A6 – Clôture commutative de langages réguliers

On fixe l'alphabet $\Sigma = \{a, b\}$, et on note Σ^* l'ensemble des mots sur Σ. Pour $w \in \Sigma^*$, on note respectivement $|w|_a$ et $|w|_b$ le nombre d'occurrences de a et de b dans w. Un *langage* sur Σ^* est un sous-ensemble non nécessairement fini de Σ^*. La *clôture commutative* d'un langage L sur l'alphabet Σ est le langage $\mathrm{CCl}(L)$ des mots $w \in \Sigma^*$ tels qu'il existe $w' \in L$ avec $|w|_a = |w'|_a$ et $|w|_b = |w'|_b$.

On admettra le théorème de Kleene : un langage peut être décrit par une expression rationnelle si et seulement s'il est reconnu par un automate fini. De tels langages sont appelés *réguliers*. L'objet du problème est d'étudier la clôture commutative de langages réguliers.

Question 0. Calculer la clôture commutative du langage régulier L_0 défini par l'expression rationnelle $a^* + b^*$.

Question 1. Soit L_1 le langage régulier défini par l'expression rationnelle $b(aa)^*$. Proposer une expression rationnelle pour $\mathrm{CCl}(L_1)$. Existe-t-il un langage régulier L_1' tel que $\mathrm{CCl}(L_1') = L_1$?

Question 2. Exhiber un langage régulier L_2 tel que $\mathrm{CCl}(L_2)$ soit le langage $L_2' := \{w \in \Sigma^* \mid |w|_a = |w|_b\}$. En déduire que la clôture commutative d'un langage régulier n'est pas nécessairement un langage régulier.

Question 3. On dit qu'un langage L est *a-borné* s'il existe $k \in \mathbb{N}$ tel que, pour tout $w \in L$, on a $|w|_a \leq k$. Montrer que, pour tout langage régulier L, si L est a-borné, alors $\mathrm{CCl}(L)$ est régulier.

Étendre l'argument aux langages réguliers *ab-bornés*, c'est-à-dire les langages réguliers L où il existe $k \in \mathbb{N}$ tel que, pour tout $w \in L$, on a soit $|w|_a \leq k$, soit $|w|_b \leq k$.

Question 4. Proposer un algorithme qui décide, étant donné un langage régulier L sur Σ, si L est a-borné. Proposer également un algorithme qui décide si L est ab-borné.

Suite des questions

Question 5. Exhiber un langage régulier L qui n'est pas ab-borné mais où $\mathrm{CCl}(L)$ est régulier. Qu'en déduire par rapport à la question 3 ?

Question 6. On dit qu'un langage L est *ultimement périodique* s'il existe $t \in \mathbb{N}$ et $p \in \mathbb{N}^*$ tel que, pour tous $k \geq t$ et $l \geq t$, s'il existe un mot $w \in L$ avec $(|w|_a, |w|_b) = (k, l)$, alors il existe un mot $w' \in L$ avec $(|w'|_a, |w'|_b) = (k + p, l)$, et un mot $w'' \in L$ avec $(|w''|_a, |w''|_b) = (k, l + p)$.

Montrer que, pour tout langage régulier L, si L est ultimement périodique, alors $\mathrm{CCl}(L)$ est régulier. Comparer au résultat établi à la question 3.

Question 7. Montrer que, pour tout langage régulier L, si $\mathrm{CCl}(L)$ est régulier, alors L est ultimement périodique. Conclure.

Indication : On pourra utiliser la notion de *quotient à gauche*. Le *quotient à gauche* d'un langage L par un mot $w \in \Sigma^*$, noté $w^{-1}L$, est l'ensemble $\{w' \in \Sigma^* \mid ww' \in L\}$. On pourra utiliser, puis démontrer, l'affirmation suivante : pour tout langage régulier L', l'ensemble de langages $\{w^{-1}L' \mid w \in \Sigma^*\}$ est fini.

Corrigé

Question 0. La clôture commutative est $a^* + b^*$, c'est-à-dire le langage L_0 lui-même. Ceci est normal : le langage proposé est *commutatif*, c'est-à-dire que si un mot appartient au langage alors toutes ses permutations (ici, le mot lui-même) appartiennent au langage.

Question 1. Le langage $\mathrm{CCl}(L_1)$ est, intuitivement, le langage des mots comportant exactement un b, et un nombre pair de a. Une expression régulière pour ce langage est $(aa)^*(aba + b)(aa)^*$.

Le langage L_1 n'est pas commutatif, car on a $baa \in L_1$ mais $aab \notin L_1$. Comme la clôture commutative d'un langage est toujours un langage commutatif, il n'existe pas de langage L_1' tel que $\mathrm{CCl}(L_1') = L_1$.

Question 2. Notons L_2' le langage proposé. On considère le langage régulier $L_2 := (ab)^*$, dont il est clair que la clôture commutative est L_2'.

On peut aisément montrer à l'aide du lemme de pompage que L_2' n'est pas rationnel : si on prend $k \in \mathbb{N}$ la longueur de pompage, le mot $a^k b^k$ est dans L_2', mais le pompage conduirait à affirmer que $a^{k+d}b^k$ est dans L_2' pour un certain $d \in \mathbb{N}^*$, ce qui serait absurde.

Question 3. Pour tout langage régulier L, on appelle *a-projection* de L le langage $L_{\downarrow a} := \{a^i \mid \exists w \in L, |w|_a = i\}$. On observe d'abord que, pour tout langage régulier L, la a-projection de L est un langage régulier. En effet, si l'on considère une expression régulière pour L et que l'on remplace chaque occurrence de b par ϵ, il est clair que l'expression régulière obtenue décrit $L_{\downarrow a}$. (En d'autres termes, $L_{\downarrow a}$ est l'image de L par le morphisme $h : \Sigma \to \Sigma^*$ défini par $h(a) := a$ et $h(b) := \epsilon$, et les langages réguliers sont clos par morphisme, c'est-à-dire que l'image d'un langage régulier par un morphisme est toujours un langage régulier.)

Pour tout langage régulier L, on dit que L est un *langage sur a* si, pour tout $w \in L$, on a $|w|_b = 0$. Pour un langage L sur a, et pour $i \in \mathbb{N}$, on appelle *(b, i)-complétion* de L le langage $L_{\uparrow b,i} := \{w \in \Sigma^* \mid a^{|w|_a} \in L \wedge |w|_b = i\}$. On observe également que, pour tout langage régulier L sur a, pour tout $i \in \mathbb{N}$, le langage $L_{\uparrow b,i}$ est un langage régulier. (Le résultat est également vrai sans l'hypothèse que L est un langage sur a, mais celle-ci permet de simplifier légèrement la définition de la complétion.) En effet, si l'on considère un automate déterministe A qui reconnaît L, on peut supposer, quitte à retirer les états inutiles (non accessibles ou non coaccessibles), que A n'a pas de transitions étiquetées b. On peut ensuite construire un automate $A_{\uparrow b,i}$ qui reconnaît $L_{\uparrow b,i}$ en créant $i + 1$ copies de l'automate A étiquetées de 0 à i, en prenant comme état initial l'état initial dans la copie 0, en prenant comme états

finaux les états finaux dans la copie i, et en ajoutant, en plus des transitions originales dans chaque copie, une transition étiquetée b qui va de chaque état dans une copie $0 \leq j < n$ vers le même état dans la copie $j+1$. Il est facile de constater que $A_{\uparrow b,i}$ reconnaît bien $L_{\uparrow b,i}$, qui est donc régulier.

Prouvons à présent le résultat demandé. Soit L un langage régulier a-borné, et soit $k \in \mathbb{N}$ la borne. Comme L est a-borné, on peut l'écrire comme $\bigcup_{0 \leq i \leq k} L_i$, où $L_i := \{w \in L \mid |w|_b = i\}$. On a alors clairement $\mathrm{CCl}(L) = \bigcup_{0 \leq i \leq k} \mathrm{CCl}(L_i)$, et comme les langages réguliers sont clos par union, il suffit de montrer que, pour chaque $0 \leq i \leq k$, le langage $\mathrm{CCl}(L_i)$ est régulier.

Ainsi, fixons $0 \leq i \leq k$, et notons $L_{i\downarrow a}$ la a-projection de L_i. Comme la clôture commutative est commutative, il est facile de voir que $\mathrm{CCl}(L_i) = (\mathrm{CCl}(L_{i\downarrow a}))_{\uparrow b,i}$. Or, comme L_i est régulier, nous avons montré initialement que $L_{i\downarrow a}$ est également régulier, et c'est un langage sur a, donc commutatif. Ainsi, on a $\mathrm{CCl}(L_{i\downarrow a}) = L_{i\downarrow a}$, qui est donc un langage régulier sur a. Ainsi, $\mathrm{CCl}(L_i)$ est également régulier d'après ce que nous avons montré initialement, ce qui conclut la preuve.

Pour les langages ab-bornés, si l'on prend un tel langage L et qu'on pose k la borne, on peut clairement écrire $L = \bigcup_{0 \leq i \leq k} L_{a,i} \cup \bigcup_{0 \leq i \leq k} L_{b,i}$, où $L_{c,i} := \{w \in L \mid |w|_c = i\}$ pour $c \in \{a, b\}$. Ainsi, il suffit de montrer que, pour $c \in \{a, b\}$ et $0 \leq i \leq k$, le langage $L_{c,i}$ est régulier, ce que l'on peut faire comme précédemment.

Question 4. La question ne spécifie pas quelle représentation du langage L est fournie en entrée. On supposera que le langage L nous est fourni sous la forme d'un automate fini ; si le langage L était fourni sous la forme d'une expression rationnelle, on pourrait utiliser les méthodes vues en cours pour obtenir à partir de l'expression rationnelle un automate fini reconnaissant le même langage.

Étant donné l'automate A, on supprime d'abord les états inutiles, ce qui peut être fait en temps linéaire. À présent, il est clair que le langage $L(A)$ reconnu par A n'est *pas* a-borné si et seulement s'il existe un cycle dans A comportant une transition étiquetée a. En effet, s'il existe un tel cycle, $L(A)$ comprend des mots avec un nombre arbitraire de a, que l'on peut obtenir en parcourant le cycle un nombre arbitraire de fois. À l'inverse, s'il n'existe pas de tel cycle, le nombre de a dans un mot de $L(A)$ est borné par le nombre total de transitions étiquetées par a dans A, puisque chaque transition ne peut être franchie qu'une seule fois dans un chemin acceptant. Ainsi, il suffit de déterminer si A contient un tel cycle.

Un algorithme naïf pour cette tâche, en temps quadratique, consiste à considérer toutes les transitions étiquetées a allant d'un état q à un état q', et, pour chaque transition, déterminer s'il existe un chemin dans A allant de l'état q' à l'état q : si oui, on a trouvé un cycle, et à l'inverse, s'il y a un cycle comportant une transition étiquetée a, il est clair que le test effectué pour cette transition va réussir. Cet algorithme est en temps quadratique. On peut également résoudre le problème en temps linéaire avec un algorithme qui considère le sous-graphe induit par les transitions étiquetées b, construit en temps linéaire le graphe acyclique orienté des composantes connexes, y ajoute les arêtes induites par les transitions étiquetées a (y compris les boucles allant d'une composante à elle-même), et teste en temps linéaire si le résultat est acyclique. Si c'est le cas, alors il ne peut y avoir de cycle impliquant une transition étiquetée a, puisqu'un tel cycle impliquerait l'existence d'un cycle dans le graphe résultat (quitte à rassembler les parties du chemin qui parcourent des arêtes étiquetées b et restent dans la même composante connexe du graphe de ces transitions). À l'inverse, tout cycle dans ce graphe donne lieu à un cycle dans le graphe original, en ajoutant des chemins au sein des composantes connexes du graphe des transitions étiquetées b.

Pour le problème de décider si un langage L est ab-borné, en posant A un automate tel que $L(A) = L$, il est facile d'observer que L n'est *pas* ab-borné si et seulement si A contient un cycle C_a incluant une transition a, un cycle C_b incluant une transition b, et l'un de C_a et C_b est accessible à partir de l'autre (ce qui est en particulier le cas s'ils sont égaux). En effet, il est clair que remplir cette condition suffit à ne pas être ab-borné. À l'inverse, si L n'est pas ab-borné, pour tout $k' \in \mathbb{N}$, il faut que L contienne un mot $w_{k'}$ qui s'écrit soit $w^a_{k'} w^b_{k'}$, où $w^a_{k'}$ contient au moins k' a et $w^b_{k'}$ contient au moins k' b, soit $w^b_{k'} w^a_{k'}$ avec les mêmes conditions. Pour k' suffisamment grand, le pompage dans un tel mot permet d'extraire

un cycle contenant une transition étiquetée par a et un cycle contenant une transition étiquetée par b, où l'un des cycles est accessible à partir de l'autre (suivant le cas).

Sans chercher cette fois à optimiser le degré du polynôme, un algorithme efficace pour cette tâche consiste à tester d'abord s'il y a un cycle contenant une transition a et une transition b, en essayant toutes les paires de telles transitions possibles, et en vérifiant l'accessibilité de l'extrémité de chaque transition depuis l'extrémité opposée de l'autre transition (via des transitions d'étiquette quelconque). À défaut, on teste s'il y a deux cycles différents, l'un contenant une transition a et l'autre une transition b, et l'un accessible à partir de l'autre, en essayant toutes les paires de telles transitions, en vérifiant l'accessibilité de l'extrémité de chaque transition depuis son autre extrémité (via des transitions d'étiquette quelconque), puis en vérifiant l'accessibilité d'une extrémité quelconque d'une des deux transitions depuis une extrémité quelconque de l'autre transition (via des transitions d'étiquette quelconque).

Question 5. Le langage régulier $(a+b)^*$ de tous les mots sur Σ^* est commutatif, donc égal à sa propre clôture commutative, qui est donc régulière ; pourtant $(a+b)^*$ n'est clairement pas ab-borné.

Si on veut un exemple moins trivial, on peut prendre $(aa+bb)^*$, ou encore $(aa)^*(bb)^*$. En tout cas, par rapport à la question 3, être ab-borné n'est manifestement pas une condition nécessaire pour avoir une clôture commutative régulière.

