Systèmes réguliers d'(in)équations sur les langages et introduction aux automates finis

Feuille « 04 » (du 20 septembre 2024)

On formalise ici les idées et techniques introduites à la feuille « 03 » sur les systèmes d'équations associées aux langages réguliers. Et on introduit le cœur du cours : les automates finis.

1 Définitions

Syntaxe Soient V un vocabulaire, n un entier supérieur ou égal à 1, et $X_1, \ldots, X_n$ n symboles de variables 2 à 2 distincts et hors de V. On appelle système **régulier** d'équations, un système de la forme :

$$\begin{cases} X_1 = t_{1,1} + \ldots + t_{1,p_1} \\ \cdots \\ X_n = t_{n,1} + \ldots + t_{n,p_n} \end{cases}$$

où

- chaque p_i est un entier (éventuellement nul) ;
- chaque $t_{i,j} \in V^* \cup V^*.\{X_1, \ldots, X_n\}$.

Par convention, si $p_i = 0$, on écrira l'équation correspondantes « $X_i = \varnothing$ ».

Par définition,

- une solution du système est un n-uplet langages $(L_1, \ldots, L_n)$ sur V tels que quand on remplace chaque X_i par L_i dans le système, alors les n équations sont satisfaites ;
- la plus petite solution d'un tel système est un n-uplet $(L_1, \ldots, L_n)$ qui est solution du système et tel que pour tout n-uplet solution, $(L'_1, \ldots, L'_n)$, on a pour tout i, $L_i \subseteq L'_i$.

En utilisant Knaster-Tarski, on peut justifier (cf. en période 2) qu'un tel système d'équations admet bien toujours une plus petite solution, et que cette solution est aussi la plus petite solution du système d'inéquations obtenu en remplaçant les égalités « $=$ » du système par des inégalités « $\supseteq$ ».

Sémantique Le langage défini par un tel système correspond à L_1 où $(L_1, \ldots, L_n)$ est la plus petite solution du système. Autrement dit, c'est X_1 qui correspond au langage défini par le système : les autres variables définissent des langages auxiliaires utilisés dans la définition de X_1. On appelle X_1 **l'axiome** ou encore **la variable initiale**.

Exemple 1. Au dernier exercice de la feuille « 03 », on a vu que le système suivant

$$\begin{cases} X = X + \mathtt{a}X + \mathtt{b}Y + \varepsilon \\ Y = \mathtt{b}X + \mathtt{a}Y \end{cases}$$

définit le langage régulier $(\mathtt{a} + \mathtt{ba}^*\mathtt{b})^*$.

Vocabulaire Une autre interprétation consiste à voir chaque équation comme un processus (mutuellement inductif) de « production » de mots : le langage associé au système étant alors l'ensemble des mots produits par l'équation de X_1. Le « $+$ » dans les équations est alors interprété comme un « choix non-déterministe » entre différentes formes de mots. Cette interprétation induit le vocabulaire suivant :

- on appelle les $t_{i,j}$ « alternatives » (une équation représente un choix fini p_j alternatives) ;
- on appelle « transitions », les alternatives de la forme $V^*.\{X_1, \ldots, X_n\}$ (ça correspond à une transition vers une autre équation) ;
- on appelle « alternative d'arrêt », les alternatives de la forme V^* (ici, pas de transition vers une autre équation).

Système sans ε-transition Une ε-transition est un $t_{i,j}$ de la forme « $\varepsilon.X_k$ », qu'on notera en fait « X_k » en pratique.
Plus formellement, un système régulier est dit sans ε-transition ssi il vérifie :

$$\text{toute alternative vérifie } t_{i,j} \in V^* \cup V^+.\{X_1, \ldots, X_n\}$$

Exemple 2. Le système de l'exemple 1 a une ε-transition dans l'équation de X. Le système suivant sans ε-transition définit le même langage :

$$\begin{cases} X = \mathtt{a}X + \mathtt{b}Y + \varepsilon \\ Y = \mathtt{b}X + \mathtt{a}Y \end{cases}$$

Unicité de la solution des systèmes sans ε-transition En s'appuyant sur l'unicité de l'équation d'Arden, on peut montrer qu'un système sans ε-transition a une unique solution (on admet ce résultat ici).

Comme étudié à la feuille « 03 », quand un système n'a pas une unique solution, même si le système d'équations et le système d'inéquations ont la même plus petite solution, pour effectuer des transformations sur le système, il est préférable de se restreindre à manipuler des inéquations : certaines égalités induisent des substitutions qui ne préservent pas la plus petite solution, ce sont justement les substitutions qu'on perd quand on passe aux inégalités.

En conséquence, dès qu'on effectuera des transformations sur un système avec ε-transition, on remplacera les « $=$ » en « $\supseteq$ » sauf si on est sûr que le système a une unique solution. On réservera les égalités pour les systèmes qui ont une unique solution, notamment ceux sans ε-transition.

