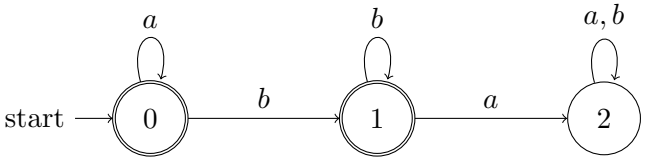
MITRO210: Automates et données structurées

Feuille d'exercices 5 – Corrigé

Antoine Amarilli

1 Quelques exemples sur le monoïde de transition

Question 0. L'automate est l'automate A_0 ci-dessous :



Question 1. La méthode à suivre, suivant [2] p 83–84, est de considérer les mots dans l'ordre radix $<$ (longueur, puis ordre lexicographique). On maintient les fonctions de Q dans Q atteintes, et des règles de récriture $u \to v$ quand un mot u nous donne le même élément qu'un mot plus petit v. Quand on considère un mot w, on teste d'abord si une règle de récriture $u \to v$ s'applique à un facteur du mot. Si c'est le cas, alors on sait qu'il n'est pas utile de considérer w : en effet, si l'on écrit $w = sut$ et $w' = svt$ le résultat obtenu en appliquant la règle, on sait que $w' < w$ a déjà été considéré et donne le même élément du monoïde de transition. Si ce n'est pas le cas, alors on calcule w, par exemple en composant des fonctions précédentes (si $w = az$ alors on compose les images de a et de z). On s'arrête quand les règles de récriture couvrent tous les mots à venir, par exemple si pour une longueur donnée tous les mots donnent le même résultat.

On obtient les éléments suivants :

- Mot vide : fonction identité
- Mot a : fonction $0 \mapsto 0$, $1 \mapsto 2$, $2 \mapsto 2$
- Mot b : fonction $0 \mapsto 1$, $1 \mapsto 1$, $2 \mapsto 2$
- Mot aa : même fonction que a, donc $aa \to a$
- Mot ab : fonction $0 \mapsto 1$, $1 \mapsto 2$, $2 \mapsto 2$
- Mot ba : fonction $0 \mapsto 2$, $1 \mapsto 2$, $2 \mapsto 2$
- Mot bb : même fonction que b, donc $bb \to b$
- Pour les mots de longueur 3, tous les mots avec un facteur aa ou bb sont couverts, donc il reste :
  - aba qui donne ba (c'est un zéro), donc $aba \to ba$
  - bab qui donne ba (c'est un zéro), donc $bab \to ba$
- Tous les mots de longueur 4 ou plus sont couverts par l'une des règles de récriture.

Les éléments du monoïde de transition sont donc ceux engendrés par ϵ, a, b, ab, et ba : on notera ce dernier 0 pour être clair. L'élément neutre est bien sûr ϵ.

Remarques utiles : 2 sera toujours envoyé vers 2 ; on a $aa = a$ et $bb = b$; on note que ba est un zéro.

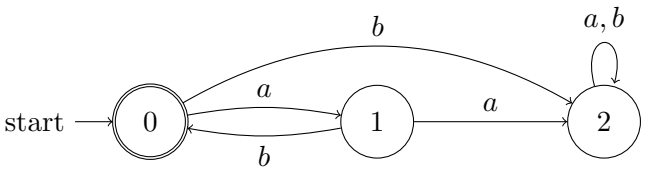
La table de multiplication (non commutative) est la suivante :

	ϵ	a	b	ab	0
ϵ	ϵ	a	b	ab	0
a	a	a	ab	ab	0
b	b	0	b	0	0
ab	ab	0	ab	0	0
0	0	0	0	0	0

Question 2. On peut écrire a^*b^* comme le langage des mots qui ne contiennent pas ba, c'est-à-dire $\bar{\emptyset}ba\bar{\emptyset}$. Ceci implique que le monoïde M_0 est apériodique, par le théorème de Schützenberger. On peut aussi voir directement que l'automate A_0 est sans cycle. Si on le vérifie sur les éléments du monoïde individuellement :

- Pour ϵ, on a $\epsilon = \epsilon^2$
- Pour a, on a $a = a^2$, et de même pour b
- Pour 0, on a $0 = 0^2$
- Pour ab, on a $(ab)^2 = 0$ mais on a bien $(ab)^3 = (ab)^2 = 0$