Question 6. Pour $i, j \in \mathbb{N}$, on appelle $S_{i,j}$ le sous-ensemble de $\{0, \dots, p-1\}^2$ défini par $(k,l) \in S_{i,j}$ s'il existe $w \in L$ tel que $|w|_a = t + ip + k$ et $|w|_b = t + jp + l$. Comme L est ultimement périodique, il est clair que $S_{i,j} \subseteq S_{i+1,j+1}$ pour tout $i \in \mathbb{N}$. Ainsi, la suite $S_{i,j}$ d'ensembles est croissante au sens de l'inclusion, et elle est bornée par $\{0, \dots, p-1\}^2$, donc il existe un rang r à partir duquel elle est constante, c'est-à-dire $S_{r',r'} = S_{r,r}$ pour tout $r' \geq r$. Par définition de l'ultime périodicité, il est clair que ceci implique en fait que $S_{r',r''} = S_{r,r}$ pour tous $r', r'' \geq r$, car si on avait $S_{r,r} \subsetneq S_{r',r''}$ pour $r', r'' \geq r$, on en déduirait par ultime périodicité que $S_{r,r} \subsetneq S_{\max\{r',r''\},\max\{r',r''\}}$, une contradiction. On appelle $t' := t + rp$.

Notons d'abord qu'on peut écrire $L = L_{<t'} \cup L_{\geq t'}$, où $L_{<t'}$ est $\{w \in L \mid |w|_a < t' \vee |w|_b < t'\}$ et $L_{\geq t'}$ est $\{w \in L \mid |w|_a \geq t' \wedge |w|_b \geq t'\}$. Il est clair que $L_{<t'}$ et $L_{\geq t'}$ sont réguliers : en effet, on peut les écrire comme l'intersection de L avec des langages réguliers. Du reste, $L_{<t'}$ est clairement ab-borné, donc $\mathrm{CCl}(L_{<t'})$ est régulier d'après la question 3. Ainsi, comme les langages réguliers sont clos par union, il suffit de montrer que $\mathrm{CCl}(L_{\geq t'})$ est régulier. Écrivons $L_{\geq t'}$ comme l'union des $L_{k,l}$ pour $(k,l) \in \{0, \dots, p-1\}^2$, chacun étant défini comme l'intersection de $L_{\geq t'}$ avec le langage régulier $\{w \in \Sigma^* \mid (|w|_a - t') \bmod p = k \wedge (|w|_b - t') \bmod p = l\}$. Il suffit de montrer que, pour tout $(k,l) \in \{0, \dots, p-1\}^2$, le langage $\mathrm{CCl}(L_{k,l})$ est régulier. Fixons $(k,l) \in \{0, \dots, p-1\}^2$. Si le langage $L_{k,l}$ est vide, alors $\mathrm{CCl}(L_{k,l})$ est clairement régulier. Sinon, s'il est non-vide, par définition de t', ceci permet de savoir que $(k,l) \in S_{r',r''}$ pour un certain choix de $r'' \geq r$ et $r' \geq r$. Ainsi, d'après la reformulation de l'ultime périodicité ci-dessus, on sait que $L_{k,l}$ contient, pour tous $i, j \in \mathbb{N}$, un mot w tel que $|w|_a = t' + ip + k$ et $|w|_b = t' + jp + l$. À l'inverse, par définition de $L_{k,l}$, pour tout $w \in L_{k,l}$, on a $(|w|_a - t') \bmod p = k$ et $(|w|_b - t') \bmod p = l$, et $|w|_a \geq t'$ et $|w|_b \geq t'$. On a ainsi montré que $\mathrm{CCl}(L_{k,l})$ est le langage des mots $w \in \Sigma^*$ contenant un nombre de a plus grand que t' et dont la différence avec t' est congrue à k modulo p, et un nombre de b plus grand que t' et dont la différence avec t' est congrue à l modulo p. Ce langage est manifestement régulier, ce qui conclut la preuve.

Notons que le résultat de cette question généralise celui de la question 3 : en effet, il est clair que tout langage ab-borné est ultimement périodique, car on peut prendre t strictement plus grand que la borne, et p arbitraire.

Question 7. On utilise d'abord l'affirmation proposée dans l'énoncé, sans la démontrer.

Fixons le langage régulier L. On définit la fonction f qui associe à $(i,j) \in \mathbb{N}$ le quotient à gauche $(a^i b^j)^{-1}\mathrm{CCl}(L)$ de $\mathrm{CCl}(L)$. Comme $\mathrm{CCl}(L)$ est régulier, d'après l'affirmation, l'image de f est un

ensemble fini.

On montre à présent un autre lemme : pour tout ensemble fini non-vide X, pour toute fonction g de $\mathbb{N}^2$ dans X, il existe $i, j \in \mathbb{N}$ et $d_i, d_j \in \mathbb{N}^*$ tels que $g(i,j) = g(i+d_i, j) = g(i, j+d_j)$. On montre ce résultat par récurrence sur le cardinal de X. Pour le cas de base, si $|X| = 1$, on peut prendre $i := 0$, $j := 0$, $d_i := 1$, $d_j := 1$. Montrons à présent le cas de récurrence. Soit $n \in \mathbb{N}$, supposons le résultat acquis pour n, et démontrons-le pour $n+1$. Soit X un ensemble tel que $|X| = n+1$, et soit g une fonction de $\mathbb{N}^2$ dans X. Considérons la suite $(g(i,0))_{i\geq 0}$ d'éléments de X. Comme cette suite est infinie et que X est fini, il doit y avoir un élément $x \in X$ qui apparaît infiniment souvent dans cette suite, et on peut ainsi choisir une suite strictement croissante $p_0 < \cdots < p_n < \cdots$ de $\mathbb{N}$ de sorte que la suite extraite $(g(p_i, 0))_{i\geq 0}$ soit la suite constante dont tous les termes valent x. Considérons à présent la fonction g' de $\mathbb{N}^2$ dans X définie par $g'(i,j) := g(p_i, j+1)$ pour tout $(i,j) \in \mathbb{N}^2$. Soit il existe $i', j' \in \mathbb{N}$ tel que $g'(i', j') = x$, soit il n'existe pas de tels $i', j' \in \mathbb{N}$.

Dans le premier cas, s'il existe $i', j' \in \mathbb{N}$ tels que $g'(i', j') = x$, alors on a $g(p_{i'}, j'+1) = x$, et on peut ainsi prendre $i := p_{i'}$, $j := 0$, $d_i := p_{i'+1} - p_{i'}$, et $d_j := j'+1$, et obtenir le résultat, car on a $g(i,j) = g(p_{i'}, 0) = x$, on a $g(i+d_i, j) = g(p_{i'} + (p_{i'+1} - p_{i'}), 0) = g(p_{i'+1}, 0) = x$, et on a $g(i, j+d_j) = g(p_{i'}, j'+1) = x$.

Dans le second cas, s'il n'existe pas de tels $i', j' \in \mathbb{N}$, alors g' est en fait une fonction de $\mathbb{N}^2$ dans $X \setminus \{x\}$. Comme cet ensemble a cardinalité n, par application de l'hypothèse de récurrence, il existe $i', j' \in \mathbb{N}$, $d'_i, d'_j \in \mathbb{N}^*$, tels que $g'(i', j') = g'(i'+d'_i, j') = g'(i', j'+d'_j)$. Par définition de g', on peut ainsi déduire i, j, d_i, d_j satisfaisant les mêmes égalités pour g. Ceci établit le cas d'induction, et conclut la preuve du lemme.

Ainsi, en utilisant le lemme pour la fonction f définie plus haut, et pour l'ensemble fini non-vide des quotients à gauche de $\mathrm{CCl}(L)$, on en conclut qu'il existe $i, j \in \mathbb{N}$, $d_i, d_j \in \mathbb{N}^*$, tels que $f(i,j) = f(i+d_i, j) = f(i, j+d_j)$. Posons $t := \max\{i, j\}$, et $p := d_i \times d_j$. Montrons que L est ultimement périodique pour t et p. Soit $k, l \geq t$, et supposons qu'il existe un mot $w \in \mathbb{N}$ avec $(|w|_a, |w|_b) = (k, l)$. Comme la clôture commutative est commutative, ceci implique que $a^i b^j a^{k-i} b^{l-j} \in \mathrm{CCl}(L)$ (ceci est bien défini car $k \geq i$ et $l \geq j$). Ainsi, $a^{k-i} b^{l-j} \in f(i,j)$. À présent, comme $f(i,j) = f(i+d_i, j)$, on a également $a^{k-i} b^{l-j} \in f(i+d_i, j)$, d'où $a^{k-i+d_i} b^{l-j} \in f(i,j)$. En appliquant d_j fois ce procédé, on déduit que $a^{k-i+p} b^{l-j} \in f(i,j)$, de sorte que $a^i b^j a^{k-i+p} b^{l-j} \in \mathrm{CCl}(L)$. Ainsi, il existe un mot $w' \in L$ avec $(|w'|_a, |w'|_b) = (k+p, l)$. On peut montrer d'une manière analogue qu'il existe un mot $w' \in L$ avec $(|w'|_a, |w'|_b) = (k, l+p)$. Ainsi, L est bien ultimement périodique.

Il ne reste plus qu'à montrer l'affirmation supplémentaire donnée dans l'énoncé. Soit L un langage régulier, et montrons que L a un nombre fini de quotients à gauche. Soit A un automate déterministe fini *complet* qui reconnaît L, et soit Q son ensemble d'états. Pour tout mot $w \in \Sigma^*$, on note $\phi(w)$ l'état de Q auquel on aboutit dans l'automate A en lisant w. Il est facile de voir que, pour tout mot $w \in \Sigma^*$, le quotient à gauche $w^{-1}L$ est entièrement déterminé par $\phi(w)$: ce quotient $w^{-1}L$ est l'ensemble des mots w' tels qu'il existe un chemin dans A étiqueté par w' qui va de $\phi(w)$ à un état final. Ainsi, il y a au plus $|Q|$ quotients à gauche, ce qui conclut la preuve.

A4 – Énumération de marquages

On fixe l'alphabet fini $\Sigma = \{a, b\}$, et on note $\hat{\Sigma} = \{\hat{a}, \hat{b}\}$ l'alphabet des *lettres marquées*. Un *mot* $w \in \Sigma^*$ est une suite finie d'éléments de Σ. Un *marquage* d'un mot w de longueur $|w|$ est un sous-ensemble m de $\{1, \ldots, |w|\}$. L'*application* du marquage m sur le mot w est un mot $\hat{w}^m$ de longueur $|w|$ sur $\Sigma \cup \hat{\Sigma}$ défini comme suit : pour tout $1 \leq i \leq |w|$, si la i-ème lettre de w est x, alors la i-ème lettre de $\hat{w}^m$ est $\hat{x}$ si $i \in m$ et x sinon.

Un *automate à marquages* est un automate sur l'alphabet $\Sigma \cup \hat{\Sigma}$. Un tel automate A définit une fonction associant, à tout mot $w \in \Sigma$, l'ensemble noté $A(w)$ des marquages m de w tels que $\hat{w}^m$ est accepté par A. Sauf mention du contraire, on supposera toujours que les automates à marquages sont déterministes.

Question 0. Donner un automate à marquages A tel que $A(w) = \emptyset$ pour tout $w \in \Sigma^*$. Donner un automate à marquages A tel que $A(w)$ contienne uniquement le marquage $\{1, 2\}$ si w est de longueur ≥ 2, et soit l'ensemble vide sinon.

Question 1. Construire un automate à marquages A tel que $A(w)$ contienne uniquement le marquage singleton $\{|w|\}$ pour tout $w \in \Sigma^*$ non-vide.

Question 2. Construire un automate à marquages A tel que $A(w)$ soit l'ensemble des marquages singleton $\{i\}$ pour chaque entier $1 \leq i \leq |w|$ tel que la i-ème lettre de w soit $a \in \Sigma$.

Question 3. Proposer un algorithme naïf pour déterminer, étant donné un automate à marquages A sur $\Sigma \cup \hat{\Sigma}$, un mot $w \in \Sigma^*$, et un marquage m de w, si $m \in A(w)$. En déduire un algorithme naïf pour calculer, étant donné A et w, l'ensemble $A(w)$. L'implémenter en pseudocode, et déterminer sa complexité.

Question 4. Étant donné un automate à marquages A (toujours supposé déterministe) et un mot $w \in \Sigma^*$, on souhaite déterminer combien $A(w)$ contient de marquages. Proposer un algorithme efficace pour ce faire, l'implémenter en pseudocode, et en décrire la complexité.

Question 5. En déduire un algorithme moins naïf pour calculer l'ensemble $A(w)$ étant donnés A et w. En exprimer la complexité en fonction de A, de w, et de $|A(w)|$.

Question 6. Quelle est la plus faible complexité envisageable pour calculer $A(w)$? Optimiser l'algorithme précédent pour s'approcher de cette meilleure complexité.

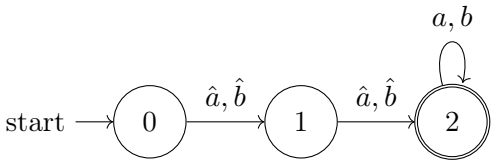
Question 7. Comment produire efficacement le i-ème marquage pour un i donné en entrée ?

Question 8. On ne suppose plus que les automates à marquages sont déterministes. Peut-on généraliser les résultats des questions précédentes ?

Corrigé

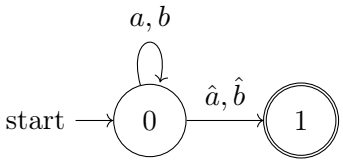
Question 0. Pour la première partie de la question, on prend simplement un automate qui n'accepte rien (sans états finaux). Ainsi l'ensemble $A(w)$ des marquages de w qui sont acceptés est vide.

Pour la seconde partie, on propose l'automate suivant :



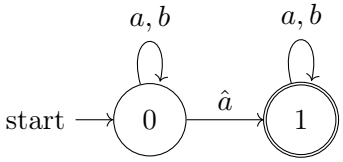
Pour un mot d'entrée w de longueur < 2, cet automate n'acceptera aucun marquage, donc $A(w) = \emptyset$. Pour un mot d'entrée w de longueur ≥ 2, cet automate lira la version marquée des deux premières lettres et la version non marquée des autres, donc $A(w)$ contient un unique marquage : $\{1, 2\}$.

Question 1. On propose :



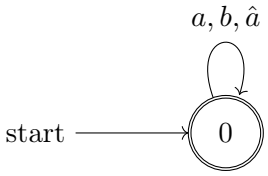
Pour un mot d'entrée w non-vide, le seul marquage que l'automate acceptera est celui où on ne marque que la dernière lettre. (Par ailleurs, on a $A(\epsilon) = \emptyset$ si le mot d'entrée est le mot vide ϵ.)