2 Éliminations basées sur le lemme d'Arden (généralisé)

Soient $A \subseteq V^*$ et $B \in \mathcal{P}(V^*) \to \mathcal{P}(V^*)$. D'après le lemme d'Arden généralisé (cf. feuille « 03 »), on peut remplacer l'inéquation $X \supseteq A.X + B(X)$ par l'inéquation $X \supseteq A^*B(X)$ en préservant la plus petite solution. Ce remplacement permet typiquement d'éliminer une occurrence de X dans le système. Voire, si B ne dépend en fait pas de X, d'éliminer complètement X de sa propre équation. Ce remplacement est à la base de deux transformations importantes sur les systèmes réguliers.

2.1 Élimination des ε-transitions

En fait, l'unique transformation qu'on effectue sur les systèmes réguliers d'inéquations, c'est d'éliminer leur ε-transitions. Cette transformation permet en effet de se ramener à un système avec une unique solution, et donc de pouvoir tranquillement passer à des équations, ce qui offre ensuite plus de possibilités de « simplifications » du système.

Lemme d'élimination des ε-transitions Pour tout système régulier avec ε-transition, on peut calculer un système régulier sans ε-transition qui reconnaît le même langage : on dit que le système obtenu est *équivalent* au système de départ (même si ce n'est que la plus petite solution qui est préservée).

La preuve de ce lemme est donnée ci-dessous sous la forme d'un algorithme qui, à partir d'un système régulier donné quelconque, construit un système régulier équivalent sans ε-transition. Son principe s'appuie sur le fait que, d'après le lemme d'Arden généralisé, on peut remplacer $X \supseteq X + B(X)$ en $X \supseteq B(X)$, en préservant la plus petite solution, ce qui pour effet d'enlever une ε-transition « trivialement redondante ». On notera ici systématiquement « X » au lieu de « $\varepsilon.X$ ».

Algorithme d'élimination des ε-transitions Initialement, sur chaque inéquation d'une variable X, on retire de son membre droit toute alternative réduite à ce même X (ε-transition trivialement

redondante) et on simplifie ce membre droit en factorisant les alternatives identiques (e.g. $t_1+t_2+t_1 = t_1 + t_2$). Ensuite on itère la transformation suivante jusqu'à obtenir un système sans ε-transition : cette transformation consiste à choisir une variable Y et à **éliminer** dans tous les membres droits des inéquations **toutes les alternatives** réduites à « Y ». Concrètement, en supposant que l'inéquation de Y soit de la forme « $Y \supseteq \Sigma_j t'_j$ », cette élimination consiste, dans à remplacer chaque inéquation s'écrivant « $X \supseteq Y + \Sigma_i t_i$ » l'alternative Y par $\Sigma_j t'_j$, c-à-d. à remplacer l'inéquation de X par « $X \supseteq (\Sigma_j t'_j) + (\Sigma_i t_i)$ » ; puis, si cette substitution introduit l'alternative « X » (dans les t'_j), à retirer aussitôt cette alternative X de la somme (ε-transition trivialement redondante) ; et enfin, à simplifier la somme obtenue en factorisant les alternatives identiques. Remarquons que le remplacement de Y par son membre droit $\Sigma_j t'_j$ dans l'inéquation de X préserve la plus petite solution du système : en effet toute solution X de l'une des inéquations satisfait aussi l'autre, sous la condition que Y est le *plus petit ensemble* satisfaisant son inéquation (qui elle n'est pas changée).

Cet algorithme termine, car à chaque itération on élimine définitivement toutes les ε-transition « Y » du Y choisi. Donc, il y a au plus n itérations (avec au plus $n-1$ remplacements à chaque itération).

Exercice 1. Soit $V = \{a, b, c\}$. Appliquer cet algorithme sur :

$$\begin{cases} X_1 \supseteq X_2 \\ X_2 \supseteq a.X_1 + c.X_3 + X_3 \\ X_3 \supseteq X_4 + \varepsilon \\ X_4 \supseteq b.X_4 + c.X_3 \end{cases}$$

Exercice 2. Soit $V = \{a, b, c\}$. Appliquer l'algorithme sur :

$$\begin{cases} X_1 \supseteq X_1 + X_2 + X_3 + aaX_1 \\ X_2 \supseteq X_1 + X_2 + X_3 + bbX_2 \\ X_3 \supseteq X_1 + X_2 + X_3 + c \end{cases}$$

Sur le système sans ε-transition obtenu, exploiter le fait qu'il a une solution unique pour obtenir un système régulier sans ε-transition le plus simple possible. En déduire une expression régulière simple reconnue par le système.

Sur l'exemple précédent, on peut se rendre compte que si on applique mal l'algorithme, l'élimination des ε-transition peut ne pas terminer. Par exemple, à partir du système

$$\begin{cases} X_1 \supseteq X_2 + X_3 + aaX_1 \\ X_2 \supseteq X_1 + X_3 + bbX_2 \\ X_3 \supseteq X_1 + X_2 + c \end{cases}$$

si on remplace *simultanément* X_2 et X_3 (par leur membre droit) dans X_1, on obtient alors :

$$\begin{cases} X_1 \supseteq X_2 + X_3 + aaX_1 + bbX_2 + c \\ X_2 \supseteq X_1 + X_3 + bbX_2 \\ X_3 \supseteq X_1 + X_2 + c \end{cases}$$

Et si on recommence le même remplacement simultané, le système ne change pas et reste avec des ε-transitions. Comme expliqué précédemment, la terminaison repose crucialement sur le fait qu'on élimine *toutes les occurrences* d'une variable en tant que ε-transition **avant** de passer à l'élimination sur une autre variable.