Question 3. On obtient l'automate A_3 suivant :

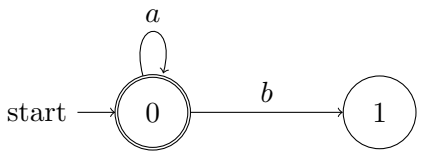


On calcule le monoïde de transition. Comme précédemment 2 est un puits donc inutile de préciser son image :

- Pour ϵ on a l'identité
- Pour a on a la fonction $0 \mapsto 1$ et $1 \mapsto 2$
- Pour b on a la fonction $0 \mapsto 2$ et $1 \mapsto 0$
- Pour aa on a la fonction envoyant tout vers 2, c'est un zéro
- Pour ab on a la fonction $0 \mapsto 0$ et $1 \mapsto 2$
- Pour ba on a la fonction $1 \mapsto 1$ et $0 \mapsto 2$
- Pour bb on a le zéro, ainsi $bb \to aa$
- Pour la longueur 3, tous les mots avec deux lettres contiguës sont le zéro dont se récrivent en aa, donc il ne reste que :
  - aba : on obtient la même chose que a, donc $aba \to a$
  - bab : on obtient la même chose que b, donc $bab \to b$
- Pour la longueur 4 tous les mots sont couverts par une règle de récriture donc on a fini.

Les éléments sont ceux engendrés par ϵ, a, b, ab, ba, et aa à nouveau noté 0. Le monoïde est apériodique car L_3 est sans étoile comme on l'a vu en cours.

Question 4. Pour a^* on a l'automate évident A_4 suivant :

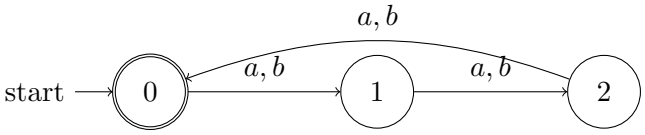


On obtient le monoïde M_4 défini comme suit, sans préciser l'image de 1 :

— Pour ϵ l'identité
— Pour a la fonction envoyant 0 vers 0
— Pour b la fonction envoyant 0 vers 1 (c'est un zéro)
— C'est toutes les fonctions possibles vu qu'on aura toujours $1 \mapsto 1$ donc on a fini

Pour $\Sigma^* b \Sigma^*$ l'automate est évident est le complémentaire de A_4 : on rend 0 non-final et 1 final. Le monoïde de transition est M_4. On observe ainsi que le monoïde de transition d'un langage ne dépend pas du statut final et non-final des états, ainsi un langage et son complémentaire ont le même monoïde syntaxique (à isomorphisme près). D'autres notions plus sophistiqués permettent d'associer aux langages des objets algébriques qui ne soient pas invariants par complémentation.

Question 5. On obtient l'automate A_5 suivant :



On calcule le monoïde M_5 :

— Pour ϵ, fonction identité
— Pour a, la fonction définie par $0 \mapsto 1$, $1 \mapsto 2$, $2 \mapsto 0$
— Pour b, sans surprise, la même fonction que a : ainsi $b \to a$. Ceci implique qu'il suffit de considérer des mots avec uniquement des a dans la suite.
— Pour aa, la fonction définie par $0 \mapsto 2$, $1 \mapsto 0$, $2 \mapsto 1$
— Pour aaa on retombe sur l'identité, donc $aaa \to \epsilon$ et on a fini.

Le monoïde M_5 est isomorphe à $\mathbb{Z}/3\mathbb{Z}$: c'est donc un groupe. Les *langages à groupe* sont ceux dont le monoïde syntaxique est un groupe : intuitivement, toute lettre (ou tout mot) est inversible. En particulier ces langages n'ont pas de puits acceptant ou rejetant (sauf s'ils sont triviaux).

Question 6. On peut construire un automate déterministe complet reconnaissant A reconnaissant F par une construction analogue à celle des tries (en n'oubliant pas d'y adjoindre un puits), mais ce n'est pas l'automate minimal. Celui-ci s'obtient en fusionnant itérativement les états q et q' tel que le langage reconnu à partir des états est le même : on obtient ainsi un automate acyclique.