Question 2. On propose :



Étant donné un mot d'entrée w, pour qu'un marquage soit accepté, il faut et il suffit que le résultat de son application contienne un seul $\hat{a}$, donc c'est bien ce qui est demandé.

Attention, l'automate suivant ne marche pas :



Pour un mot w, l'ensemble $A(w)$ serait alors l'ensemble de tous les sous-ensembles de positions contenant un a (y compris l'ensemble vide), et non des singletons.

Question 3. On suppose que le mot d'entrée w est représenté dans un tableau, et que le marquage m est représenté dans une liste ou un tableau de booléens. L'algorithme naïf est simplement de modifier m suivant w, et de tester l'acceptation avec l'automate. La complexité est en $O(|w|)$. (Noter qu'elle ne dépend pas de l'automate, c'est normal car c'est un automate déterministe.)

On a donc l'algorithme naïf consistant à essayer tous les marquages possibles, sa complexité est en $O(2^{|w|} \times |w|)$. C'est évidemment très mauvais, et on va faire mieux.

Pseudocode (on ne demandera pas d'expliciter la représentation de l'automate, et on pourra supposer qu'on dispose d'une fonction pour tester qu'un mot est accepté par l'automate) :

```
w := mot d'entrée de n lettres
```

```
Fonction explore(i, acc):
  Si i == n:
    w2 := List.rev(acc)
    Si l'automate accepte w2:
      Produire w2
    Fin Si
  Sinon:
    x := w[i]
    explore(i+1, x::acc)
    explore(i+1, hat(x)::acc)
  Fin Si
Fin Fonction

explore(0, [])
```

Question 4. (Pour se simplifier la vie dans le corrigé, on va supposer que les automates sont complets ; on peut proposer aux candidats de faire de même si la bonne définition de « l'état obtenu après lecture de ... » devient un problème.)

Pour tout état q de l'automate, on note A_q l'automate obtenu en commençant la lecture à l'état q. On remarque d'abord une induction évidente : pour le cas de base on a $|A_q(\epsilon)| = 0$ ou $= 1$ selon que q est final ou non, et si l'on extrait la première lettre $w = xw'$, on a :

$$|A_q(w)| = \left|A_{\delta(q,x)}(w')\right| + \left|A_{\delta(q,\hat{x})}(w')\right|$$

où δ dénote la fonction de transition de l'automate.

On peut ainsi calculer les $|A_q(w[\geq i])|$ pour tous les suffixes de w de proche en proche. La complexité est en $O(|Q| \times |w|)$.

Pseudocode (on ne demandera pas d'expliciter la représentation de l'automate) :

```
w := mot d'entrée de n lettres
Count := tableau de (n+1) par |Q| cases initialisé à 0

Pour q dans Q:
  Si q est final:
    Count[n][q] := 1
  Sinon
    Count[n][q] := 0
  Fin Si
Fin Pour

Pour i de n exclu à 0 inclus:
  Pour q dans Q:
    x := w[i]
    qx := delta(q, x)
    qhatx := delta(q, hat(x))
    Count[i][q] := Count[i+1][qx] + Count[i+1][qhatx]
  Fin Pour
Fin Pour

q0 := état initial
Renvoyer Count[0][q0]
```

Question 5. On va intuitivement explorer l'arbre binaire complet de tous les marquages possibles comme en question 3. L'arbre est de hauteur $|w|$: un nœud n à profondeur i représente un marquage défini sur les i premières positions suivant le choix fait dans le chemin de la racine à n de prendre le fils gauche (ne pas marquer) ou le fils droit (marquer). Une fois parvenu à une feuille, on produira le marquage correspondant si le mot obtenu en l'appliquant est accepté.

L'idée est qu'on calcule à chaque nœud de l'arbre le nombre de marquages acceptés qui commencent par le marquage courant. Pour ce faire, on maintient au cours de notre parcours préfixe de l'arbre l'état auquel on se trouve quand on a lu le préfixe du mot d'entrée marqué comme décrit dans le préfixe de marquage de ce nœud. On utilise après la structure (précalculée) de la question précédente pour abandonner l'exploration si le sous-arbre où l'on se trouve ne contient plus aucun nœud marqué.

Quelle est la complexité de cet algorithme ? Outre le précalcul de la question précédente, appelons partie *intéressante* de l'arbre les feuilles correspondant à des marquages acceptés, ainsi que tous leurs ancêtres. Notre exploration de l'arbre ne considère que la partie intéressante, ainsi qu'éventuellement les enfants de ces nœuds (pour abandonner l'exploration à ce moment), mais ceci n'ajoute qu'un facteur constant. On effectue un traitement en temps constant à chaque nœud visité. Ainsi la complexité est proportionnelle en le nombre de nœuds de la partie intéressante. On peut la majorer par $O(|A(w)|\times|w|)$, puisque chaque feuille contribue $|w|$ nœuds au plus. (Cette majoration est grossière, mais noter que s'il y a un seul marquage dans $A(w)$ on passera bien un temps $|w|$ à le produire.) D'où une complexité totale de $O(|Q| \times |w| + |A(w)| \times |w|)$.

Question 6. On cherche une complexité en $O(\sum_{m\in A(w)} |m|)$ plus quelque chose ne dépendant que de w et A. C'est en effet la meilleure complexité « possible » puisque pour calculer tous les marquages de $A(w)$ il faut au minimum le temps de les écrire.

Pour ce faire, il faut intuitivement descendre plus vite dans l'arbre pour sauter aux feuilles, et éviter de perdre du temps à descendre étape par étape en choisissant l'enfant gauche (c'est-à-dire qu'on ne marque rien).

Formellement, on rappelle qu'un nœud est *intéressant* s'il a un descendant qui est une feuille correspondant à un marquage accepté. Un *descendant gauche* d'un nœud n est un nœud n' accessible à partir de n en descendant à gauche un certain nombre de fois. Un nœud interne est *zappable* s'il est intéressant mais que seul son enfant gauche est également intéressant.

On veut précalculer, pour chaque nœud intéressant n de l'arbre, son plus haut descendant gauche qui n'est pas zappable (possiblement une feuille). On le représente par sa profondeur et par l'état qu'on atteint alors dans l'automate dans la lecture de ce qui a été zappé.

L'algorithme sera alors le suivant. On commence à la racine, et on explore l'arbre en maintenant l'état courant (obtenu en lisant le préfixe de marquage correspondant au nœud courant). À chaque nœud, s'il est zappable, alors on saute au plus haut descendant gauche qui ne l'est pas : il n'y a rien à faire sur tous les nœuds sautés. On explore alors l'alternative correspondant à l'enfant droit (marquer la position en question, c'est-à-dire l'ajouter au marquage en cours de production dans un accumulateur) ; puis celle de l'enfant gauche (ne rien marquer) si elle est intéressante *en effectuant une récursion terminale*. Une fois parvenu à une feuille, on produit le marquage qui est actuellement dans l'accumulateur.

Il est clair que l'algorithme produit toujours le même résultat qu'avant. Justifions d'abord qu'il a la complexité attendue. Pour ce faire, le premier résultat est produit en temps constant : à chaque nœud on effectue un travail constant, et le nombre de nœuds visités est la cardinalité du marquage produit. Ensuite, l'état actuel de la pile d'appel correspond aux endroits où on a choisi d'ajouter un élément au marquage : on dépile ces appels jusqu'à trouver un endroit où le choix de ne pas mettre l'élément est intéressant, ce qui est de coût linéaire en le marquage précédent. Puis, on complète le marquage en un coût linéaire en le prochain marquage. Ainsi, au total, le nombre d'étapes est au plus deux fois la cardinalité totale des marquages.

Expliquons enfin quel précalcul on fait exactement, car on ne peut bien sûr pas faire de précalcul

réel dans l'arbre qui est de taille exponentielle. Heureusement, l'information ne dépend que la position i du nœud courant et de l'état q courant, pour nous dire qu'il faut alors zapper jusqu'à la position $i + j$ et parvenir à l'état q'. Ceci se calcule de proche en proche. Cas de base : à la fin du mot il ne faut pas zapper. Induction : à la position i et à l'état q, pour x la lettre à lire, si lire $\hat{x}$ mène à un état intéressant (c'est-à-dire $\left|A_{\delta(q,\hat{x})(w[\geq i+1])}\right| > 0$), alors quand on est à l'état q en position i on ne zappe pas. Sinon, on regarde à l'état $\delta(q, x)$ en position $i + 1$ à quelle position j et état q' on doit zapper, et c'est notre résultat. (Cette construction revient essentiellement à éliminer des ϵ-transitions dans le produit de l'automate et du mot.)

Ainsi le précalcul total est encore en $O(|Q| \times |w|)$, à quoi s'ajoute la complexité optimale recherchée.

Question 7. C'est une simple astuce : on parcourt notre arbre en considérant à chaque nœud combien de marquages sont possibles à gauche et à droite, et on récurse dans le sous-arbre gauche ou droit en fonction de comment ces nombres se comparent à l'index i désiré, qu'on ajuste quand on récurse à droite en lui soustrayant le nombre de solutions à gauche. La complexité est en $O(|w|)$ (le zapping ne nous aide pas en complexité asymptotique).

[On pourrait sans doute réaliser $O(\log |w|)$ en passant par les automates d'arbres...]

Question 8. On peut bien sûr déterminiser l'automate et appliquer les techniques des questions précédentes ; mais on paie alors une exponentielle en l'automate. La question est de savoir si on peut éviter cela, c'est-à-dire conserver une complexité polynomiale en l'automate fourni en entrée.

Le comptage, et donc la production de la i-ème solution, ne se généralisent pas. Intuitivement, le problème est que le non-déterminisme de l'automate peut nous conduire à compter plusieurs exécutions différentes mais qui reconnaîtront le même marquage. Sous une hypothèse de complexité, ce n'est pas possible.

En revanche, l'algorithme de la question 6 peut se généraliser avec une sorte de déterminisation à la volée de l'automate, et des index plus sophistiqués. Intuitivement, on n'utilise plus les comptes de la question 4 que pour savoir s'ils sont nuls ou non-nuls, ce qu'on peut effectivement calculer par une variante de la question 4 même si l'automate n'est plus déterministe. Ensuite, on maintient au cours de notre exploration l'ensemble des états où on peut se trouver sur un nœud. Les index de nœuds à zapper ne sont calculés que pour des nœuds individuels : pour chaque profondeur i et état q, on calcule la profondeur j à laquelle zapper et l'ensemble des états auquel on peut alors parvenir (ce n'est plus un seul état). Maintenant, pour savoir à un ensemble d'états Q où il faut zapper, c'est le min à prendre sur l'ensemble des nœuds individuels, donc on peut retrouver à quelle profondeur zapper. Il faut en revanche savoir à quel état on arrive à cette profondeur à partir de chaque état de Q. C'est pourquoi, à chaque profondeur i et état q, on calcule également, pour toutes les profondeurs de zapping des autres états à cette même profondeur, l'information de quels états sont accessibles.

[L'algorithme présenté ici est détaillé dans [ABMN19].]

Références

[ABMN19] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. In *ICDT*, 2019.

A3 – Automates et langages probabilistes

Pour tout ensemble non-vide X, une *distribution* sur X est une fonction π de X dans l'ensemble $\mathbb{Q}_+$ des rationnels positifs. Le *support* de π est $\text{supp}(\pi) := \{x \in X \mid \pi(x) > 0\}$. On supposera toujours que le support d'une distribution est un ensemble *fini*.

Le *produit* de π par $v \in \mathbb{Q}_+$, noté $v \cdot \pi$, est la distribution sur X définie par $(v \cdot \pi)(x) := v \times \pi(x)$ pour tout $x \in X$. Pour π et π' deux distributions sur X, leur *somme*, notée $\pi + \pi'$, est la distribution sur X définie par $(\pi + \pi')(x) := \pi(x) + \pi'(x)$ pour tout $x \in X$. On dit que la distribution π est *normalisée* si on a $\sum_{x \in X} \pi(x) = 1$. On note $\Pi(X)$ l'ensemble des distributions normalisées sur X.

On fixe un alphabet Σ. Un *langage probabiliste* L est une distribution sur l'ensemble Σ^* des mots sur l'alphabet Σ. Pour $a \in \Sigma$, on note aL le langage probabiliste défini par $(aL)(w) := L(w')$ si w peut s'écrire comme aw' (c'est-à-dire que w n'est pas vide et commence par a) et $(aL)(w) := 0$ sinon. On note L_ϵ le langage probabiliste normalisé défini par $L_\epsilon(\epsilon) := 1$ et $L_\epsilon(w) := 0$ pour tout $w \neq \epsilon$.

Un *automate probabiliste* est une paire $A = (Q, \delta)$ où $Q = \{1, \ldots, n\}$ est l'ensemble d'états et δ est une fonction de Q dans $\Pi(\{\$\} \cup (\Sigma \times Q))$ telle que, pour tout $q \in \{1, \ldots, n\}$, pour tout $(a, q') \in \text{supp}(\delta(q))$, on a $q' > q$.

On définit par récurrence le langage probabiliste $\mathcal{L}(A, i)$ accepté par A à l'état $i \in Q$: on a $\mathcal{L}(A, n) := L_\epsilon$, et, pour $1 \leq i < n$, en notant $\pi := \delta(i)$, on a :

$$\mathcal{L}(A, i) := \pi(\$) \cdot L_\epsilon + \sum_{\substack{a \in \Sigma \\ j \in Q}} \pi(a, j) \cdot (a\mathcal{L}(A, j))$$

Le *langage* $\mathcal{L}(A)$ accepté par A est le langage probabiliste $\mathcal{L}(A, 1)$.

Question 0. Construire un automate probabiliste A tel que $\mathcal{L}(A)$ soit le langage probabiliste normalisé L sur $\{a, b\}$ défini par :
- ϵ a probabilité 0.1 dans L ;
- a a probabilité 0.2 dans L ;
- abb a probabilité 0.3 dans L ;
- b a probabilité 0.4 dans L.

Question 1. Montrer que, pour tout automate probabiliste A, le langage probabiliste $\mathcal{L}(A)$ est normalisé.

Question 2. Montrer que, réciproquement, pour tout langage probabiliste normalisé L, on peut construire un automate probabiliste A_L tel que $\mathcal{L}(A_L) = L$.