2.2 Élimination des autres transitions par introduction des « $*$ »

Théorème : tout système régulier définit un langage régulier Pour tout système régulier, on peut calculer une expression régulière *équivalente*, c'est-à-dire qui correspond au même langage.

Comme expliqué précédemment, il est préférable de commencer par éliminer les ε-transitions dans le système. On peut donc alors calculer l'expression régulière sur le système sans ε-transition, en raisonnant avec des équations, ce qui permet souvent d'apporter des simplifications au système à la volée. L'algorithme pour construire une telle expression régulière, procède par éliminations successives de variable. Chacune de ces éliminations consiste à :

1. choisir une équation sur une variable X en s'arrangeant pour factoriser le X dans le membre droit (factorisation à droite sur les transitions contenant X) : l'équation de X s'écrit alors $X = AX+B$ avec A et B deux expressions régulières sans aucune occurrence de X (l'expression B pouvant contenir d'autres variables que X dans son vocabulaire) ;
2. on remplace alors $X = AX + B$ par $X = A^*B$ (lemme d'Arden) ;
3. on injecte l'équation $X = A^*B$ dans tous les membres droits de manière à éliminer complètement X des autres équations.

On répète cette élimination jusqu'à ce que l'équation de l'axiome ne contienne plus aucune variable. Comme on élimine une variable à chaque itération, l'algorithme termine en au plus n itérations. En pratique, il est préférable de simplifier le système au fur et à mesure des calculs et choisir l'ordre d'élimination des variables de façon à engendrer une expression régulière aussi simple que possible.

Exercice 3. Sur $V = \{0, 1\}$, on considère le système d'équations :

$$\begin{cases} x_0 = 0x_0 + 1x_1 + \varepsilon \\ x_1 = 0x_2 + 1x_1 \\ x_2 = 0x_0 + 1x_1 \end{cases}$$

Trouver l'expression régulière engendrée par chacune des stratégies :

1. éliminer x_2 puis x_1 ;
2. éliminer x_1 puis x_2 ;
3. remplacer $0x_0 + 1x_1$ par x_2 dans x_0 (ce qui « réduit » le système), puis éliminer x_1, puis x_2.

Quelle est la relation entre ces 3 expressions régulières, trouvées par ses différentes stratégies ?

3 Des expressions régulières aux automates

Comme expliqué en feuille « 03 », pour trouver un système régulier qui est équivalent à une expression régulière donnée, on pourrait procéder en effectuant l'élimination de variables précédente à l'envers : c'est-à-dire en éliminant successivement chaque « * » par l'introduction d'une nouvelle variable et d'une nouvelle inéquation. Cette technique fonctionne très bien quand on fait les calculs « à la main ».

On souhaite cependant ici formaliser un peu plus l'algorithme de calcul. De plus, dans la suite, pour simplifier la plupart traitement des systèmes réguliers, on va définir des transformations qui se limitent à manipuler seulement certaines formes de systèmes réguliers. Une de ces formes est appelée *système d'automate*. Se limiter à cette forme n'est pas restrictif en pratique, puisque tout système régulier pourra être transformé en système d'automate définissant le même langage. En effet, notre algorithme de transformation des expressions régulières va directement produire un tel automate.

Définition des systèmes d'automate Un système régulier est dit « système d'automate » ssi il vérifie :

$$\text{toute alternative vérifie } t_{i,j} \in \{\varepsilon\} \cup (V \cup \{\varepsilon\}).\{X_1, \ldots, X_n\}$$

Un « système d'automate » est sans ε-transition ssi il vérifie :

$$\text{toute alternative vérifie } t_{i,j} \in \{\varepsilon\} \cup V.\{X_1, \ldots, X_n\}$$

NB : dans un automate, la seule alternative d'arrêt possible est « ε ».

Exercice 4. Vérifier que sur un automate, l'algorithme d'élimination des ε-transitions donne bien un automate sans ε-transitions.

Notations Étant donné un système d'automate Δ, on notera $\mathcal{L}(\Delta)$ le langage défini par le système. On notera généralement Q son ensemble de variables et s l'axiome (pour « start symbol »).

En toute généralité, toute équation d'un automate sans ε-transition sur une variable q peut être représentée sous la forme

$$q = r + \Sigma_{i \in I}\, a_i.q_i$$

où :

— r (pour « return ») est soit ε (alternative d'arrêt), soit $\varnothing$ (pas d'alternative d'arrêt dans cette équation) ;
— I est un ensemble **fini** d'indices [1], avec pour tout $i \in I$, $a_i \in V$ et $q_i \in Q$.

