_0]$	$[r_3]$	$[r_5]$	$[r_2]$		$[r_1]_1$	
« signatures »					$[r_3][r_1]_1$	$[r_5][r_1]_1$	$[r_5][r_1]_1$
$Q_{/\sim_2}$	$\{r_0\}$	$\{r_3\}$	$\{r_5\}$	$\{r_2\}$	$\{r_1\}$	$\{r_4,$	$r_6\}$
noms des classes	$[r_0]$	$[r_3]$	$[r_5]$	$[r_2]$	$[r_1]$	$[r_4]_2$	
« signatures »						$[r_5][r_4]_2$	$[r_5][r_4]_2$
$Q_{/\sim_3}$	$Q_{/\sim_2}$						

Ci-dessus, pour les signatures calculées sur $\sim_0$, les états r_0, r_3, r_5, r_2 ont chacun une signature unique au sein de leur classe de $\sim_0$: donc, ils deviennent chacun une classe singleton dans $\sim_1$. Les autres (r_1, r_4 et r_6) ont même signature au sein de leur classe commune de $\sim_0$: ils forment donc une seule classe dans $\sim_1$. D'après les signatures sur $\sim_1$, r_1 n'a pas la même signature que les deux autres et donc se séparent d'eux dans $\sim_2$. Les signatures sur $\sim_2$ restent identique pour r_4 et r_6, donc ils restent ensemble. Au final, on trouve qu'en fusionnant r_4 et r_6, on obtient l'automate minimal, tel que dessiné ci-dessous :



Exercice 15. Dans le calcul de minimisation de l'exemple 3 précédent, on a trouvé $r_1 \sim_1 r_4$ mais $r_1 \not\sim_2 r_4$. Cela signifie qu'il existe un mot w de longueur exactement 2 tel que $w \in \mathcal{L}_A(r_1) \not\Leftrightarrow w \in \mathcal{L}_A(r_4)$. Quel est ce mot, comment le trouve-t-on à partir des signatures de $\sim_1$ et $\sim_0$?

Exercice 16. Minimiser les automates suivants :