Question 3. Un automate probabiliste $A = (Q, \delta)$ est dit *déterministe* si, pour tout $q \in Q$, pour chaque $a \in \Sigma$, il existe au plus un unique $q' \in Q$ tel que $(a, q') \in \text{supp}(\delta(q))$. Montrer que, pour tout langage probabiliste normalisé L, on peut construire un automate probabiliste déterministe A'_L tel que $\mathcal{L}(A'_L) = L$.

Indication : On pourra décomposer L en le mot vide et en les sous-langages, pour $a \in L$, des mots de $\text{supp}(L)$ qui commencent par a.

Suite des questions

Question 4. On appelle *uniforme* un langage probabiliste normalisé L tel que $L(u) = L(v)$ pour tous $u, v \in \mathrm{supp}(L)$. Pour $n \in \mathbb{N}$, on considère le langage probabiliste uniforme L_n sur $\Sigma = \{a, b\}$ dont le support est le langage régulier $(a + b)^n$. Calculer, en fonction de n, le nombre d'états des automates probabiliste A_{L_n} et A'_{L_n} défini par la construction des deux questions précédentes. Proposer un automate probabiliste déterministe A'_n à $n + 1$ états tel que $\mathcal{L}(A'_n) = L_n$, et justifier brièvement sa correction.

Question 5. Soit $w = a_1 \dots a_n$ un mot de Σ^* de longueur $n \in \mathbb{N}$. Pour tout $S \subseteq \{1, \dots, n\}$, on note $w_{|S}$ le sous-mot de w obtenu en conservant les lettres placées aux positions de S : formellement, en notant $i_1, \dots, i_{|S|}$ les éléments de S dans l'ordre croissant, on définit $w_{|S} := w_{i_1} \cdots w_{i_{|S|}}$. Pour tout $w' \in \Sigma^*$, on note $L_{w'}$ le langage probabiliste normalisé qui donne probabilité 1 à w'. On définit le langage probabiliste $L_{\subseteq w}$ pour tout $w \in \Sigma^*$ de longueur $n \in \mathbb{N}$ comme suit :

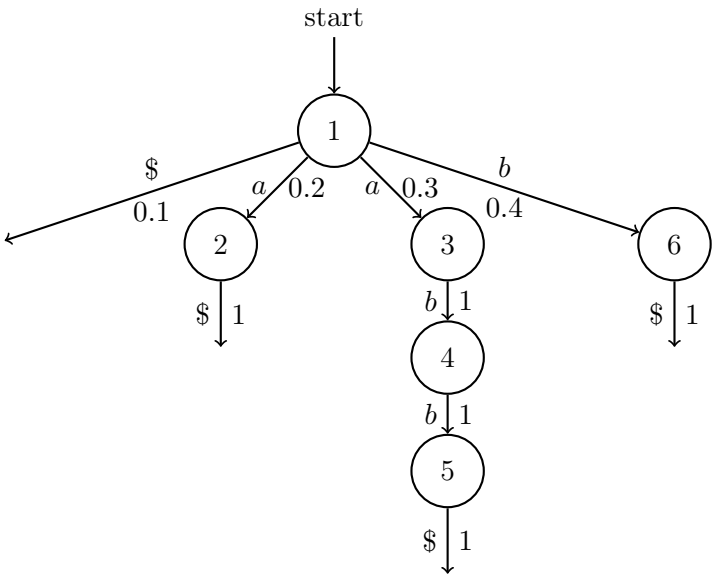
$$L_{\subseteq w} := \sum_{S \subseteq \{1,\dots,n\}} 2^{-n} \cdot L_{w_{|S}}.$$

Proposer un automate probabiliste A_w à $n+1$ états (non nécessairement déterministe) tel que $\mathcal{L}(A_w) = L_{\subseteq w}$, et prouver sa correction.

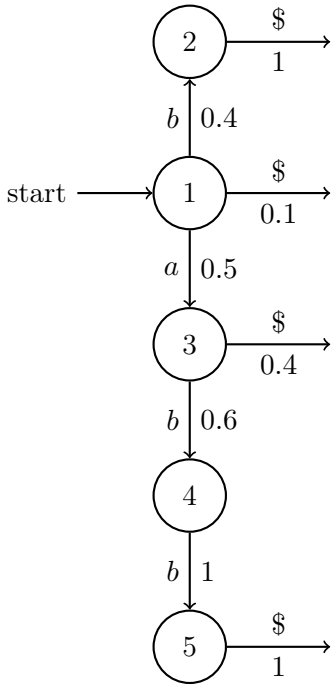
Question 6. Soit A un automate déterministe *non-probabiliste* tel que le langage $\mathcal{L}(A)$ accepté par A soit fini et non-vide. Expliquer comment construire un automate probabiliste déterministe A' tel que $\mathcal{L}(A')$ soit le langage probabiliste normalisé uniforme de support $\mathcal{L}(A)$.

Corrigé

Question 0. On peut par exemple prendre l'automate probabiliste suivant, dont on vérifie sans peine qu'il satisfait les conditions de la définition :



Cet automate correspond au résultat de la question 2 sur le langage décrit. On peut aussi accepter celui qui correspond au résultat de la question 3, à savoir :



Question 1. On montre l'affirmation pour $\mathcal{L}(A, i)$ par récurrence descendante finie avec prédécesseurs pour i de n à 1. Le cas de base correspond à $\mathcal{L}(A, n)$, qui vaut L_ϵ, et ce langage probabiliste est normalisé. Pour le cas de récurrence, supposons que l'affirmation est vraie pour un certain $1 < i \leq n$, c'est-à-dire que $\mathcal{L}(A, j)$ est normalisé pour tout $j \geq i$, et montrons que $\mathcal{L}(A, i-1)$ est normalisé. On appellera *masse* d'une distribution la somme de ses valeurs (qui vaut 1 pour les distributions normalisées) : on observe facilement que la masse de la somme de deux distributions est la somme des masses, et que la masse du produit d'une distribution par une valeur $v \in \mathbb{Q}_+$ est le produit de v et de la masse de la distribution originale. On va montrer que $\mathcal{L}(A, i-1)$ a pour masse 1. Dans la formule qui définit $\mathcal{L}(A, i-1)$, la masse du premier terme de la somme est $\pi(\$)$, et celui du deuxième terme de la somme est la somme de la masse, pour $a \in A$ et pour $j \in Q$ (ou, de façon équivalente par la condition sur le support, pour $j > i$), de $\pi(a, j)$; en effet, comme $\mathcal{L}(A, j)$ pour $j > i-1$ est normalisé par hypothèse de récurrence, il est clair que c'est également le cas de $a\mathcal{L}(A, j)$ pour tout $a \in \Sigma$, et ainsi la masse de l'expression sommée est $\pi(a, j)$. Ainsi, la masse totale est $\pi(\epsilon) + \sum_{a \in \Sigma, j > i-1} \pi(a, j)$, c'est-à-dire 1 car pi est une distribution normalisée. Ainsi, $\mathcal{L}(A, i-1)$ est normalisé, ce qui conclut le cas de récurrence. Du résultat que nous avons établi par récurrence, on déduit que $\mathcal{L}(A) = \mathcal{L}(A, 1)$ est normalisé, ce qui termine la preuve.

Question 2. Pour tout mot $w = a_1 \cdots a_n$ de Σ, on construit un automate probabiliste $A_w = (Q_w, \delta_w)$ tel que $\mathcal{L}(A_w)$ soit le langage probabiliste normalisé qui donne probabilité 1 à w et 0 à tout mot différent de w. On prend simplement $Q_w = \{1, \ldots, n+1\}$, on définit $\delta(n+1)$ comme la distribution normalisée donnant probabilité 1 à \$, et pour $1 \leq q \leq n$ on définit $\delta(q)$ comme la distribution normalisée donnant probabilité 1 à $(a_i, q+1)$. Il est clair que A_w satisfait les conditions d'un automate probabiliste, et que son langage est comme demandé ; on peut formellement montrer par récurrence descendante que $\mathcal{L}(A, i)$ pour $1 \leq i \leq n+1$ est la distribution normalisée donnant probabilité 1 au suffixe de longueur $(n+1) - i$ de w.

On construit à présent $A = (Q, \delta)$ en construisant, pour chaque mot non-vide $w \in \mathrm{supp}(L)$, l'automate $A_{w'}$ pour w' le suffixe de longueur $|w| - 1$ de w. (On construit un seul $A_{w'}$ même si plusieurs mots partagent le même suffixe w'.) On renumérote ensuite les états de chaque $A_{w'}$ à des intervalles disjoints consécutifs d'entiers à partir de 1, et on ajoute à Q un état 1 où $\delta(1)$ est défini comme suit : on donne probabilité $L(\epsilon)$ à \$, et, pour chaque mot non-vide $w \in \mathrm{supp}(L)$, en notant a la première lettre de w et

w' le suffixe de longueur $|w|-1$ de w, on donne probabilité $L(w)$ à $(a, q_{w'})$ où $q_{w'}$ est le numéro de l'état initial de $A_{w'}$. Il est clair que le résultat est bien un automate probabiliste et que son langage probabiliste est comme demandé.

Question 3. On montrera l'affirmation demandée par récurrence sur la longueur du plus long mot du support de L, c'est-à-dire que l'on montrera, pour tout $i \in \mathbb{N}$, la propriété suivante : pour tout langage probabiliste normalisé L, si le plus long mot de $\mathrm{supp}(L)$ a une longueur de i au plus, alors il existe un automate probabiliste A tel que $\mathcal{L}(A) = L$. Ceci permet d'établir le résultat, car le support d'un langage probabiliste est toujours fini.

Pour le cas de base $i = 0$, le seul langage probabiliste normalisé qui convient est le langage L_ϵ, qui est accepté par l'automate probabiliste déterministe $(\{1\}, \delta_\epsilon)$ où $\delta_\epsilon(1)$ est la distribution normalisée qui donne la probabilité 1 à \$.

Pour le cas de récurrence, supposons que l'on a prouvé l'affirmation au rang $i-1$ pour $i \in \mathbb{N}^*$, et démontrons-la au rang i. Fixons le langage probabiliste normalisé L. Pour chaque $a \in \Sigma$, soit L_a le langage probabiliste (généralement non normalisé) obtenu comme la restriction de L au domaine des mots sur l'alphabet Σ qui commencent par a. Soit s_a la masse de L_a : on appelle $\Sigma' \subseteq \Sigma$ l'ensemble $\{a \in \Sigma \mid s_a > 0\}$. Pour $a' \in \Sigma'$, le langage probabiliste $L'_{a'} := \frac{1}{s_{a'}} \cdot L_{a'}$ est alors normalisé. De plus, il est clair que l'on a :

$$L = L(\epsilon) \cdot L_\epsilon + \sum_{a' \in \Sigma'} s_{a'} \cdot L'_{a'}$$

Pour chaque $a' \in \Sigma'$, le langage L'_a est un langage probabiliste normalisé, et la longueur de son mot le plus long est strictement plus petite que la longueur du mot le plus long de L. Ainsi, par hypothèse de récurrence, pour chaque $a' \in \Sigma'$, on peut construire un automate probabiliste $A_{a'}$ tel que $\mathcal{L}(A_{a'}) = L_{a'}$; on note $n_{a'}$ son nombre d'états.

On construit à présent l'automate A pour L à partir des $A_{a'}$ pour $a' \in \Sigma'$. On renumérote d'abord les états des $A_{a'}$ par des intervalles d'entiers consécutifs disjoints, l'union disjointe de ces intervalles formant l'intervalle allant de 2 à $2 + \sum_{a' \in \Sigma'} n_{a'}$ exclu : on nomme $i_{a'}$ pour $a' \in \Sigma'$ le numéro du premier état de l'automate $A_{a'}$ après renumérotation. On ajoute ensuite un état 1 à l'automate A dont l'image par δ associe la probabilité $L(\epsilon)$ à \$, et associe, pour $a' \in \Sigma'$, la probabilité $s_{a'}$ au couple $(a', i_{a'})$. Noter que ceci garantit que la condition de déterminisme à l'état 1, et cette condition est remplie aux autres états par hypothèse de récurrence.

On vérifie que cette définition satisfait les conditions d'un automate probabiliste. Il est clair, comme L est normalisé, que $L(\epsilon) + \sum_{a'} s_{a'} = 1$, et ainsi $\delta(1)$ est normalisée ; les autres images de δ sont normalisées car les $A_{a'}$ sont des automates probabilistes. De plus, il est clair que la condition sur le support de δ est respectée, car c'est immédiat pour 1, et c'est vrai (même après renumérotation) sur les autres états, car les $A_{a'}$ sont des automates probabilistes. Il est ensuite aisé de vérifier que l'on a bien $\mathcal{L}(A) = L$, car l'équation ci-dessus correspond à la définition de $\mathcal{L}(A, 1)$, ce qui nous permet de conclure grâce à l'hypothèse de récurrence sur les $\mathcal{L}(A_{a'})$ pour $a' \in \Sigma'$. Ainsi, le cas de récurrence est montré, ce qui conclut la preuve.

Question 4. On montre facilement que, pour tout $n \in \mathbb{N}$, le nombre d'états de A_{L_n} est de $1 + 2^{n-1} \times n$. En effet, il y a d'abord l'état initial, et il y a ensuite un automate par suffixe w' dont la taille est $|w'| = (n-1)+1 = n$, et le nombre de suffixes w' est de 2^{n-1}. (Si on ne fait pas l'optimisation consistant à ne pas construire de $A_{w'}$ redondants, on obtient $1 + 2^n \times n$.)

On montre par récurrence sur $n \in \mathbb{N}$ que le nombre d'états de A'_{L_n} est de $2^{n+1} - 1$:

Cas de base $n = 0$ **:** L'automate A'_{L_0} a 1 état, ce qui est égal à $2^{0+1} - 1$;

Cas de récurrence : Si l'on suppose pour un certain $n \in \mathbb{N}$ que le nombre d'états de A'_{L_n} est de $2^{n+1} - 1$ (hypothèse de récurrence), alors le nombre d'états de $A'_{L_{n+1}}$ dans la construction précédente est de $1 + 2(2^{n+1} - 1)$, c'est-à-dire $2^{n+2} - 1$, ce qui prouve le cas de récurrence.

On peut à présent construire $A'_n = (\{1, \ldots, n+1\}, \delta'_n)$ en définissant $\delta'_n(n+1)$ comme la distribution normalisée qui donne probabilité 1 à \$, et en définissant $\delta'_n(i)$ pour $1 \leq i \leq n$ comme la distribution normalisée qui donne probabilité $1/2$ à $(a, i+1)$ et $1/2$ à $(b, i+1)$: noter que ceci remplit la condition de déterminisme. On prouve aisément par récurrence descendante finie que $\mathcal{L}(A'_n, i) = L_{n+1-i}$ pour tout $1 \leq i \leq n+1$, d'où $\mathcal{L}(A'_n) = L_n$, ce qui conclut.