Calcul d'un système d'automate dérivé d'une expression régulière Soit V un vocabulaire. On construit ci-dessous une fonction $\mathcal{S}$, qui étant donné une expression régulière $E \in \mathbb{E}[V]$, produit un quadruplet (Q, s, f, Δ) tel que :

1. Q est un ensemble fini de symboles disjoint de V ;
2. s et f sont deux symboles distincts de Q ;
3. Δ est un système d'automate (avec ε-transitions) sur V, ayant Q comme ensemble de variables, d'axiome s, et dont l'inéquation de f est simplement « $f \supseteq \varepsilon$ », qui est l'unique inéquation de Δ à contenir une alternative d'arrêt ;
4. pour tout symbole q absent de $Q \cup V$, le système noté $\Delta[f \supseteq q]$ obtenu en remplaçant l'inéquation de f par $f \supseteq q$, vérifie $\mathcal{L}(\Delta[f \supseteq q]) = \mathcal{L}(E).q$.

La construction de $\mathcal{S}(E)$ est définie par *induction* sur la construction de E, le système Δ y étant simplement représenté comme un *ensemble* d'inéquations :

— soient s et f deux nouveaux symboles distincts entre eux et absents de V, on pose
$\mathcal{S}(\varnothing) \stackrel{def}{=} (\{s, f\}, s, f, \{s \supseteq \varnothing\,,\, f \supseteq \varepsilon\})$ et $\mathcal{S}(\epsilon) \stackrel{def}{=} (\{s, f\}, s, f, \{s \supseteq f, f \supseteq \varepsilon\})$;
— de même, pour tout $a \in V$, on pose
$\mathcal{S}(a) \stackrel{def}{=} (\{s, f\}, s, f, \{s \supseteq a.f\,,\, f \supseteq \varepsilon\})$;
— pour tout $E_1 \in \mathbb{E}[V]$ et $E_2 \in \mathbb{E}[V]$,
soient $(Q_1, s_1, f_1, \Delta_1) \stackrel{def}{=} \mathcal{S}(E_1)$ et soient $(Q_2, s_2, f_2, \Delta_2) \stackrel{def}{=} \mathcal{S}(E_2)$,
quitte à effectuer des renommages dans un des deux quadruplets, on peut supposer sans perte de généralité que $Q_1 \cap Q_2 = \varnothing$, on prend :

$$\mathcal{S}((E_1 \,.\, E_2)) \stackrel{def}{=} (Q_1 \cup Q_2, s_1, f_2, \Delta_1[f_1 \supseteq s_2] \cup \Delta_2)$$

de plus, soit s et f deux nouveaux symboles distincts entre eux et absents de $Q_1 \cup Q_2 \cup V$, on pose :

$$\mathcal{S}((E_1 + E_2)) \stackrel{def}{=} (Q_1 \cup Q_2 \cup \{s, f\}, s, f, \{s \supseteq s_1 + s_2\,,\, f \supseteq \varepsilon\} \cup \Delta_1[f_1 \supseteq f] \cup \Delta_2[f_2 \supseteq f])$$

— pour tout $E_0 \in \mathbb{E}[V]$, soit $(Q_0, s_0, f_0, \Delta_0) \stackrel{def}{=} \mathcal{S}(E)$,
soient s et f deux nouveaux symboles distincts entre eux et absents de Q_0,

$$\mathcal{S}(E_0{}^*) \stackrel{def}{=} (Q \cup \{s, f\}, s, f, \{s \supseteq s_0 + f\,,\, f \supseteq \varepsilon\} \cup \Delta_0[f_0 \supseteq s])$$

On vérifie ensuite que la construction de $\mathcal{S}$ préserve bien les 4 propriétés énoncées (cf. exo 7). Finalement, étant donné E une expression régulière quelconque, soit (Q, s, f, Δ) le système d'automate

1. Typiquement, I sera un ensemble fini d'entiers - mais formellement, il est commode que I puisse être un ensemble fini quelconque (cf. feuille « 05 »).

retourné par $\mathcal{S}(E)$ et soit $q \notin Q \cup V$. Soit $\Delta' \stackrel{def}{=} \Delta[f \supseteq q] \cup \{q \supseteq \varepsilon\}$. On en déduit (en utilisant la propriété P2 de l'exo 7) que $\mathcal{L}(\Delta') = \mathcal{L}(E)$. Par ailleurs, en éliminant de q de Δ', on retombe exactement sur Δ, on en déduit donc $\mathcal{L}(\Delta) = \mathcal{L}(E)$. L'automate produit par $\mathcal{S}(E)$ est donc bien correct.

Exercice 5. Soit $V = \{a, b\}$ appliquer cet algorithme sur l'expression $a^*b + b^*a$. On évitera de faire des renommages en choisissant directement des noms 2 à 2 distincts. On notera ainsi les noms de variables successives introduites pour s et f : $s_0, f_0, s_1, f_1, \ldots$ jusqu'au s et f finaux. Terminer en simplifiant l'automate obtenu.

Exercice 6. Idem avec $(a^*b^*)^*$. En déduire une forme plus simple de cette expression régulière.