On observe que les mots qui ne sont pas facteurs d'un mot de F vont envoyer n'importe quel état vers le puits. En effet, par l'absurde : si un mot $w \in \Sigma^*$ n'est pas facteur d'un mot de F et

envoie un état q vers un état q' qui n'est pas le puits, alors q n'est manifestement pas le puits non plus. Comme l'automate est émondé (à part pour le puits), il y a un mot u étiquetant un chemin de l'état initial à q, et un mot v étiquetant un chemin de l'état q' à un état final. Ainsi le mot uwv est accepté par l'automate donc il appartient à F et w est donc un facteur d'un mot de F, c'est impossible.

Ainsi, le monoïde de transition est engendré par les facteurs de F. Noter qu'il est possible que deux facteurs différents engendrent le même élément : par exemple, pour $F = \Sigma^k$ pour un k quelconque, tous les mots de la même longueur engendreront le même élément. C'est possible même pour des facteurs de longueurs différentes, par exemple pour $F = \{aa, bab\}$, alors aa et bab vont tous deux envoyer l'état initial sur l'état final et tous les autres états vers le puits.

On peut observer que le monoïde de transition a la propriété suivante, pour n la longueur maximale d'un mot de n : pour tout produit d'éléments $m_1 \cdots m_{n+1}$ où chaque m_i est différent de l'élément neutre, on a $m_1 \cdots m_{n+1} = 0$, avec le 0 défini en envoyant tout le monde vers le puits. On dit que le monoïde est *nilpotent* Évidemment c'est indispensable d'exclure le cas de l'élément neutre pour que la définition ait un intérêt. Noter que cette propriété n'est pas vérifiée par tous les monoïdes : par exemple M_4 ci-dessus la vérifie aussi (vu que l'unique élément non-neutre est le zéro), mais pas M_3 (on a $(ab)^2 = ab$ et $ab \neq \epsilon$ et $ab \neq 0$, donc pour tout $n \in \mathbb{N}$ on a $(ab)^n = ab \neq 0$), ni M_0 (on a $a^2 = a$ donc le raisonnement est similaire).

2 Quelques propriétés du monoïde de transition

Question 0. On note δ_w l'élément du monoïde de transition associé à un mot $w \in \Sigma^*$. On vérifie les propriétés (elles découlent immédiatement du fait que l'égalité est une relation d'équivalence) :

- Réflexivité : on a bien $\delta_w = \delta_w$ pour tout $w \in \Sigma^*$ ainsi $w \sim w$
- Symétrie : pour tous w, w' tels que $w \sim w'$ on a $\delta_w = \delta_{w'}$, on a évidemment $\delta_{w'} = \delta_w$ donc $w' \sim w$
- Transitivité : soient $u, v, w \in \Sigma^*$ tels que $u \sim v$ et $v \sim w$, on a donc $\delta_u = \delta_v$ et $\delta_v = \delta_w$, la transitivité de l'égalité nous donne $\delta_u = \delta_w$ ainsi $u \sim w$

Question 1. Soient $u, v \in \Sigma^*$ et supposons $u \in L$ et $u \sim v$. On sait comme $u \in F$ que δ_u envoie l'état initial q_0 vers un certain état final $q \in F$ ainsi $\delta_u(q_0) = q \in F$. Or $u \sim v$ donc $\delta_u = \delta_v$ ainsi $\delta_v(q_0) = q \in F$. Ainsi lire le mot v depuis l'état initial aboutit à un état final donc $v \in L$.

Question 2. Soient $x, z, y, y' \in \Sigma^*$ des mots et supposons $y \sim y'$ c'est-à-dire $\delta_y = \delta_{y'}$. On veut montrer $xyz \sim xy'z$, donc montrer l'égalité des fonctions δ_{xyz} et $\delta_{xy'z}$. Soit $q \in Q$ un état arbitraire. Après lecture de x on aboutit à un certain état $q' \in Q$ donné par $q' = \delta_x(q)$. Après lecture de y depuis q' on aboutit à un état $q'' \in Q$ donné par $q' = \delta_y(q')$. De même, après lecture de y depuis q' on aboutit au même état $q'' = \delta_y(q')$ puisqu'on sait que $\delta_y = \delta_{y'}$. Ensuite, après lecture de z depuis q'' on aboutit à un certain état $q''' \in Q$. Ainsi lire xyz et $xy'z$ depuis q aboutissent au même état q'''. Comme q est arbitraire, on en déduit que $\delta_{xyz} = \delta_{xy'z}$ et donc $xyz \sim xy'z$.