Question 5. Pour construire l'automate et justifier de sa correction, on considérera les langages probabilistes $L_{\subseteq w,i}$ pour $1 \leq i \leq n+1$, définis comme $L_{\subseteq w,i} := L_{\subseteq w[i,\ldots,n]}$, où on dénote par $w[i \ldots n]$ le suffixe de longueur $n-i+1$ de w, qui commence à la position i (ainsi $w[n+1 \ldots n]$ est le mot vide ϵ). On a donc $L_{\subseteq w,1} = L_{\subseteq w}$.

On va d'abord montrer l'équation suivante :

$$L_{\subseteq w} = 2^{-n} \cdot L_\epsilon + \sum_{1 \leq i \leq n} 2^{-i} a_i \cdot L_{\subseteq w,i}.$$

Intuitivement, cette équation affirme que le mot vide a probabilité 2^{-n}, et que pour des mots non-vides, on saute un certain nombre de caractères, avec une probabilité qui décroît exponentiellement en le nombre de caractères sautés, puis on lit le prochain caractère et on lit un mot défini sur le suffixe qui suit.

Pour montrer cette équation, on va récrire l'équation originale qui définit $L_{\subseteq w}$ afin de faire apparaître les $L_{\subseteq w,i}$. Dans cette équation originale, récrivons la somme sur $S \subseteq \{1, \ldots, n\}$ pour traiter séparément le cas où $S = \emptyset$, et distinguer les autres cas suivant la plus petite valeur de S. On obtient :

$$L_{\subseteq w} = 2^{-n} \cdot L_\epsilon + \sum_{1 \leq i \leq n} \sum_{\substack{S \subseteq \{1,\ldots,n\} \\ |S| \geq 1 \\ \min S = i}} 2^{-n} \cdot L_{w_{|S}}.$$

On observe à présent que l'on peut récrire la somme interne : au lieu d'énumérer les sous-ensembles de $\{1, \ldots, n\}$ non-vides dont le plus petit élément vaut i, il est équivalent d'énumérer les sous-ensembles de $\{i+1, \ldots, n\}$ que l'on peut ajouter à i (ce sont intuitivement les éléments suivants de l'ensemble). Au demeurant, on sait, une fois qu'on a choisi le plus petit élément i de S, que le premier caractère de $L_{w_{|S}}$ est a_i, ce qui nous permet de récrire l'équation comme :

$$L_{\subseteq w} = 2^{-n} \cdot L_\epsilon + \sum_{1 \leq i \leq n} \sum_{S' \subseteq \{i+1,\ldots,n\}} 2^{-n} \cdot (a_i L_{w_{|S'}}).$$

On factorise à présent 2^{-i}, et on observe ensuite que, pour tous langages probabilistes L_1 et L_2 et $a \in \Sigma$, on a $a(L_1 + L_2) = (aL_1) + (aL_2)$. Ainsi, on peut factoriser a_i, et obtenir :

$$L_{\subseteq w} = 2^{-n} \cdot L_\epsilon + \sum_{1 \leq i \leq n} 2^{-i} a_i \cdot \sum_{S' \subseteq \{i+1,\ldots,n\}} 2^{-(n-i)} L_{w_{|S'}}.$$

On peut à présent, pour chaque valeur $1 \leq i \leq n$ de la somme extérieure, soustraire $i+1$ à toutes les valeurs de S', et remplacer w par son suffixe $w[i+1 \ldots n]$ commençant à la position $i+1$. Ainsi, on obtient :

$$L_{\subseteq w} = 2^{-n} \cdot L_\epsilon + \sum_{1 \leq i \leq n} 2^{-i} a_i \cdot \sum_{S' \subseteq \{1,\ldots,n-i\}} 2^{-(n-i)} L_{w[i+1\ldots n]_{|S'}}.$$

On reconnaît alors la définition de $L_{\subseteq w,i}$, ce qui nous permet d'aboutir à l'équation annoncée.

Cette équation suggère la construction de l'automate $A_w = (\{1, \ldots, n+1\}, \delta)$ comme suit : on fixe $\delta(n+1)$ comme la distribution normalisée qui donne probabilité 1 à \$, et pour $1 \leq i < n+1$, on définit $\delta(i)$ comme donnant la probabilité 2^{n-i+1} à \$ et, pour tout $i < j \leq n+1$, comme donnant la probabilité 2^{j-i} au couple (a_j, j). On prouve par une récurrence descendante immédiate que, pour tout $1 \leq i \leq n+1$, on a $\mathcal{L}(A, i) = L_{\subseteq w,i}$: le cas de récurrence utilise l'équation ci-dessus, appliquée aux suffixes de w.

Question 6. On considère l'automate déterministe non-probabiliste $A = (Q, q_1, F, \delta)$, où Q est l'ensemble fini d'états (on note $n := |Q|$), $q_1 \in Q$ est l'état initial, $F \subseteq Q$ est l'ensemble des états finaux, et $\delta : Q \times \Sigma \to Q$ est la fonction partielle de transition. Comme $\mathcal{L}(A)$ est fini, il est clair que, quitte à éliminer les états inutiles de A, on peut supposer que A ne contient pas de *cycle*, c'est-à-dire qu'il n'y a pas de séquence d'états $q_1, \ldots, q_n$ tel que $q_n = q_1$ et, pour tout $1 \leq i < n$, il existe $a_{i+1} \in \Sigma$ tel que $q_{i+1} \in \delta(q_i, a_{i+1})$. En effet, l'existence d'un tel cycle implique que le langage accepté par A est infini (à condition que les états du cycle soient accessibles et coaccessibles, ce qui est assuré par notre pré-traitement de A). L'absence de tels cycles assure que l'on peut renuméroter les états de sorte à avoir $Q = \{1, \ldots, n\}$ et de sorte que, pour tous $1 \leq i, j \leq n$ et $a \in \Sigma$, si $j \in \delta(i, a)$ alors $j > i$: le numéro de chaque état peut être obtenu par un tri topologique, c'est-à-dire comme le numéro dans l'inverse du tri par date de fin de visite dans une exploration en profondeur d'abord. De plus, comme on a retiré les états inutiles, on peut supposer que l'état initial est 1 (car l'état 1 est inutile sinon).

On calcule ensuite, pour chaque état $1 \leq i \leq n$, le nombre m_i de mots acceptés par l'automate modifié en prenant l'état i pour état final. Ceci peut être calculé pour i de n à 1 en prenant $m_n := 1$ (l'état n ne peut pas avoir de transitions sortantes, et il est forcément final parce qu'il est utile), et pour $1 \leq i < n$ on calcule m_i comme la somme, pour $a \in \Sigma$ tel qu'il existe $j_a > i$ tel que $j_a \in \delta(i, a)$ (où j_a est unique pour i et a parce que l'automate A est déterministe et reste déterministe après les modifications effectuées), de m_{j_a}, somme à laquelle on ajoute 1 si l'état i est final. Il est clair que ce résultat est correct parce que les ensembles de mots sur lesquels on somme pour chaque valeur de i sont disjoints (il y a le mot vide, et les autres ensembles contiennent uniquement des mots non-vides qui diffèrent par leur première lettre).

On définit enfin l'automate probabiliste $A' = (Q, \delta')$ comme suit, ce qui revient intuitivement à donner une probabilité aux transitions de A : pour tout $1 \leq i \leq n$, on définit $\delta'(i)$ comme la distribution qui donne la probabilité 0 à \$ si $i \notin F$ et $\frac{1}{m_i}$ sinon, et donne à (a, j) pour $a \in \Sigma$ et $j > i$ la probabilité 0 si $j \notin \delta(a, i)$ et m_j/m_i sinon : notons que ceci satisfait la condition de déterminisme, par déterminisme de l'automate A. Il est facile de constater que le résultat est bien un automate probabiliste, que la distribution est correctement normalisée, et on prouve par récurrence descendante sur l'état que le langage probabiliste de l'automate est correct (à savoir qu'il capture le langage probabiliste normalisé uniforme dont le support est le langage de l'automate non-probabiliste A initial où l'on rend initial l'état que nous avons numéroté i).

C3 – Langages réguliers de Delphes

Soit A et B deux alphabets et $u = u_1 \cdots u_n \in A^n$ et $v = v_1 \cdots v_n \in B^n$. On note $u * v$ le mot $(u_1, v_1) \cdots (u_n, v_n)$ de $(A \times B)^n$. Soit $\mathcal{A} = (Q, A \times B, \delta, I, F)$ un automate fini (non nécessairement déterministe) on note $L(\mathcal{A})$ le langage reconnu par $\mathcal{A}$.

Un *oracle* pour $\mathcal{A}$ est un ensemble $O \subseteq B^*$ tel $|O \cap B^n| = 1$ pour tout entier $n \in \mathbb{N}$. On note $\mathcal{A}(O)$ le langage des mots reconnus par $\mathcal{A}$ avec l'oracle O, c'est-à-dire,

$$\mathcal{A}(O) = \{u \in A^* | \exists v \in O, u * v \in L(\mathcal{A})\}$$

On appelle *langages réguliers de Delphes* les langages reconnus par des automates avec oracle.

Question 0. Montrez que le langage $\{a^n b^n \mid n \in \mathbb{N}\}$ est un langage régulier de Delphes.

Question 1. Montrez que l'intersection et l'union de deux langages réguliers de Delphes est un langage régulier de Delphes.

Question 2.

a. Montrez que la classe des langages réguliers est dénombrable.

b. Montrez que la classe des langages réguliers de Delphes n'est pas dénombrable.

Question 3. Montrez que le langage sur $\{a, b\}$ des mots avec autant de a que de b n'est pas un langage régulier de Delphes.

Question 4. Soit L un langage régulier de Delphes avec $L = \mathcal{A}(O)$. Soit $v \in B^*$, On note $E_v = \{u \mid u * v \in L(\mathcal{A})\}$.

Montrez que si L est un langage régulier alors le langage $K = \{v \mid E_v \subseteq L\}$ est un langage régulier.

Question 5. Dans la suite, on note $<_{\text{lex}}$ l'ordre lexicographique sur les mots pour un ordre abitraire sur l'alphabet A.

Soit L un langage régulier sur l'alphabet A, montrez que le langage $L_{\text{lex}} = \{u \mid u \in L \wedge \exists v \in L \cap A^{|u|}, v <_{\text{lex}} u\}$ est également un langage régulier.

Question 6. Soit L un langage régulier reconnu par un automate avec oracle $\mathcal{A}(O)$. Montrez qu'il existe un oracle O' régulier tel que $\mathcal{A}(O) = \mathcal{A}(O') = L$.

Question 7. Un oracle O est dit *uniforme* s'il existe un mot infini $w \in B^\omega$ tel que O est la suite des préfixes de w. Dans ce cas, on note $\mathcal{A}(w)$ pour $\mathcal{A}(O)$.

a. Montrez que $\{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable par un automate avec oracle uniforme.

b. Pour K un langage, on note $\textsc{Prefix}(K)$ l'ensemble de ces préfixes. C'est-à-dire, l'ensemble $\textsc{Prefix} = \{u | \exists v \in A^*, uv \in K\}$. Un langage K est clos par préfixe si $K = \textsc{Prefix}(K)$. Montrez que si K est langage régulier non fini clos par préfixe alors il existe u et v tel que $\textsc{Prefix}(uv^*) \subseteq K$.

c. Soit L un langage régulier reconnu par un automate avec oracle uniforme $\mathcal{A}(w)$. Montrez qu'il existe $u, v \in B^*$ tel que tel que $\mathcal{A}(O) = \mathcal{A}(uv^\omega)$.

Corrigé

Question 0. Il suffit de prendre pour l'oracle $0^{\frac{n}{2}}10^{\frac{n}{2}}$, qui marque la moitié du mot.

Question 1. Une construction d'automate produit sans difficulté particulière.

Question 2. Les automates sont dénombrables, on peut utiliser n'importe des suites finis d'entiers dans $\mathbb{N}$ pour coder ensemble, alphabet, transitions, ect et conclure.

Les automates de Delphes ne sont pas dénombrable : on peut prendre un langage qui accepte le développement décimale d'un réel en utilisant l'oracle pour ça.

Question 3. Pleins d'arguments sont possibles. Voici une proposition. On regarde les mots de la forme $u_i = a^i b^{n-i}$ sur un oracle de taille $2n$. Nécessairement il existe i, j tel que $q.u_i = q.u_j$ mais il existe v tel que $u_i v \in L$ et $u_j v \notin L$ ce qui donne une contradiction.

Question 4. On construit un automate pour K en prenant le produit cartésien pour $\mathcal{A}$ avec l'automate pour L en effaçant les transition de l'alphabet A.

Formellement $(q, q').b \to (p, p')$ s'il existe a tel que $q.(a, b) = p$ et $q'.a = p'$.

Ça donne un automate non déterministe mais il marche.

Question 5. On réalise une copie de l'automate qu'on encode par un produit cartésien et une autre copie qui stock la valeur d'un mot virtuelle. On va deviner non déterministiquement le mot v associer au mot u.

Dans la première copie, v et u on la même valeur, dans la seconde copie ils peuvent diverger arbitrairement.

pour chaque $(q, 0)$ on a les transitions $(q, 0).a = (q.a, 0)$ et $(q, 0).a = (q.a, 1, q.b)$ pour $b < a$. Et $(q, 1, q').a = (q.a, 1, q'.b)$ pour toute lettre b.

Question 6. On utilise encore une fois K mais il faut extraire de K un oracle régulier, i.e. un langage régulier tel que $A^n \cap T$ est de taille 1 pour tout n. Pour ça on peut prendre $K \setminus K^c_{\text{lex}}$ qui a la bonne forme puisque ça sélectionne le plus petit mot de K pour chaque taille par ordre lexicographique.

Question 7.

a. Encore un argument de pompage. On peut utiliser le fait qu'un configuration de l'algorithme c'est une paire (q, i) où i est la position sur l'oracle. Si on prends n suffisamment grand et on considère $c_i = (q_{\text{init}}.a^{2n+i}b^{n-i}n, 3n)$ pour $i \in \{0, \dots, |Q|\}$. Il y a nécessairement deux configurations égale, on peut conclure.

b. Pas de difficulté ici. Pour tout langage régulier non fini, on peut trouver u, v, w avec v non vide tel que $uv^*w \subseteq K$. Le reste suit.

c. Il faut prendre un langage un peu différent de la question 5 : $\{v \mid E_{v'} \subseteq L \text{ pour } v' \in \textsc{Prefix}(v)\}$. Montrer que c'est un langage régulier non vide, non fini, et conclure avec la question d'avant.