Exercice 7. (avancé) On admet les deux propriétés suivantes :

P1 Étant donné deux systèmes réguliers d'inéquations Δ et Δ' sur un même vocabulaire V et qui vérifient, en tant qu'ensemble d'inéquations, $\Delta \subseteq \Delta'$. Soit Q l'ensemble des variables de Δ. Alors les plus petites solutions de Δ et Δ' coïncident sur Q.

P2 Soit V un vocabulaire et $q \notin V$. Soit Δ un système régulier d'inéquations sur le vocabulaire $V \cup \{q\}$ et sur un ensemble de variables Q. Soit un langage L_1 sur V. Si $\mathcal{L}(\Delta) = L_1.q$, alors pour tout vocabulaire $V' \supseteq V$ tel que $V' \cap Q = \varnothing$ et toute expression régulière E_2 sur V', on a $\mathcal{L}(\Delta \cup \{q \supseteq E_2\}) = L_1.\mathcal{L}(E_2)$ (si on garde pour $\Delta \cup \{q \supseteq E_2\}$ l'axiome de Δ).

Prouver que la construction de $\mathcal{S}$ préserve bien les 4 propriétés énoncées dans sa définition.

4 Intersection de systèmes d'automates (sans ε-transition)

On a vu des algorithmes pour transformer une expression régulière en système d'automate sans ε-transition et réciproquement. Cela permet finalement de *réduire* le problème du calcul de l'intersection de 2 expressions régulières à celui du calcul de l'intersection de 2 systèmes d'automate sans ε-transition.

Notion de réduction (à savoir) *En science*, « **réduire** » un « problème » P à un « problème » Q, veut dire « on peut résoudre P en résolvant Q ».

Autrement dit, cela signifie qu'on peut ramener toute *instance* de P, e.g. « calculer l'intersection de deux expressions régulières E_1 et E_2 **données** », à une *instance* de Q, e.g. « calculer l'intersection des deux systèmes d'automates sans ε-transition Δ_1 et Δ_2 obtenus à partir de E_1 et E_2 » de telle façon que la réponse à cette instance de Q (e.g. le système Δ obtenu par le calcul d'intersection de Δ_1 et Δ_2) permet de donner la réponse à l'instance de P (e.g. avec l'expression régulière obtenue à partir de Δ).

En *informatique*, la transformation de l'instance de P en instance de Q et celle de la réponse sur Q à la réponse sur P doivent de plus *être calculables par un algorithme* (parfois en satisfaisant une condition de « coût algorithmique » supplémentaire).

Définir et utiliser des réductions est au cœur de sciences en général, et de l'informatique en particulier. En programmation, les appels de procédure (définie dans une librairie) peuvent généralement être vues comme des cas particuliers de réduction (on utilise une procédure prévue pour résoudre Q afin de résoudre P).

Calcul de l'intersection de systèmes d'automate sans ε-transition Soient Δ et Δ' deux systèmes d'automates sans ε-transition dont les ensembles de variables sont notés respectivement Q et Q' avec comme axiomes respectifs s et s'. On construit ci-dessous un système d'automate Δ_0 sans ε-transition tel que $\mathcal{L}(\Delta_0) = \mathcal{L}(\Delta) \cap \mathcal{L}(\Delta')$. Cette construction est basée sur les trois principes suivants :

1. L'ensemble de variables de Δ_0 est $Q \times Q'$. En effet, chaque variable de Δ_0 est notée « (q, q') » pour $q \in Q$ et $q' \in Q'$ et représente le langage $q \cap q'$ (on assimile ici la variable q avec le langage qu'il représente).
2. L'axiome de Δ_0 est (s, s').
3. En supposant que l'équation de q dans Δ est donnée par $q = r + \Sigma_{i \in I} a_i.q_i$ et celle de q' dans Δ' par $q' = r' + \Sigma_{j \in I'} a_j.q'_j$, l'équation de chaque (q, q') est donnée par :

$$(q, q') = (r \cap r') + \Sigma_{(i,j) \in I \times I' \mid a_i = a'_j} a_i.(q_i, q'_j)$$

où $r \cap r'$ correspond à l'intersection usuelle dans $\mathcal{P}(\{\varepsilon\})$.

NB Il est bien aussi de renommer les variables sur le système obtenu pour avoir des noms plus courts que (q, q').

Exercice 8. Justifier que l'algorithme est correct (en utilisant les propriétés de $\cap$ données en feuille « 03 »).

Exercice 9. Sur le vocabulaire $\{a, b\}$, calculer l'intersection des deux systèmes d'automates Δ et Δ' suivants :

$$\Delta \stackrel{def}{=} \begin{cases} s = a.s + a.q_1 + b.s + b.q_2 \\ q_1 = b.f \\ q_2 = a.f \\ f = \varepsilon \end{cases} \qquad \Delta' \stackrel{def}{=} \begin{cases} s' = \varepsilon + a.q' + b.s' \\ q' = a.s' + b.q' \end{cases}$$

En déduire une expression régulière pour $(a+b)^*(ab + ba) \cap (ab^*a + b)^*$.