Question 3. On montre d'abord que $\equiv$ est une relation d'équivalence. Cette fois ça découle immédiatement du fait que l'équivalence logique est une relation d'équivalence :

- Réflexivité : manifestement pour tout mot $y \in \Sigma^*$ il est vrai que pour tous mots $x, z \in \Sigma^*$ on a $xyz \in L$ ssi $xyz \in L$, et donc $y \equiv y$

— Symétrie : pour tous mots $y, y' \in \Sigma^*$, si on suppose que $y \equiv y'$ c'est-à-dire que que pour tous mots $x, z \in \Sigma^*$ on a $xyz \in L$ ssi $xy'z \in L$, alors manifestement il est vrai aussi que pour tous mots $x, z \in \Sigma^*$ on a $xy'z \in L$ ssi $xyz \in L$, et ainsi $y' \equiv y$
— Transitivité : soit $y, y', y'' \in \Sigma^*$ trois mots et supposons que $y \equiv y'$ et $y' \equiv y''$. Montrons que $y \equiv y''$. Soit $x, z \in \Sigma^*$ deux mots, montrons la double implication. Si $xyz \in L$ alors on sait par $y \equiv y'$ que $xyz \in L$ implique $xy'z \in L$ et on sait par $y' \equiv y''$ que $xy'z \in L$ implique $xy''z \in L$, donc on a $xy''z \in L$. L'implication inverse est symétrique. On a donc bien que, pour tous mots $x, z \in \Sigma^*$, on a $xyz \in L$ ssi $xy''z \in L$. Donc on a bien $x \equiv z$.

On montre ensuite que $\equiv$ est une congruence. Soit $x, z \in \Sigma^*$ des mots arbitraires, et soit $y, y' \in \Sigma^*$ deux mots tels que $y \equiv y'$. Il faut montrer que $xyz \equiv xy'z$. Pour cela, montrons que, pour tous mots $x', z' \in \Sigma^*$, on a $x'(xyz)z' \in L$ ssi $x'(xy'z)z' \in L$. Mais comme $y \equiv y'$, en appliquant la définition de $\equiv$ aux mots $x'x$ et zz', on a que $(x'x)y(zz') \in L$ ssi $(x'x)y'(zz') \in L$. C'est précisément ce qu'il nous fallait, ce qui conclut.

Question 4. On montre d'abord que, pour tous mots $u, v \in \Sigma^*$, la relation $u \sim v$ implique $u \equiv v$. Soit $x, z \in \Sigma^*$ deux mots quelconques. On va considérer la lecture de xuz et xvz par l'automate A. Soit q_0 l'état initial de A. La lecture de x depuis q_0 nous mène à un certain état q. Maintenant, la lecture de u et de v depuis q nous mène au même état $q' = \delta_u(q) = \delta_v(q)$, puisque $u \sim v$ signifie que $\delta_u = \delta_v$. Ensuite, la lecture de z depuis q' nous mène à un certain état q''. Si q'' est final alors xuz et xvz sont tous deux acceptés, sinon ils sont tous deux rejetés. Ainsi, comme x et z étaient arbitraires, il est vrai que pour tous $x, z \in \Sigma^*$ on a $xuz \in L$ ssi $xvz \in L$, et on a donc établi $u \equiv v$.

Noter qu'on n'a pas utilisé la minimalité de l'automate pour établir cette direction : ceci montre en réalité que, pour tout automate A' reconnaissant le langage L, la relation d'équivalence définie par le monoïde de transition de A' raffine la relation d'équivalence syntaxique.