A3 – Maintenance incrémentale de langages réguliers

On fixe un alphabet fini Σ. Un *mot* $w \in \Sigma^*$ est une suite finie d'éléments de Σ, et un langage $L \subseteq \Sigma^*$ est un ensemble de mots. Un langage est *régulier* s'il est reconnu par un automate fini, ou dénoté par une expression rationnelle (on admet que ces caractérisations sont équivalentes).

Question 0. Si l'on fixe un langage régulier $L \subseteq \Sigma^*$, le problème d'*appartenance à* L est de déterminer, étant donné en entrée un mot $w \in \Sigma^*$, si $w \in L$. Quelle est la complexité du problème d'appartenance à L en fonction de la longueur du mot d'entrée ?

Ce sujet s'intéresse à la *complexité incrémentale* du problème d'appartenance à un langage régulier L. Dans ce problème, on reçoit en entrée un mot $w \in \Sigma^*$ de longueur n. On effectue d'abord un *pré-traitement* pour déterminer si $w \in L$ et pour construire si on le souhaite une structure de données auxiliaire : cette phase de pré-traitement doit s'exécuter en $O(n)$. Ensuite, on reçoit des *mises à jour*, c'est-à-dire des paires (i, a) pour $1 \leq i \leq n$ et $a \in \Sigma$, données l'une après l'autre. À chaque mise à jour, on modifie le mot w pour que sa i-ème lettre devienne a, et on doit déterminer si $w \in L$ après cette modification. La longueur n du mot ne change jamais. La *complexité incrémentale* d'un langage est la complexité dans le pire cas pour prendre en compte une mise à jour, exprimée en fonction de n.

Question 1. Montrer que tout langage régulier a une complexité incrémentale en $O(n)$.

Question 2. Montrer que le langage régulier a^* sur l'alphabet $\Sigma = \{a, b\}$ a une complexité incrémentale en $O(1)$.

Question 3. Soit L_3 le langage des mots sur l'alphabet $\Sigma = \{a, b\}$ comportant au moins deux a, un nombre pair de a, et un nombre de b qui n'est pas divisible par 3. Ce langage est-il régulier ? Quelle est sa complexité incrémentale ?

Question 4. On dénote par $w_1, \ldots, w_n$ les lettres d'un mot $w \in \Sigma^*$ de longueur n. Pour toute permutation $\sigma \colon \{1, \ldots, n\} \to \{1, \ldots, n\}$, on écrit par abus de notation $\sigma(w)$ pour désigner le mot $w_{\sigma(1)} \cdots w_{\sigma(n)}$. Un langage L est *commutatif* si pour tout $w \in \Sigma^*$, pour n la longueur de w, pour toute permutation $\sigma \colon \{1, \ldots, n\} \to \{1, \ldots, n\}$, on a $w \in L$ si et seulement si $\sigma(w) \in L$.

Montrer que tout langage régulier commutatif a une complexité incrémentale en $O(1)$.

Question 5. Proposer une structure de données pour stocker un ensemble d'entiers S qui supporte les opérations suivantes :

- Ajouter un entier dans S, en $O(1)$;
- Retirer un entier de S, en $O(1)$;
- Parcourir les entiers actuellement stockés dans S, en $O(|S|)$.

Écrire le pseudocode pour ces opérations.

Question 6. On considère le langage L_6 sur l'alphabet $\Sigma = \{a, b, c\}$ dénoté par l'expression rationnelle $c^*ac^*bc^*$. Montrer que sa complexité incrémentale est $O(1)$ en utilisant la structure de données de la question précédente.

Suite des questions

Question 7. À quelle classe de langages peut-on généraliser cette technique ?

Question 8. Soient deux langages L_1 et L_2 de complexité incrémentale en $O(1)$. Quelle est la complexité incrémentale des langages $L_1 \cap L_2$ et $L_1 \cup L_2$?

Question 9. On s'intéresse aux langages réguliers admettant au moins une "lettre neutre" (notion que le candidat ou la candidate aura dû identifier à la question 7). Proposer une classe aussi générale que possible de langages de complexité incrémentale en $O(1)$.

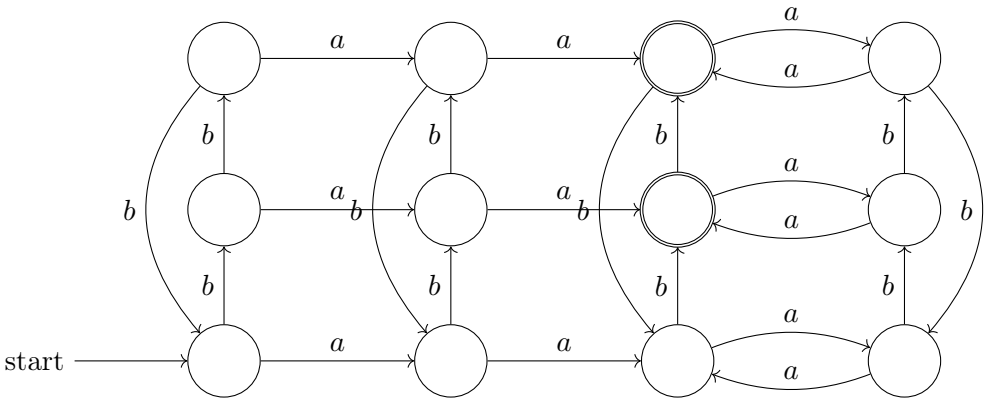
Corrigé

Question 0. On se donne un automate déterministe pour le langage L (on n'a pas besoin de se demander de la complexité de le calculer). Étant donné le mot w, on simule son exécution dans l'automate. La complexité est donc en $O(n)$. *[C'est surtout une question de cours, on attend juste $O(n)$ avec une phrase d'explication. On ne demande pas la complexité en fonction de l'automate ou autre représentation du langage.]*

Question 1. On stocke le mot w dans un tableau (ce qu'on fera toujours par la suite). Il suffit, à chaque mise à jour, de refléter la modification sur le tableau, puis de réévaluer si le mot courant appartient ou non au langage avec la question précédente.

Question 2. On maintient un compteur du nombre courant de a et du nombre courant de b, qu'on initialise au prétraitement. On maintient à jour ces compteurs à chaque mise à jour (ceci nécessite de connaître le nouveau caractère, et l'ancien caractère qui vient d'être changé, qu'on retrouve avec le tableau). Le mot courant est dans le langage si et seulement si le nombre de b est de 0.

Question 3. Le langage est régulier. On peut exhiber un automate :



On peut aussi remarquer plus intelligemment que c'est l'intersection de trois langages dont on montre facilement qu'ils sont réguliers (en exhibant un automate), or on sait d'après le cours que les langages réguliers sont clos par intersection.

Sa complexité incrémentale est en $O(1)$: on maintient un compteur du nombre de a et de b et on teste la condition.

Question 4. *Indication : considérer un automate déterministe et précalculer des chemins.*

On considère un automate déterministe pour le langage concerné. Dans le cadre du précalcul en $O(n)$, on calcule pour chaque état q de l'automate, pour chaque lettre $a \in \Sigma^*$, pour chaque entier

$0 \leq i \leq n$, la fonction $\delta^i(q, a)$ qui indique l'état auquel on aboutit après lecture de a^i depuis q. Ce calcul se fait de proche en proche : $\delta^0(q, a) := q$ et $\delta^{i+1}(q, a) := \delta(\delta^i(q, a), a)$ pour $\delta = \delta^i$ la fonction de transition de l'automate. Noter que le nombre d'états de l'automate et de lettres est une constante indépendante de la longueur n du mot, donc ce précalcul est bien en $O(n)$.

On maintient avec complexité incrémentale $O(1)$ un compteur du nombre d'occurrences de chaque lettre dans le mot, comme à la question précédente.

Pour savoir si le mot courant appartient à L, on utilise le fait que L est commutatif pour savoir que c'est le cas si et seulement si le mot $a_1^{n(a_1)} a_2^{n(a_2)} \cdots a_k^{n(a_k)}$ y appartient, où $a_1, \ldots, a_k$ désigne les lettres de Σ et $n(a)$ désigne le nombre d'occurrences dans le mot courant de la lettre a, qui est l'information maintenue par les compteurs. En effet, par définition des compteurs, le mot en question est une permutation du mot courant.

Or l'acceptation de ce mot par l'automate se détermine en $O(1)$ à l'aide des fonctions précalculées : on calcule $q_1 = \delta^{n(a_1)}(q_0, a_1)$, puis $q_2 = \delta^{n(a_2)}(q_1, a_2)$, et ainsi de suite, en k étapes où k est la taille de l'alphabet qui est une constante indépendante de n. On teste enfin si le dernier état obtenu est final ou non.

Question 5. *Indication : utiliser un tableau, mais s'inspirer de la structure de liste.*

Le problème est qu'une simple liste ne permet pas de retirer un élément identifié par sa valeur (il faut retrouver le maillon de liste qui le stocke), et qu'un tableau ne permet pas de parcourir efficacement les cases occupées. On pourrait naturellement concilier les deux, c'est-à-dire une structure de liste avec un tableau stockant des pointeurs vers les maillons. Pour que la suppression soit plus simple, on utiliserait spontanément des listes doublement chaînées.

Ceci dit, les pointeurs et les listes chaînées ne sont pas au programme, donc on attend plutôt une solution élémentaire à base de tableaux uniquement.

On stocke un tableau "suivant", un tableau "précédent", et une variable "début", avec l'invariant que les éléments actuellement stockés peuvent être parcourus comme début, suivant[début], suivant[suivant[début]], etc., jusqu'à atteindre la valeur -1.

```
début := -1
suivant := tableau d'entiers de taille n initialisé à -1
précédent := tableau d'entiers de taille n initialisé à -1

Fonction Énumérer():
  courant := début
  Tant que courant != -1:
    Produire courant
    courant := suivant[courant]
  Fin Tant que
Fin Fonction

Fonction Ajouter(x):
  // On ajoute au début
  Assertion(suivant[x] == précédent[x] == -1)
  suivant[x] := début
  // précédent[x] reste à -1
  Si début != -1:
    précédent[début] := x
  Fin Si
  début := x
Fin Fonction
```

```
Fonction Supprimer(x):
  // Si on supprime le premier élément, on change début
  Si début == x:
    début := suivant[x]
  Fin Si
  // On saute par dessus l'élément supprimé
  Si suivant[x] != -1:
    précédent[suivant[x]] := précédent[x]
  Fin Si
  Si précédent[x] != -1:
    suivant[précédent[x]] := suivant[x]
  Fin Si
  // Maintenant, on supprime
  suivant[x] := -1
  précédent[x] := -1
Fin Fonction
```

Noter que bien sûr les éléments ne sont pas parcourus dans l'ordre.

Question 6. On maintient une structure de données de la question 5 pour l'ensemble de positions contenant la lettre a, une autre pour l'ensemble des positions contenant la lettre b, en $O(1)$ par la question précédente. On maintient le nombre de a et de b comme aux questions 2–4, en $O(1)$.

Pour savoir si le mot courant est dans L, on vérifie d'abord s'il y a exactement un a et un b. Si non, alors la réponse est non. Si oui, alors on énumère le contenu des deux structures, ce qui est en $O(1)$ car elles contiennent chacune un élément. On trouve aussi la position de l'unique a et de l'unique b, et on les compare, pour savoir si on est ou non dans L.

Question 7. Intuitivement, si on peut partitionner l'alphabet Σ entre des lettres qui n'ont pas d'effet et des lettres sur lesquelles on doit réaliser un mot fini (ou un langage fini), alors la même technique s'appliquera.

Pour être précis (on attend une telle formalisation du candidat ou de la candidate) : on définit une lettre a qui est *neutre* pour L comme à la question 13. On définit le *réduit* de L pour une lettre neutre a comme le langage $L^{-a} := \{w^{-a} \mid w \in L\}$. On étend cette définition à un ensemble non-vide de lettres neutres : $L^{-\Sigma'}$ pour $\Sigma' \subseteq \Sigma$ non-vide est l'ensemble des $w^{-\Sigma'}$ pour $w \in L$ où $w^{-\Sigma'}$ s'obtient en retirant de w toutes les lettres de Σ'. Pour un langage L avec un ensemble non-vide Σ' de lettres neutres, si $L^{-\Sigma'}$ est fini, alors la complexité incrémentale est en $O(1)$ comme à la question précédente.

Pour le montrer, on crée une structure de données de la question 5 pour chaque lettre de $\Sigma \setminus \Sigma'$. On maintient ces structures, ainsi que le nombre d'occurrences de chaque lettre, en $O(1)$. Soit N la longueur maximale d'un mot de $L^{-\Sigma'}$; c'est une constante indépendante de n. Pour savoir si le mot courant appartient ou non à L, tant que le nombre d'occurrences total des lettres de $\Sigma \setminus \Sigma'$ est $> N$, on peut répondre non directement, en $O(1)$. Sinon, on utilise les structures de données de la question 5 pour identifier le sous-ensemble (de taille au plus N) de positions contenant des lettres de Σ'. On le trie en temps $O(N \log N)$, ce qui est toujours constant. On lit le mot formé et on teste (en temps constant car il est de taille $\leq N$) s'il appartient à $L^{-\Sigma'}$, ce qui conclut.

Question 8. Si on peut maintenir une structure avec complexité incrémentale en $O(1)$ pour l'appartenance à L_1, et pour l'appartenance à L_2, alors ces structures nous permettent de savoir si on appartient à L_1 et à L_2, ou à L_1 ou à L_2, toujours en $O(1)$.

Question 9. La réponse naïve (mais nécessitant un peu de recul) est : la clôture par intersections et par unions de la classe de la question 7 et des langages commutatifs couverts en question 4. Mais on s'attend à ce que le candidat ou la candidate sache se représenter la puissance d'expression de cette classe :

— L'intersection d'un langage de la question 7 avec un langage commutatif permet de poser des conditions commutatives sur les lettres neutres (qui ne sont du coup plus neutres), mais elle n'est pas intéressante sur les autres lettres (puisque le langage est fini)

— L'union de tels langages est intéressante parce qu'elle permet de changer la partition entre lettres neutres et non-neutres. En d'autres termes, on a une complexité incrémentale de $O(1)$ pour le langage "j'ai un nombre pair de a, une lettre neutre b, et exactement un c avant exactement un d, OU j'ai un nombre pair de c, une lettre neutre d, et exactement un a avant exactement un b".