5 Complémentaire d'automates déterministes et complets

Les transformations vues jusqu'ici sur les systèmes d'automate ont permis de donner un algorithme pour l'intersection d'expressions régulières. Peut-on réutiliser le formalisme des systèmes d'automate pour le calcul du complémentaire d'une expression régulière ?

Soit L un langage sur un vocabulaire V. Par définition, $\overline{L} = \{w \in V^* \mid w \notin L\}$. Une première étape pour espérer pouvoir calculer le complémentaire serait d'être capable d'avoir une fonction calculable, appelée `acc` (pour « accepte »), qui prend en entrée un système d'automate Δ et un mot de $w \in V^*$ et retourne un booléen indiquant si $w \in \mathcal{L}(\Delta)$.

Soit Δ un automate sans ε-transition sur un vocabulaire V et un ensemble de variables Q. Pour $q \in Q$, on note $\mathcal{L}_\Delta(q)$ le langage associé à q dans la solution de Δ. L'équation de q étant de la forme « $q = r + \Sigma_{i \in I} a_i.q_i$ » avec $r \in \{\varepsilon, \varnothing\}$, $a_i \in V$ et $q_i \in Q$, on en déduit :

— $\varepsilon \in \mathcal{L}_\Delta(q)$ ssi $r = \varepsilon$;
— pour $a \in V$ et $w \in V^*$, $a.w \in \mathcal{L}_\Delta(q)$ ssi $\exists\, i \in I, a = a_i \wedge w \in \mathcal{L}_\Delta(q_i)$

Autrement dit, on voit émerger ici un calcul de « $w \in L_\Delta(q)$ » par récurrence sur w et paramétré par q. Pour simplifier les calculs, on étudie déjà le cas où pour tout a, il existe un unique $i \in I$ avec $a = a_i$. C'est le cas où chaque équation dans Δ de $q \in Q$ s'écrit sous la forme :

$$q = r_q + \Sigma_{a \in V} a.q_{(q,a)} \qquad \text{où } r_q \in \{\varepsilon, \varnothing\} \text{ et pour tout } a \in V,\ q_{(q,a)} \in Q$$

Δ est alors ce qu'on appelle un système **d'automate déterministe et complet**. Un exemple de tel système est donné par le système suivant qui calcule $(ab^*a + b)^*$:

$$\Delta_{\text{pair}} \stackrel{def}{=} \begin{cases} s = \varepsilon + a.q + b.s \\ q = a.s + b.q \end{cases}$$

Pour définir notre fonction `acc` plus facilement, il est commode d'introduire une représentation plus adaptée.

Automates déterministes complets On appelle **automate déterministe et complet** un 5-uplet A de la forme $\langle Q, V, s, \delta, F\rangle$ où Q est l'ensemble de variables (aussi appelées états), V est le vocabulaire, s la variable initiale (ou *état initial*), $\delta \in Q \times V \to Q$ est la *fonction de transition*, et $F \subseteq Q$ est l'ensemble des variables avec une alternative d'arrêt (on parle d'ensemble d'*états finaux*).

Autrement dit, pour toute variable $q \in Q$, l'équation de q s'écrit dans la forme voulue pour Δ ci-dessus avec :

— $r_q \stackrel{def}{=}$ si $q \in F$ alors ε sinon $\varnothing$;

— $q_{(q,a)} \stackrel{def}{=} \delta(q, a)$

On note $\mathcal{L}_A(q)$ le langage associé à q dans la solution du système d'équations. Par convention le langage reconnu/engendré par l'automate, noté $\mathcal{L}(A)$, vaut $\mathcal{L}_A(s)$.

Soit A un automate déterministe et complet. On a donc

— $\varepsilon \in \mathcal{L}_A(q)$ ssi $\varepsilon \in F$;

— pour $a \in V$ et $w \in V^*$, $a.w \in \mathcal{L}_A(q)$ ssi $w \in \mathcal{L}_A(\delta(q, a))$

Cela permet de définir la fonction $\mathrm{acc}_A : Q \times V^* \to \{f, v\}$ telle que $\mathrm{acc}_A(q, w) = v$ ssi $w \in \mathcal{L}_A(q)$. En effet, la définition de $\mathrm{acc}_A(q, w)$ procède par récurrence sur w (pour tout q) :

— pour tout $q \in Q$, $\mathrm{acc}_A(q, \varepsilon) \stackrel{def}{=}$ si $q \in F$ alors v sinon f ;

— pour $a \in V$ et $w \in V^*$, pour tout $q \in Q$, $\mathrm{acc}_A(q, a.w) \stackrel{def}{=} \mathrm{acc}_A(\delta(q', a), w)$

Vérifier « $\mathrm{acc}_A(q, w) = v$ ssi $w \in \mathcal{L}_A(q)$ » est alors trivial par récurrence sur w.