On montre à présent que, pour tous mots $u, v \in \Sigma^*$, la relation $u \equiv v$ implique $u \sim v$. On procède par contraposition et on montre que $u \not\sim v$ implique $u \not\equiv v$. Soit $u, v \in \Sigma^*$ deux mots tels que $u \not\sim v$, c'est-à-dire que les fonctions δ_u et δ_v sont des fonctions différentes. Ainsi, il y a un état $q \in Q$ de l'automate tel que $\delta_u(q) = q'$ et $\delta_v = q''$ avec $q' \neq q''$. Comme l'automate A est minimal, les états q' et q'' sont nécessairement distinguables, c'est-à-dire qu'il y a un mot $z \in \Sigma^*$ accepté depuis un seul des deux états. Quitte à échanger u et v, on suppose que z est accepté depuis q' mais rejeté depuis q''. Du reste, comme l'automate minimal A n'a pas d'états qui ne soient pas accessibles, il existe un chemin de l'état initial q_0 jusqu'à q : soit $x \in \Sigma^*$ un mot étiquetant un tel chemin. On sait à présent que le mot xuz est accepté par l'automate : la lecture de x mène de q_0 à q, celle de u mène de q à q', et celle de z mène de q' à un état final. En revanche le mot xvz est rejeté par l'automate : la lecture de x mène à nouveau de q_0 à q, celle de v mène de q à q'', et celle de z mène de q'' à un état non final. Ainsi $xuz \in L$ mais $xvz \notin L$. Ainsi, x et z témoignent du fait que $xuz \not\equiv xvz$, ce qu'il fallait démontrer.

On a donc démontré les deux implications et établi que $\sim$ et $\equiv$ sont bien les mêmes relations.

3 Requêtes d'appartenance de facteur

Question 0. Sans aucun précalcul, on peut répondre à une requête $[i, j[$ en temps $O(j - i)$, c'est-à-dire $O(n)$ en général, simplement en lisant le facteur concerné dans l'automate.

Question 1. Si on s'autorise un précalcul coûteux, on peut simplement précalculer un tableau T indexé par les couples (i, j) tels que $1 \leq i, j \leq n+1$, et écrire dans $T[i][j]$ un booléen indiquant

si on a $i \leq j$ et le facteur $a_i \cdots a_{j-1}$ est accepté par l'automate. Ce précalcul prend un temps $O(n^3)$ si on le réalise de façon naïve, et peut être ramené à $O(n^2)$ avec l'astuce classique de tester les facteurs en considérant les positions de départ successives.

Question 2. Dans l'exercice, on ne cherche pas à garantir de complexité efficace en fonction du langage L. On peut donc prendre l la longueur maximale d'un mot de L : c'est une valeur finie et supposée constante. On peut ensuite, pour chaque position $1 \leq i \leq l$, s'intéresser aux requêtes $[i, j[$ pour $1 \leq j \leq l$, et regarder lesquelles sont vraies et stocker cette information dans un tableau T de taille $n \times l$: ceci est en temps constant pour chaque position, donc le précalcul est en $O(n)$. Ensuite, le tableau T permet de répondre à toutes les requêtes en temps constant.

On peut chercher des solutions plus efficaces avec l'algorithme d'Aho-Corasick : on peut précalculer la structure de données de cet algorithme en $O(||L||)$, et il faut précalculer aussi pour chaque nœud du trie un tableau explicite de booléens de l Booléens, pour l la longueur maximale d'un mot de L, des longueurs des suffixes qui sont dans le langage (c'est-à-dire suivre les pointeurs raccourcis), ceci étant majorable grossièrement par $O(||L|| \times l)$. Ensuite, pour chaque position de fin dans le mot on indique à quel nœud du trie on correspond : ceci est en temps $O(n)$ sans facteur dans le langage. Pour répondre à une requête on peut maintenant assurer $O(1)$: on consulte le tableau de Booléens pour la position de fin. On a donc une réponse en requêtes en temps constant indépendant de l'automate et de n, et un précalcul en $O(||L|| \times l + n)$.

Question 3. Bien sûr, une séquence de nœuds qui convienne sont les feuilles concernées, qui sont du reste faciles à trouver : mais elles sont trop nombreuses. On peut alors avoir l'idée intuitive d'itérer l'opération suivante : tant qu'on a un nœud dont les deux enfants sont pris, alors remplacer les deux enfants par ce nœud, ce qui ne change pas l'intervalle représenté et fait diminuer le nombre de nœuds. On peut alors avoir l'intuition que, in fine, on aura un nombre constant de nœuds à chaque hauteur. La construction qui suit vise à établir ce fait.