— L'union permet aussi, pour la même partition, de poser des conditions commutatives différentes suivant ce qui est attendu pour les lettres non-neutres, par exemple couvrir des langages comme "j'ai une lettre neutre d, et j'ai exactement un a avant exactement un b et un nombre pair de c OU exactement un b avant exactement un a et un nombre impair de c.

Pour résumer, une forme normale serait : une union finie de langages qui sont l'entrelacement (shuffle) d'un langage commutatif sur un alphabet Σ', et d'un singleton (un seul mot) sur l'alphabet $\Sigma \setminus \Sigma'$. *[Cette caractérisation est celle des langages ZG, voir [AJP21, AP21]. On peut conjecturer qu'il s'agit des seuls langages réguliers de complexité incrémentale $O(1)$, ou en tout cas c'est le cas sous une certaine hypothèse de complexité.]*

Références

[AJP21] Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic Membership for Regular Languages. In *ICALP*, 2021.

[AP21] Antoine Amarilli and Charles Paperman. Locality and Centrality: The Variety ZG. Preprint : `http://arxiv.org/abs/2102.07724`, 2021.

A2 – Langages réguliers épars

On fixe l'alphabet fini $\Sigma = \{a, b\}$. Un *mot* $w \in \Sigma^*$ est une suite finie d'éléments de Σ, et un langage $L \subseteq \Sigma^*$ est un ensemble de mots. La *densité* de L est la fonction $\delta_L : \mathbb{N} \to \mathbb{N}$ où $\delta_L(n) := |L \cap \Sigma^n|$ pour $n \in \mathbb{N}$ est le nombre de mots de longueur n dans L. On dit que L est *épars* si on a $\delta_L = O(P)$ pour un polynôme P.

Un langage *régulier* est un langage dénoté par une expression rationnelle, ou reconnu par un automate déterministe fini (on admet que ces caractérisations sont équivalentes). Tous les automates sont supposés déterministes.

Question 0. Donner un exemple d'un langage régulier épars, et d'un langage régulier non-épars.

Question 1. Est-il vrai que l'union de deux langages réguliers épars est un langage régulier épars ? Que penser de la concaténation ? Que penser de l'itération (étoile de Kleene) ?

Question 2. Est-il vrai qu'un langage est épars si et seulement si son complémentaire ne l'est pas ?

Question 3. Soient u, v, v', w des mots de Σ^*. Supposons qu'un langage L contienne tous les mots du langage dénoté par l'expression régulière $u(av|bv')^*w$. Montrer que L n'est pas épars.

Question 4. En s'inspirant de la question précédente, proposer une condition suffisante sur un automate fini pour que le langage accepté par l'automate ne soit pas épars.

Question 5. Écrire le pseudocode d'un algorithme naïf pour tester le critère de la question 4 sur un automate fourni en entrée, et discuter de sa complexité.

Question 6. On suppose qu'étant donné l'automate, on dispose d'une fonction permettant de calculer les composantes fortement connexes de son graphe en temps linéaire. En déduire un algorithme pour tester le critère de la question 4 en temps linéaire.

Question 7. Montrer que le critère identifié en question 4 est en réalité une caractérisation des automates reconnaissant des langages qui ne sont pas épars.

Question 8. Conclure à la caractérisation suivante : les langages réguliers épars sont exactement les unions finies de langages de la forme $u_0 v_1^* u_1 v_2^* u_2 \cdots v_k^* u_k$ pour des mots $u_0, \ldots, u_k, v_1, \ldots, v_k \in \Sigma^*$.

Suite des questions

Question 9. Quels sont les régimes possibles pour la fonction de densité d'un langage régulier, et quels sont les langages les réalisant ?

Question 10. Quelle est la complexité, étant donné un automate, de déterminer le régime de la fonction de densité de son langage ?

Corrigé

Question 0. Le langage vide est épars. Le langage Σ^* ne l'est pas puisque δ_{Σ^*} est la fonction qui à n associe 2^n, qui n'est pas dominée par un polynôme.

Question 1. On note d'abord que les langages réguliers sont clos par ces opérateurs, donc il suffit de vérifier si les langages épars le sont. *[On s'attend bien à ce que la clôture des langages réguliers par ces opérations soit mentionnée.]*

Les langages épars sont clos par union : pour L_1 et L_2 deux langages épars, si on a $\delta_{L_1} = O(P_1)$ et $\delta_{L_2} = O(P_2)$ pour deux polynômes P_1 et P_2, alors comme pour tout $n \in \mathbb{N}$ on a $|(L_1 \cup L_2) \cap \Sigma^n| = |L_1 \cap \Sigma^n \cup L_2 \cap \Sigma^n| \leq |L_1 \cap \Sigma^n| + |L_2 \cap \Sigma^n|$, on a $\delta_{L_1 \cup L_2} = O(P_1 + P_2)$ et $P_1 + P_2$ est également un polynôme. *[On exigera la preuve formelle.]*

Les langages épars sont clos par concaténation : soient L_1 et L_2 deux langages épars. Pour simplifier, on les suppose dominés par un polynôme commun, de sorte à ce que $\delta_{L_1} = O(P)$ et $\delta_{L_2} = O(P)$ pour un polynôme P, que pour simplifier l'on supposera également croissant. Pour tout $n \in \mathbb{N}$, on a $|L_1 L_2 \cap \Sigma^n| = \left|\cup_{0 \leq i \leq n} (L_1 \cap \Sigma^i) \times (L_2 \cap \Sigma^{n-i})\right|$. Ceci est majoré par $\sum_{0 \leq i \leq n} \left|L_1 \cap \Sigma^i\right| \times \left|L_2 \cap \Sigma^{n-i}\right|$, majorable par hypothèse par $\sum_{0 \leq i \leq n} P(i)P(n-i)$. Comme P est croissant ceci se majore grossièrement par $(n+1)P(n)^2$. C'est également un polynôme.

Les langages épars ne sont manifestement pas clos par étoile : le langage fini $a + b$ est épars mais son itération est Σ^* qui ne l'est pas.

Question 2. C'est faux, car il se peut qu'aucun des deux ne soit épars : le complémentaire de $a\Sigma^*$ est $b\Sigma^*|\epsilon$ et aucun des deux n'est épars.

En revanche, il est bien sûr impossible qu'un langage et son complémentaire soient tous deux épars, puisque leur union serait alors éparse par la question précédente, or il s'agit de Σ^* qui ne l'est pas.

Question 3. Soit L' le langage de l'expression rationnelle. Il suffit de montrer que L' n'est pas épars pour conclure. *[La formulation de la question avec L et L' est pour aider pour la question suivante.]*

Posons n_0 la somme des longueurs de u et w, posons m le PPCM des longueurs de av et bv' de sorte que $m = p\,|av| = q\,|bv'|$, et étudions $\delta_{L'}$ aux valeurs $n_0 + km$ pour $k \in \mathbb{N}$. Pour chaque "bloc" de taille m, on a au moins deux possibilités (manifestement différentes) : le remplir par des av, ou le remplir par des bv'. (Bien sûr, on a potentiellement davantage de possibilités en réalité.) Mais ceci assure que $\delta_{L'}(n_0 + km) \geq 2^k$, manifestement non dominable par un polynôme.

Question 4. *[Attention, pour le bon déroulement de la suite du sujet, il faut parvenir exactement à la formulation proposée.]*

La question 3 suggère que si l'on considère un automate déterministe alors il reconnaîtra un langage non-épars s'il y a un état q satisfaisant les conditions suivantes :

— Il y a un chemin (correspondant à u) allant de l'état initial à q ;

— Il y a une transition a allant de q à un état q', et un chemin (correspondant à v) revenant de q' à q ;

— Il y a une transition b allant de q à un état q'', et un chemin (correspondant à v') revenant de q'' à q ;

— Il y a un chemin (correspondant à w) allant de q à un état final.

On peut reformuler : il y a un état q dans l'automate qui est accessible et co-accessible (attention, ces deux termes ne sont pas au programme) avec deux boucles différentes allant de q à lui-même, l'une commençant par a et l'autre par b.

C'est une condition suffisante : si un automate présente ce motif alors il reconnaît un langage non épars par la question précédente. La question 7 montre qu'il s'agit en réalité d'une condition nécessaire et suffisante : un automate sans ce motif reconnaît un langage épars.

Question 5. Attention, le graphe correspondant à l'automate est en réalité un multi-graphe, avec potentiellement des boucles. Les détails de la représentation d'entrée ne sont pas spécifiés pour pouvoir en discuter avec le candidat ou la candidate, qui peut faire les choix qui l'arrangent. Il faut être indulgent toutefois car les multi-arêtes et les boucles ne sont explicitement pas au programme.

Pour commencer, il ne faut pas oublier d'éliminer les états inutiles. On peut le faire naïvement en temps quadratique, voire cubique (pour chaque état, pour chaque état final, faire un test d'accessibilité ; et de même avec l'état initial), ou plus intelligemment en temps linéaire : exploration BFS à partir de l'état initial pour marquer les accessibles, et une seule exploration BFS à partir de l'ensemble des états finaux (et en inversant le sens des arêtes pour marquer les co-accessibles). *[Pour cette question, ne pas exiger d'algorithme efficace pour cela : cf la question suivante.]*

Ensuite, on teste le motif à rechercher : pour chaque choix de sommet q ayant deux transitions a et b allant respectivement vers q' et q'' (noter que bien sûr q, q', q'' ne sont pas nécessairement distincts), on teste s'il y a un chemin de q' à q et de q'' à q. C'est en $O(nm)$ pour n le nombre d'états et m le nombre de transitions.

Par exemple :

```
Input: graphe G de n sommets représenté par listes d'adjacences

Fonction Accessible(u, v):
  Q := file()
  Q.insère(u)
  Visité = tableau de n cases initialisé à False
  Tant que Q n'est pas vide:
    x := Q.top()
    Q.pop()
    Si x == v:
      Renvoyer Vrai
    Fin Si
    Si Visité[x]:
      Continuer
    Fin Si
    Visité[x] := Vrai
    Pour chaque (x, y):
      Q.push(y)
    Fin Pour
  Fin Tant que
  Renvoyer Faux
Fin Fonction

Utile = tableau de n cases initialisé à Vrai
```

```
Pour chaque état q:
  Si non Accessible(qinit, q):
    Utile[q] := Faux
  Fin Si
Fin Pour

Pour chaque état q:
  OK := Faux
  Pour chaque état final qf:
    Si accessible(q, qf):
      OK := Vrai
      Break
    Fin Si
  Fin Pour
  Si non OK:
    Utile[q] := Faux
  Fin Si
Fin Pour

Pour chaque q ayant deux transitions (q, qa) et (q, qb):
  Si Utile[q] et accessible(qa, q) et accessible(qb, q):
    Renvoyer Vrai
  Fin Si
Fin Pour

Renvoyer Faux
```

Question 6. *[On rappelle que la notion de composante fortement connexe (CFC) d'un graphe orienté est exigible.]*

Il faut comme précédemment éliminer les états non-accessibles et non-coaccessibles, mais cette fois il faut impérativement le faire en temps linéaire.

On note que le motif que l'on recherche se situe forcément dans une CFC, donc il suffit de considérer chaque CFC.

À présent, notons qu'une occurrence du motif interdit dans une CFC témoigne de l'existence d'un état q avec des transitions a et b menant à des états respectifs q' et q'' qui sont dans la même CFC. Mais réciproquement, s'il y a un tel état q, alors comme q' et q'' sont dans la même CFC que q, il y a un chemin retour, et donc un motif interdit.

Pour le reformuler autrement, la condition revient à affirmer que chaque CFC est soit un cycle, soit un cycle vide (un seul état dont toutes les transitions sortantes sortent de la CFC).

Pour résumer, après élimination des états inutiles, si on suppose chaque état annoté par un identifiant de sa CFC, alors il suffit de vérifier chaque état q pour tester si on a deux transitions sortantes qui sont vers des états de la même CFC que q (potentiellement identiques, potentiellement q lui-même). C'est manifestement faisable en temps linéaire.

Question 7. Montrons que si l'automate n'a pas le motif interdit, ou autrement dit si chacune de ses CFC est un cycle (après suppression des états inutiles), alors le langage qu'il reconnaît est épars. L'automate étant déterministe, chaque mot de son langage a un unique chemin acceptant (allant d'un état initial à un état final). Notons K le nombre de CFC ; c'est une constante.

On considère l'application ϕ associant à chaque mot les informations suivantes sur son chemin

acceptant : pour chaque CFC, l'information de la première et dernière position du chemin acceptant où on est dans cette CFC (c'est clairement un segment du chemin, noter en particulier que par définition on ne peut pas revenir dans une CFC une fois qu'on en est parti), et les états à cette première et dernière position. L'application ϕ associe donc à un mot de longueur n un K-tuple d'entiers entre 0 et n et un $2K$-tuple d'états de Q. Ainsi, il y a au plus $(n+1)^K \times |Q|^{2K}$ images possibles ; c'est un polynôme P de degré K.

Or, l'application ϕ est injective. En effet, sachant qu'un chemin acceptant se trouve à l'état q en position i et r en position j, le chemin entre i et j doit rester dans la CFC commune de q et r, et comme cette CFC est un cycle il y a un seul chemin possible.

Pour L le langage accepté par l'automate, on a donc $|L \cap \Sigma^n| \leq P$ par ce qu'on vient de dire, ce qui conclut que le langage est épars.

Question 8. Un langage de la forme indiquée est manifestement régulier, et épars pour la même raison qu'en question précédente. Ainsi, des unions finies de tels langages sont des langages réguliers épars.

Pour la réciproque, on raisonne comme à la question précédente : un langage régulier épars est accepté par un automate qui satisfait donc le critère de la question 4, or le langage accepté par un tel automate s'écrit comme une union finie de langages de la forme requise suivant leur image par la fonction ϕ. Intuitivement, les u_i correspondent à des tours complets de boucle dans une CFC, et les v_i correspondent à des déplacements entre CFC ou à des tours incomplets de boucle dans une CFC (qu'on peut mettre par convention à la fin du segment où l'on reste dans une CFC).