Exercice 10. On programme ici « $\mathrm{acc}_A(s, w)$ » en PYTHON. On représente les symboles de Q et de V comme des chaînes de caractère. L'automate déterministe et complet est 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 tel que `delta[q][a]` représente $\delta(q, a)$: ce codage repose sur le fait que $(Q \times V) \to Q \simeq Q \to (V \to Q)$ (cf. compléments de la feuille « 02 » sur la relation $\simeq$ entre ensembles).

Par exemple, l'automate correspondant au système Δ_{pair} ci-dessus peut être représenté par :

```
pair = ({'s', 'q'},
        {'a', 'b'},
        's',
        { 's': { 'a': 'q', 'b': 's' },
          'q': { 'a': 's', 'b': 'q' }  },
        { 's' })
```

Pour `A` est un tel 5-uplet et `(s, delta, F)` et `w` une liste formée uniquement de symboles du vocabulaire, définir une fonction `accepte(A, w)` qui retourne un booléen correspondant à $\mathrm{acc}_A(s, w) = v$.

```
def accepte(A, w):
```

Définir ensuite une fonction `complement(A)` telle que `accepte(complement(A),w)` retourne systématiquement le même booléen que « **not** `accepte(A,w)` ». Écrire aussi une fonction `display_system(A)` qui affiche le système d'équations de l'automate.

Théorème du complémentaire d'un automate déterministe complet

Soit $A = \langle Q, V, s, \delta, F\rangle$ un automate déterministe et complet. On pose $\overline{A} \stackrel{def}{=} \langle Q, V, s, \delta, \overline{F}\rangle$.
On a : $\mathcal{L}(\overline{A}) = \overline{\mathcal{L}(A)}$.

Exercice 11. Montrer que pour tout $q \in Q$, pour $a \in V$ et $w \in V^*$,

$$\mathrm{acc}_{\overline{A}}(q, w) = v \Leftrightarrow \mathrm{acc}_A(q, w) = f$$

En déduire une preuve du théorème.

Exercice 12. Donner $\overline{\Delta_{\text{pair}}}$ le système du complémentaire de l'automate sous-jacent à Δ_{pair}. Calculer l'expression régulière associée et vérifier que c'est bien le résultat attendu.

6 Déterminisation de systèmes d'automate (sans ε-transition)

Pour trouver un système d'automate déterministe et complet équivalent à un langage donné, il faut s'arranger pour que dans chaque équation vérifie :

1. il n'y a pas deux transitions avec un symbole de V qui soit un *préfixe commun* ;
2. tout symbole de V apparaît dans une transition de l'équation.

Si on voit le calcul de acc comme une suite de *choix* de transitions « $\delta(q, a)$ » en fonction du dernier symbole a lu et de l'état courant q, la première condition garantit le caractère *déterministe* du choix (au plus un choix possible), alors que le second garantit le caractère *complet* (au moins un choix possible). C'est l'explication de la terminologie « automate déterministe et complet ».

Exercice 13. La représentation Python des automates à l'exo 10 est plus générale que la représentation mathématique : les automates représentés en Python sont déterministes mais ils ne sont pas forcément complets. Par exemple, on pourrait y représenter l'automate suivant qui reconnaît a^*b^+ avec :

```
ast_bst = ({'s', 'q'},
           {'a', 'b'},
           's',
           { 's': { 'a': 's', 'b': 'q' },
             'q': { 'b': 'q' }  },
           { 'q' })
```

En effet, ça peut être vu comme un codage du système d'automate :

$$\begin{cases} s = a.s + b.q \\ q = \varepsilon + b.q \end{cases}$$

La fonction `display_equation(A)` du corrigé donne bien le bon système d'équation sur un tel automate incomplet [2], mais la fonction `accepte(A,w)` n'a pas le comportement attendu (elle lève parfois une exception au lieu de retourner `False` comme attendu). Modifier la fonction `accepte` du corrigé pour qu'elle fonctionne aussi sur les automates incomplets.

Pour *transformer* le système d'automate de façon à garantir le caractère déterministe (1) et complet (2), on applique les deux stratégies ci-dessous (chacune établissant la propriété de numéro correspondant) :

1. *factoriser à gauche les préfixes communs*, quitte à introduire de nouvelles variables avec leurs équations pour représenter les suffixes obtenus par ces factorisations. Par construction, un tel suffixe correspond à une somme de variables. Autrement dit, on applique la même technique de *factoriser à gauche* que pour le calcul d'intersection, sauf qu'au lieu d'appliquer les règles de factorisation du « . » sur « ∩ », on applique celles de « . » sur « + » (plus simples !).
2. *compléter* les équations : en ajouter des transitions « $a.p$ » où p est une variable associée à l'équation $p = \Sigma_{a \in V}\, a.p$, ce qui préserve en fait le langage. En effet, le langage de p est alors $\varnothing$ (par Arden). L'équation de p vérifie bien elle-même les conditions (1) et (2). On dit que p est un *état puits* : il suffit d'une seule variable/état puits pour tout le système d'équations.

2. Implicitement, on suppose quand même que pour tout `q in Q`, on ait `q in delta` : le caractère incomplet ne peut venir que du fait qu'il y a des `a in V` tels que `not (a in delta[q])`.