On remarque d'abord que, dans notre construction de l'arbre T, pour deux nœuds n et n' quelconques, les intervalles ι_n et $\iota_{n'}$ sont soit disjoints si n et n' sont incomparables dans l'arbre, soit inclus l'un dans l'autre si n et n' sont comparables (plus précisément $\iota_n \subseteq \iota_{n'}$ si n est descendant de n' et vice-versa). Ainsi, les nœuds que l'on cherche sont une antichaîne dans l'arbre, c'est-à-dire un ensemble de nœuds deux à deux incomparables.

On propose alors la construction suivante. Cherchons les valeurs l et r qui sont les premières strictement à gauche et strictement à droite de l'intervalle : c'est-à-dire $l = i - 1$ et $r = j$. Quitte à rajouter des symboles $\bullet$ autour du mot, on peut garantir que $1 \leq l \leq r \leq |w|$. Identifions l et r aux feuilles correspondant à ces positions, et considérons les chemins C_l et C_r remontant de ces feuilles jusqu'à la racine : ils s'intersectent en un nœud interne n qui est le plus petit ancêtre commun de l et r, manifestement n est un nœud interne dont l'enfant gauche est dans C_l et dont l'enfant droit est dans C_r. Appelons C'_l et C'_r les préfixes de C_l et C_r qui sont strictement en dessous du nœud n. On choisit comme nœuds de notre séquence les enfants droits des nœuds de C'_l qui ne sont pas dans C'_l (dans l'ordre de bas en haut), et les enfants gauches des nœuds de C'_r qui ne sont pas dans C'_r (dans l'ordre de haut en bas). Ces nœuds sont en nombre en plus $2l$ c'est à dire $2 \log_2 n$, et on les trouve en temps logarithmique en remontant simultanément depuis l et r jusqu'à tomber sur le même nœud.

Il est manifeste que les intervalles des nœuds retenus sont deux à deux disjoints et ordonnés de gauche à droite la bonne manière, et qu'ils sont inclus dans l'intervalle désiré vu qu'ils sont à droite de l et à gauche de r. Par ailleurs, si on prend toute feuille de l'intervalle désiré, en remontant depuis cette feuille on préserve l'invariant d'être à chaque profondeur entre le nœud de C'_l et le nœud de C'_r à cette profondeur. Forcément on rejoint ainsi C'_l ou C'_r en remontant

puisque C_l et C_r se rejoignent en n, et forcément c'est avant n car les deux enfants de n sont dans $C'_l \cup C'_r$. Si l'on rejoint un nœud de C'_l c'est forcément comme un enfant droit n' d'un nœud de C'_l, et de même si on rejoint un nœud de C'_r c'est forcément comme un enfant gauche n', en tout cas n' fait partie de la séquence et couvre donc la feuille de laquelle on est parti. Ainsi la construction est correcte.

Question 4. Le précalcul est le même que celui vu en cours, en temps linéaire en le mot à automate fixé.

Étant donné une requête d'appartenance de facteur sur l'intervalle $[i, j[$, on calcule la séquence de nœuds $n_1, \ldots, n_k$ de la question précédente, en $O(\log n)$. On a dans le précalcul stocké les éléments τ_n du monoïde de transition sur chaque nœud. Or, comme on sait que $[i, j[$ est la concaténation des $\iota_{n_1}, \ldots, \iota_{n_k}$ par définition, alors l'élément du monoïde de transition correspondant à l'intervalle $[i, j[$ est le produit des $\tau_{n_1} \cdots \tau_{n_k}$. Ce calcul est en temps $O(\log n)$ à automate fixé, et il nous dit en particulier si le facteur appartient ou non au langage.

Question 5 (bonus). La construction que nous allons présenter ici effectue un précalcul en $O(k \times n)$, où $k := |Q|$, soit linéaire en le mot à automate fixé. Et elle permet de répondre aux requêtes en temps $O(k)$, c'est-à-dire à nouveau en temps constant à automate fixé. Cette construction est tirée de [1] section 2.2. En revanche, à la différence de la construction précédente, cette construction a l'inconvénient de ne pas pouvoir être efficacement mise à jour, a priori, si le mot est modifié.