Question 9. Il y a les langages non-épars dont la fonction de densité est $\Theta(2^{\Theta(n)})$, que nous avons caractérisés comme ceux n'ayant pas la forme de la question 8, ou acceptés par un automate comportant le motif interdit de la question 4. Ensuite, on peut montrer que pour un langage épars L, les conditions suivantes sont équivalentes :

1. L est de densité $O(n^k)$;
2. L est accepté par un automate où il n'y a pas de chemin acceptant qui passe par $k+1$ CFC cycliques (CFC qui ne sont pas un singleton sans boucle) ;
3. L est une union finie de langages de la forme $u_0 v_1^* u_1 v_2^* u_2 \cdots v_t^* u_t$ avec $t \leq k$.

En effet, (3) implique manifestement (1). Ensuite, (2) implique (3) exactement comme à la question 7. Par ailleurs, la négation de (2) implique la négation de (1) dans le même esprit qu'en questions 3 et 4 : si l'on prend un chemin témoin passant par $k+1$ CFC cycliques avec au moins un tour de boucle de chaque, que l'on ajuste les parties intermédiaires pour identifier des tours entiers de boucles dans chaque CFC, et qu'on prend le PPCM des longueurs de boucle, alors on a la possibilité de déplacer les $k+1$ points intermédiaires entre CFCs cycliques en contrôlant le nombre de tours de chaque boucle, donc $\Omega(n^{k+1})$ possibilités.

D'où la caractérisation :

1. L est de densité $\Theta(n^k)$;
2. L est accepté par un automate où il n'y a pas de chemin acceptant qui passe par $k+1$ CFC cycliques (CFC qui ne sont pas un singleton sans boucle) mais il y en a un qui passe par k telles CFC ;
3. L est une union finie de langages de la forme $u_0 v_1^* u_1 v_2^* u_2 \cdots v_t^* u_t$ avec $t \leq k$, avec $t = k$ pour au moins un terme de l'union.

Noter le cas particulier $k = 0$ où la densité est donc bornée, il n'y a aucune CFC cyclique (donc le langage est bien fini), et L est une union finie de langages singletons.

Ceci implique en particulier que, parmi les langages épars, la densité doit être équivalente à un polynôme dont le degré est déterminable par la caractérisation ci-dessus, et notamment qu'aucun régime "intermédiaire" n'est possible.

Question 10. On a vu en question 6 comment tester en temps linéaire si un automate accepte ou non un langage épars (par la caractérisation des questions 4 et 7).

Si le langage est épars, une fois calculées les CFC, et étiquetées les CFC cycliques, le plus grand nombre de CFC cycliques traversées par un chemin acceptant se calcule de bas en haut en temps linéaire, ce qui permet de savoir en temps également linéaire où on se place dans les régimes de la question 9.

[Les résultats de ce sujet ont été originellement montrés dans [SYZS92], et peuvent également être trouvés dans [Pin19], Chapitre XII, Section 4.2]

Références

[Pin19] Jean-Éric Pin. Mathematical foundations of automata theory. `https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf`, 2019.

[SYZS92] Andrew Szilard, Sheng Yu, Kaizhong Zhang, and Jeffrey Shallit. Characterizing regular languages with polynomial densities. In *MFCS*, 1992. Non disponible en libre accès.

A5 – Ensembles inévitables

On fixe un alphabet fini $\Sigma = \{a, b\}$. Pour $w \in \Sigma^*$, on écrit $w = w_1 \cdots w_n$ où $n := |w|$ est la *longueur* de w. On dit qu'un mot $w \in \Sigma^*$ *évite* un mot $s \in \Sigma^*$ si s n'apparaît *pas* comme facteur de w. On dit que w *évite* un ensemble de mots $S \subseteq \Sigma^*$ s'il évite chaque mot de S.

Question 0. Donner un mot de longueur au moins 12 qui évite l'ensemble $S = \{aaaa, aaab, aba, baaa, bab, bbbb\}$.

Question 1. Donner le pseudocode d'un algorithme simple qui détermine, étant donné un mot $w \in \Sigma^*$ et un ensemble fini $S \subseteq \Sigma^*$, si w évite S. Analyser sa complexité en temps et en espace quand tous les mots de S font la même longueur.

On dit qu'un ensemble $S \subseteq \Sigma^*$ est *inévitable* s'il n'existe qu'un nombre fini de mots $w \in \Sigma^*$ qui évitent S. Sinon, il est dit *évitable*.

Question 2. L'ensemble de la question 0 est-il inévitable ? L'ensemble $\{aaa, abb, baa, abab\}$ est-il inévitable ?

Question 3. Montrer que l'ensemble $\{aaa, bab, baab, bbb\}$ est inévitable.

Question 4. Montrer le lemme suivant : pour tout alphabet fini Σ fixé et $k \in \mathbb{N}$, il existe $n \in \mathbb{N}$ tel que pour tout mot $w \in \Sigma^*$ tel que $|w| > n$, il existe deux entiers $p < q$ tels que pour tout $0 \leq l < k$, on ait $w_{p+l} = w_{q+l}$.

Question 5. Utiliser cette observation pour proposer un algorithme (pas nécessairement efficace) qui, étant donné un ensemble fini $S \subseteq \Sigma^*$, détermine si S est inévitable. Décrire sa complexité en temps et en espace.

Question 6. On dit qu'une expression rationnelle e est *inévitable* si l'ensemble (généralement infini) des mots du langage de e est inévitable. Donner un exemple d'expression rationnelle inévitable.

Question 7. L'expression rationnelle e est *bornée* s'il existe $n \in \mathbb{N}$ tel que tout mot satisfaisant e soit de longueur $\leq n$. Si e est bornée, comment peut-on déterminer si elle est inévitable ?

Question 8. Expliquer comment déterminer si une expression rationnelle e quelconque est inévitable.

Question 9. Pour e une expression rationnelle, on dit qu'un ensemble $S \subseteq \Sigma^*$ est *inévitable pour* e s'il existe un ensemble infini de mots du langage de e qui évitent S. Étant donné une expression rationnelle e et un ensemble fini S, expliquer comment déterminer si S est inévitable pour e.

Question 10. Montrer que, pour tout ensemble inévitable $S \subseteq \Sigma^*$, il existe un sous-ensemble $S' \subseteq S$ fini tel que S' soit inévitable.

Question 11. Étant donné une expression rationnelle e inévitable, expliquer comment calculer un sous-ensemble inévitable fini du langage de e.

Corrigé

Question 0. On peut prendre par exemple *aabbaabbaabb*.

Question 1. La principale difficulté est d'être rigoureux avec les indices.

```
Entrée:
  - mot w représenté comme un tableau de longueur n,
  - ensemble S représenté comme un tableau de longueur m de tableaux de
  longueurs nS[i] pour chaque i

Pour j de 0 à m exclu faire:
  Pour i de 0 à n-nS[j] exclu faire:
    ok := True
    Pour d de 0 à nS[j] exclu faire:
      Si w[i+d] != S[i][d]:
        ok := False
        Sortir de Pour
      Fin si
    Fin pour
    Si ok:
      // on a trouvé une occurrence de S[j] dans w
      Renvoyer faux
    Fin si
  Fin pour
Fin pour
Renvoyer vrai
```

Si les m mots de S font la même longueur l, la complexité en temps est en $O(n \times m \times l)$ et la complexité en espace est constante.

[On peut faire mieux en temps en utilisant l'algorithme de Knuth-Morris-Pratt (prétraitement en $O(m \times l)$ puis vérification en $O(n \times m)$) ou l'algorithme d'Aho-Corasick (prétraitement en $O(m \times l)$ puis vérification en $O(n)$), mais cela n'est pas demandé.]

Question 2. L'ensemble de la question 0 est évitable puisque, pour tout $i \in \mathbb{N}$, le mot $(aabb)^i$ l'évite.
L'ensemble $\{aaa, abb, baa, abab\}$ est évitable aussi : prendre b^i pour tout $i \in \mathbb{N}$.

Question 3. Procédons par l'absurde. Comme l'ensemble contient *aaa* et *bbb*, un mot suffisamment long évitant l'ensemble doit nécessairement comporter un b suivi d'un a, c'est-à-dire un facteur ba ; et si le mot est suffisamment long ce facteur est suivi d'autres caractères. Mais alors ce facteur ne peut être suivi de b (sinon on a un facteur *bab*), donc il est suivi de a, c'est-à-dire un facteur *baa*. Mais ce facteur ne peut être suivi de a (ce qui donnerait *aaa*), ni de b (ce qui donnerait *baab*), contradiction. Ainsi l'ensemble est-il bien inévitable.

Question 4. C'est une application immédiate du principe des tiroirs. On prend $n = |\Sigma|^k + k$. Ce mot a strictement plus de $|\Sigma|^k$ positions où commence un facteur de taille k, donc par application du principe des tiroirs il y a deux positions différentes où commence un facteur identique, ce qui est le résultat voulu.

Question 5. Soit k la longueur maximale d'un mot de S. Montrons tout d'abord que S n'est pas inévitable si et seulement s'il existe un mot évitant S avec deux occurrences distinctes d'un même facteur de longueur k. *(C'est une indication possible.)*

Dans le sens direct, si S est évitable alors il existe une infinité de mots évitant S, donc il existe des mots arbitrairement longs évitant S, en particulier des mots de longueur strictement supérieure au n de la question précédente. Ainsi, si on prend un tel mot, il a deux occurrences distinctes du même facteur de taille k.

Réciproquement, s'il existe un mot évitant S avec deux occurrences du même facteur de longueur k, alors on enlève un suffixe de ce mot pour que le mot se termine à la fin de la deuxième occurrence du facteur : le mot w_0 résultant évite toujours S. Soit $d > 0$ le décalage entre les deux occurrences. On peut ensuite construire des mots de plus en plus longs évitant S en ajoutant itérativement une copie de la lettre à d positions en partant de la fin : les mots résultants évitent tous S car ils ont un ensemble de facteurs de longueur k qui est le même que celui de w_0, qui évitait S, or la propriété d'éviter S est entièrement déterminée par les facteurs de longueur k par définition de k. Ainsi, S est évitable.

La caractérisation que nous avons démontrée implique que S est inévitable si et seulement si tous les mots de longueur $> n$ ont un facteur dans S, pour n la valeur de la question précédente. En effet, le sens réciproque est trivial (car il y a alors un nombre fini de mots évitant S), et pour le sens direct on vient de montrer que si S est inévitable alors tout mot ayant deux occurrences distinctes d'un même facteur de longueur k n'évite pas S, donc en particulier tous les mots de longueur $> n$.

Ceci suggère l'algorithme suivant : on énumère tous les mots de longueur $\leq n$, et on teste s'ils ont un facteur dans S. La complexité de cet algorithme est en $\left(\sum_{w \in S} |w|\right) \times |\Sigma|^k + k$ où k est la longueur maximale d'un mot de S, c'est-à-dire exponentiel en S. La complexité en espace demande de stocker le mot en cours, c'est-à-dire $|\Sigma|^k + k$.

En pratique il vaudrait mieux construire incrémentalement un mot par backtracking, en abandonnant si on contient un facteur dans S, et en réussissant si on a répété un facteur de longueur k déjà atteint.

Un bien meilleur algorithme (en $O(|\Sigma|^{k+1})$ pour k la longueur maximale d'un mot de S) est de considérer chaque mot de Σ^k et de construire un graphe orienté sur cet ensemble (avec $|\Sigma|^{k+1}$ arêtes) : on relie w à w' si et seulement s'il existe $a \in \Sigma$ tel que, si on écrit $w = a'w''$, on a $w' = w''a$. On élimine ensuite tous les sommets qui contiennent un mot de S. Il est clair que si ce graphe a un cycle alors on construit (comme dans la caractérisation précédente) un mot infini évitant S ; à l'inverse si on peut construire un tel mot alors comme à la caractérisation précédente un mot suffisamment long témoigne de l'existence d'un cycle dans ce graphe.

Question 6. Tout simplement $(aaa|bab|baab|bbb)^*$ d'après la question 3.

Question 7. Si une expression rationnelle est bornée, alors son langage est fini, et on peut simplement appliquer la question 5.

Question 8. On construit à partir de e l'expression rationnelle $\Sigma^* e \Sigma^*$ des mots qui ont un facteur dans e, i.e., ceux qui n'évitent pas e. On souhaite savoir si le complémentaire de ce langage est fini. Pour ce faire, on peut transformer e en un automate fini qui reconnaît le même langage avec les outils du cours, et complémenter cet automate (la clôture par complémentaire est également au programme). Il s'agit alors de déterminer si cet automate reconnaît un langage fini : après suppression des états inutiles, c'est le cas si et seulement s'il n'a pas de cycle.

On peut tester algorithmiquement tout cela, mais l'étape coûteuse est la complémentation : l'automate obtenu pour $\Sigma^* e \Sigma^*$ n'est généralement pas déterministe, et la clôture d'un automate implique de le complémenter d'abord, ce qui prend un temps exponentiel en général.

(Noter que cette construction généralise celle de la question 5.)

Question 9. On construit aisément l'expression rationnelle e_S des mots qui ont un facteur dans S, i.e., n'évitent pas S. On souhaite savoir si l'intersection de e avec le complémentaire de ce langage est finie. Pour ce faire, comme à la question 8, on construit à partit de e_S un automate A_S qu'on complémente en un automate A'_S (en le déterminisant au préalable), et on construit à partir de e un automate A. On teste alors si les langages de A et A'_S ont une intersection infinie en construisant un automate pour l'intersection (clôture par intersection au programme) et en testant comme à la question précédente.

Question 10. Soit $S \subseteq \Sigma^*$ inévitable. Soit n la longueur maximale d'un mot qui évite S. Ainsi, tout mot de longueur $n+1$ a un facteur dans S : soit S' l'ensemble de ces facteurs. C'est un ensemble fini, et il est inévitable puisque par définition tout mot de longueur $\geq n+1$ a un préfixe de longueur $n+1$ qui a un facteur dans S.

Question 11. En suivant la question précédente : à partir de l'automate construit à la question 8, on construit l'ensemble fini des mots dans le complémentaire L' du langage $\Sigma^* e \Sigma^*$ directement à partir de l'automate, on regarde leur longueur maximale n, puis on prend tous les mots de longueur $n+1$ et on leur trouve un facteur qui satisfait e. (En pratique il suffit de se limiter à un ensemble clos par préfixe des mots de longueur $n+1$, donc on peut notamment s'arrêter à tous les mots qui n'ont pas de continuation dans L', qu'on peut voir sur l'automate.)