Exercice 14. Quel est le système d'équations correspondant à l'automate retourné par `complement(ast_bst)`, pour `ast_bst` l'automate défini à l'exo 13 et `complement` la fonction du corrigé de l'exo 10 ? Quel est le langage de ce système d'équations ? Calculer le complémentaire du langage de `ast_bst` et montrer que ce n'est pas celui de `complement(ast_bst)`.

Exercice 15. On *suppose déjà écrite* une fonction `new_symbol(Q, prefix)` (donnée dans le corrigé) qui étant donné un ensemble de chaînes `Q` et une chaîne `prefix`, retourne une chaîne absente de `Q` préfixée par `prefix`. Écrire une fonction `complete(A)` qui modifie `A` de manière à le rendre complet en appliquant la stratégie 2 ci-dessus (ce qui permettra d'insérer un appel à `complete` à la première ligne de `complement` pour garantir que le calcul de complémentaire est correct, cf. l'exo 14).

Méthode (informelle) pour la déterminisation de système d'automates Le principe générale est :

1. Se ramener à un système avec une unique solution (e.g. éliminations des ε-transitions).
2. Éliminer les variables/équations inutiles (variables inutilisées depuis l'axiome ou qui admettent $\varnothing$ comme solution).
3. En commençant de l'axiome, *factoriser à gauche* chaque équation quitte à introduire de nouvelles variables avec leurs équations – dont on élimine aussitôt les ε-transitions (cf stratégie 1 ci-dessus) ; au passage, il est bien de « simplifier » le système : élimination des variables/équations inutiles, factorisation des variables équivalentes (e.g. même membre droit), etc ; par construction, chaque nouvelle variable correspond à une somme des variables d'origine : on évitera d'introduire deux variables pour la même somme (réutiliser la variable existante plutôt que d'en introduire une autre) ; réappliquer itérativement la factorisation sur les nouvelles équations jusqu'à ce que toutes les équations du systèmes soient déterministes.
4. Quand le système est déterministe, vérifier s'il est complet, sinon ajouter l'état puits et *compléter* les équations (cf stratégie 2 ci-dessus).

On formalisera davantage cette méthode à la feuille « 05 », mais il faut d'abord se l'approprier sur des exemples...

Exercice 16. Soit $V = \{a, b\}$. Pour chacune des expressions régulières E suivantes sur V, donner un automate déterministe et complet équivalent, en dérivant le système d'équations « à la main » avec la méthode donnée ci-dessus. En déduire ensuite une expression régulière, notée $\overline{E}$, dont le langage est $\overline{\mathcal{L}(E)}$ (on essaiera d'avoir une expression régulière aussi « simple » que possible).

1. $E_1 \stackrel{def}{=} ba + bba$
   indication d'une expression possible pour le complémentaire : $a^? + bb^?b^? + (b^?b^?b^?a + bbb).(a+b)^+$;
2. $E_2 \stackrel{def}{=} ab^* + aa$;
3. $E_3 \stackrel{def}{=} (a+b)^*.a$

Exercice 17. Soit $V = \{a, b, c\}$. Pour chacun des systèmes d'automates suivants (d'axiome q_1), donner un automate déterministe et complet équivalent. On ne demande pas de calculer d'expression régulière (ouf !), mais de « simplifier » le système autant que possible (en gardant le système déterministe et complet).

$$\Delta_1 \stackrel{def}{=} \begin{cases} q_1 \supseteq q_2 + q_4 \\ q_2 \supseteq a.q_2 + b.q_3 \\ q_3 \supseteq \varepsilon \\ q_4 \supseteq a.q_5 \\ q_5 \supseteq b.q_5 + \varepsilon \end{cases} \qquad \Delta_2 \stackrel{def}{=} \begin{cases} q_1 \supseteq a.q_1 + a.q_2 + c.q_3 \\ q_2 \supseteq b.q_1 + q_3 \\ q_3 \supseteq b.q_3 + b.q_4 + \varepsilon \\ q_4 \supseteq b.q_4 + \varepsilon \end{cases}$$

$$\Delta_3 \stackrel{def}{=} \begin{cases} q_1 \supseteq a.q_1 + b.q_1 + a.q_2 \\ q_2 \supseteq b.q_3 \\ q_3 \supseteq a.q_3 + b.q_3 + a.q_4 \\ q_4 \supseteq b.q_5 \\ q_5 \supseteq a.q_5 + b.q_5 + \varepsilon \end{cases} \qquad \Delta_4 \stackrel{def}{=} \begin{cases} q_1 \supseteq q_2 + a.q_3 + q_5 \\ q_2 \supseteq a.q_1 + q_3 \\ q_3 \supseteq q_1 + b.q_2 + a.q_4 \\ q_4 \supseteq c.q_4 + \varepsilon \\ q_5 \supseteq a.q_5 + b.q_5 \end{cases}$$

Exercice 18. La méthode décrite ci-dessus permet-elle toujours de trouver un automate déterministe et complet équivalent à l'automate de départ ? En particulier à quelle condition cette méthode termine-t-elle ? Justifier informellement.