Pour cette construction, on construit le graphe produit G de l'automate et du mot, c'est-à-dire un graphe dont les sommets sont les paires $Q \times \{0, \ldots, n\}$ qu'on appellera des *configurations*, et avec une arête de (q, i) à $(q', i+1)$ ssi on a $q' = \delta(q, a_{i+1})$. Le graphe G se construit en temps $O(k| \times n)$. On va ensuite couvrir les sommets de ce graphe par des *chaînes*. Une chaîne est simplement un chemin G : elle a un sommet de départ et un sommet de fin, et on lui donne également un *rang* compris entre 1 et k. On va construire une collection de chaînes qui garantisse que chaque configuration apparaisse dans une et une seule chaîne, c'est-à-dire qu'on maintient un tableau M de taille $O(k \times n)$ qui indique pour chaque configuration à quelle chaîne elle appartient. On imposera les conditions suivantes :

— Pour chaque position $0 \leq i \leq n$, si on appelle $C_1, \ldots, C_k$ les chaînes qui contiennent les configurations de la forme $\{(q, i) \mid q \in Q\}$, alors les rangs de ces chaînes sont deux à deux distincts. Autrement dit, ces chaînes portent tous les rangs $\{1, \ldots, k\}$, exactement une fois pour chaque rang.
— Pour chaque configuration (q, i) avec $i < n$, pour $(q', i+1)$ son successeur avec $q' = \delta(q, a_{i+1})$, si on appelle C et C' les chaînes qui contiennent (q, i) et $(q', i+1)$ respectivement, alors soit $C = C'$, soit C' est de rang inférieur à C'.

Autrement dit, à chaque position i, les configurations correspondantes de G sont couvertes par k chaînes de chacun des k rangs différents. Chaque chaîne se poursuit ensuite aussi longtemps que possible, mais il se peut que deux chaînes entrent en conflit, c'est-à-dire que leur état successeur à la position suivante soit le même. Dans ce cas, c'est seulement la chaîne de plus petit rang qui continue sur ce sommet : les autres chaînes s'arrêtent. Lorsque l chaînes se terminent à la position i, cela doit impliquer qu'il y a l configurations à la position $i+1$ qui n'ont pas de prédécesseur : on fait alors commencer de nouvelles chaînes à ces configurations, en leur faisant porter les rangs qui sont devenus libres, c'est-à-dire les rangs des chaînes qui se sont terminées à la position i.

Pour le dire encore d'une autre manière : il y a une seule chaîne de rang 1, qui commence en position 0 sur un certain état q et se poursuit jusqu'à la position n, en suivant le run sur le mot entier qui commence en q. Pour le rang 2, il y a une chaîne qui commence en position 0 et qui se poursuit jusqu'à rencontrer la fin du mot ou la chaîne de rang 1 : dans ce deuxième cas une autre chaîne de rang 2 recommence juste après à un état qui n'est pas encore couvert, et elle se poursuit de la même manière jusqu'à la fin du mot ou jusqu'à rencontrer la chaîne de rang 1 ; et ainsi de suite. Pour les rangs 3 et suivants, la construction est analogue.

On prétend qu'on peut construire toutes ces chaînes en temps linéaire en le graphe produit. On va supposer que les chaînes sont numérotées par des entiers appelés leur identifiant (il y en aura au plus $k \times (n+1)$ à savoir le nombre de sommets de G) et que les informations à leur sujet sont stockées dans un tableau T indexé par les identifiants de chaînes. Pour une chaîne donnée on stocke deux informations : son rang (initialisé à la création de la chaîne), et la position à laquelle elle s'arrête (qu'on stockera au cours de la construction).

La construction procède de manière inductive. Supposons qu'on a construit des chaînes et que tous les sommets de G jusqu'à la position i sont couvertes. L'initialisation pour $i = 0$ est bien sûr qu'on crée k chaînes avec les k rangs différents pour stocker chaque état. Pour la récurrence, quand on considère la lettre a_{i+1}, on peut étendre les chaînes qui doivent l'être (et le refléter dans le tableau M), en utilisant les rangs pour résoudre des conflits suivant ce que l'on a expliqué. On peut ensuite préparer une liste des rangs des chaînes qui viennent de se terminer, noter dans T leur position de fin, et démarrer de nouvelles chaînes avec les rangs devenus libres sur les configurations $(q, i+1)$ pour lesquelles $M[q, i+1]$ n'est pas défini. À la fin du mot, on note simplement dans T que toutes les chaînes non encore terminées sont à présent terminées. Tout ceci prend un temps $O(k)$ à chaque position, donc un temps $O(k \times n)$ au total.

On va également précalculer un tableau auxiliaire M' de taille $k \times (n+1)$ tel que, pour chaque position $0 \leq i \leq n$ et chaque rang $1 \leq j \leq k$, le tableau $M'[j, i]$ nous donne l'unique état q telle que la configuration (q, i) soit couverte par une chaîne de rang j. Comme les rangs à la position i sont tous distincts, on sait en effet qu'il y a une unique telle chaîne. Ce tableau se calcule directement à partir de M : pour chaque configuration (q, i) on consulte $M[q, i]$ pour trouver le rang j de la chaîne qui couvre (q, i) et on initialise $M'[j, i] := q$. Ceci s'effectue aussi en temps $O(k \times n)$.

On explique à présent comment évaluer les requêtes. Étant donné des positions i, j, on va vouloir simuler l'exécution de l'automate A sur le mot $a_i \cdots a_j$ à partir de l'état q_0, mais on va accélérer cette exécution en utilisant les chaînes précalculées. Au départ, l'automate se situe à l'état initial q_0 à la position $i-1$, c'est-à-dire sur la configuration $(q_0, i-1)$: celui-ci appartient à une certaine chaîne C_0 indiquée dans $M[q_0, i-1]$, et on sait grâce au tableau T à quelle position j_1 cette chaîne se finit. De deux choses l'une : soit $j_1 \geq j$, soit $j_1 < j$. Dans le premier cas, pour retrouver l'état de l'automate à la position j, il suffit de suivre la chaîne C_0, qui est ininterrompue jusque là. Mais on peut simplement le faire en allant à la position j, en regardant les configurations de la forme (q, j) pour chaque état q, et en consultant les valeurs $M[q, j]$ pour retrouver quelle case contient C_0 : ceci peut se faire avec l'identifiant de la chaîne C_0. Mais en réalité on sait que C_0 est identifiée de façon unique par son rang, donc on peut aussi retrouver l'état q en consultant $M'[j', j]$ pour j' le rang de C_0, ce qui s'effectue en temps constant. En résumé, on retrouve ainsi en temps constant l'état q auquel l'automate aboutit en position j, et selon que q est final ou non, on sait si le facteur $a_i \cdots a_j$ est accepté par l'automate ou non.

Dans le second cas, la chaîne C_0 se termine à une position $j'_0 < j$ qui nous est donnée par le tableau T. On peut alors retrouver par la même technique qu'au premier cas, en temps constant,

l'état q'_0 tel que la lecture du mot $a_i \cdots a_{j_1}$ depuis q_0 nous conduise à q'_0 à la position j'_0, c'est-à-dire qu'on atteint la configuration (q'_0, j'_0). On suit alors manuellement la transition $q_1 = \delta(a_{j'_0}, q'_0)$ pour trouver le prochain état à la prochaine position, c'est-à-dire qu'on arrive ensuite à la configuration (q_1, j_1) avec $j_1 := j'_0 + 1$. Comme la chaîne C_0 aurait dû continuer à cette configuration mais s'est terminée, cela implique que la chaîne C_1 qui commence à cette configuration (et qu'on retrouve avec le tableau M) est de rang plus petit que C_0 ne l'était.

On peut alors distinguer à nouveau deux cas selon que $T[C_1]$ est plus petit que j ou non ; et on peut répéter l'argument. On, on observe qu'on ne répétera l'argument que k fois au plus, car le rang décroît à chaque étape. Formellement, si on considère la séquence $C_0, C_1, \ldots$ des configurations successivement obtenues, alors le rang de ces configurations est strictement décroissant. Ainsi, il y a au plus k telles configurations. Chaque application de l'argument est en temps constant : il s'agit simplement de consulter le tableau T, de consulter le tableau M' si nécessaire, et de suivre une transition de l'automate. La complexité d'évaluer une requête est donc bien en $O(k)$, comme nous l'avions affirmé. Ceci conclut la présentation de l'algorithme.

Références

[1] M. Bojańczyk. Factorization forests. In *International Conference on Developments in Language Theory*, 2009.

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