MITRO210 : Automates et données structurées

Feuille d'exercices 2 – Corrigé

Antoine Amarilli

Avertissement : ce corrigé n'a été que peu relu. N'hésitez pas à me contacter pour toute question ou erreur.

1 Automates alternants

Question 0. On retombe sur la définition habituelle d'un NFA.

Question 1. Si le mot w est vide, on vérifie juste si l'état initial est final. Sinon, on suit la définition : en notant $w = av$, pour chaque état vers lequel l'état initial a une transition notée a, on vérifie si v est accepté (avec l'algorithme habituel pour les NFA), et on renvoie vrai si c'est le cas pour chacun des états.

Question 2. On explique comment construire l'automate A'. On a un état initial q'_0 dans A' correspondant à l'état initial q_0 de A, et on met q'_0 final dans A' si et seulement si q_0 est final dans A. Ensuite, pour chaque lettre a, on a un état de A' avec une transition a dans A' depuis q'_0, correspondant à l'ensemble des états de A vers lesquels q_0 a une transition a dans A. Intuitivement, on veut accepter si le mot à venir est accepté depuis tous les états de l'ensemble en question.

Étant donné un état S de l'automate A' correspondant à un ensemble d'états de A, quand on lit une lettre a, on va intuitivement utiliser le nondéterminisme de A' pour deviner vers quel état transitionner. Plus précisément, s'il y a un état de S qui n'a pas de transition sortante étiquetée a dans A, alors il n'y a pas de transition étiquetée a depuis S. Sinon, on choisit pour chaque état de S un successeur par a et on va vers l'ensemble de ces successeurs, formant un nouvel ensemble (dont la cardinalité est au plus celle de S, potentiellement moins si certains de ces sucesseurs sont identiques).

Les états de A' qui sont finaux, outre potentiellement q'_0, sont les états correspondant à un ensemble S d'états de A qui sont tous finaux.

On se convainc sans peine par induction sur le mot que la lecture d'un mot av nous conduit précisément aux ensembles d'états S tels que, si on note S_a les a-successeurs de q_0 dans A, alors en lisant v depuis chaque état de S_a simultanément on peut arriver à l'ensemble d'états S. Ceci justifie que la construction est correcte, c'est-à-dire que A' reconnaît bien le même langage que A.

L'automate A' obtenu est un NFA, et son nombre d'états est exponentiel en A.

Question 3. On généralise l'idée de la construction précédente. Les états du nouvel automate A' sont des ensembles d'états de l'automate A. Pour I l'ensemble des états initiaux de A, on a un état initial $\{q_0\}$ dans A pour chaque $q_0 \in I$. On met comme finaux dans A' les ensembles S où chaque état est final dans A. Ensuite, quand on lit une lettre a à partir de l'état S de A', on peut aller à chaque ensemble S' obtenu comme suit :

— Pour chaque état q de S qui est universel, on met tous les a-successeurs de q dans S' ;
— Pour chaque état q de S qui est existentiel, on choisit un a-successeur de q pour le mettre dans S' ; en particulier si q n'a aucun a-successeur alors il n'y a pas de a-successeur S' de S.

On démontre immédiatement par induction que, pour chaque mot v, les ensembles S qu'on peut obtenir sont précisément les ensembles d'états qu'on peut atteindre depuis un état initial en lisant v et en choisissant nondéterministement un successeur de chaque état existentiel et tous les successeurs de chaque état universel pour chaque lettre.

Ainsi, on obtient un NFA, et la construction est là encore exponentielle en A.

2 Recherche de sous-mots non-contigus

Question 0. On peut simplement calculer tous les sous-mots de v (il y en a $2^{|v|}$ au maximum, qui ne sont pas forcément tous différents) et voir si on obtient u. La complexité est de $O(|v|2^{|v|})$. (Noter qu'on suppose que u est de longueur au plus v, sinon on rejette immédiatement ; ainsi la complexité peut être exprimée indépendamment de $|u|$.)

Question 1. Il est clair qu'un algorithme glouton fonctionne : pour chaque lettre successive de u, on cherche sa prochaine occurrence dans v, et on voit si on parvient ainsi à lire tout u.

Pour se convaincre de la correction, l'invariant est que quand on a lu les i premiers caractères de u avec $0 \leq i \leq |u|$, alors la position courante est la longueur du plus petit préfixe de v qui admette le préfixe de longueur i de u comme sous-mot.

La complexité de cet algorithme est de $O(|v|)$.

Une autre façon de le voir est qu'on peut construire un NFA en temps $O(|u|)$ qui reconnaisse les mots dont u est sous-mot : on a des boucles sur chaque lettre et on peut avancer quand on lit la bonne lettre, le premier état est initial et le dernier est final. On se rend ensuite compte que, pour qu'un mot soit accepté, on a toujours envie d'avancer (sauf sur le dernier état), ainsi quand on a une transition non-boucle étiquetée a alors la boucle étiquetée par a est redondante. Ainsi, on obtient en $O(|u|)$ un DFA reconnaissant les mots dont A est sous-mot. Il n'y a plus qu'à lire v avec cet automate pour conclure en $O(|v|)$.

Question 2. Il suffit de calculer, pour chaque position i de v, pour chaque caractère a, l'emplacement du prochain a à partir de i dans v (s'il existe).

Avec cette structure de données, on peut manifestement exécuter l'algorithme de la question précédente en temps $O(|u|)$ étant donné le mot u.

Pour calculer la structure de données, on traite le mot v de droite à gauche en mémorisant la position de la dernière occurrence de chaque lettre qui a été vue. Ceci donne les bonnes informations, en temps $O(|v| \times |\Sigma|)$.

Question 3. En réalité, la question précédente nous donne directement le DFA demandé. Cet automate est également décrit dans [2].

Question 4. On construit un DFA pour cette tâche simplement par l'intersection des deux automates pour u et pour v. La complexité est $O(|u| \times |v| \times |\Sigma|)$.

Question 5. On dispose d'un DFA et on veut savoir quel est le mot le plus long que ce DFA accepte. (Noter que ce DFA est manifestement sans cycle, du reste le langage des mots qui sont facteurs d'un mot donné est bien sûr un langage fini.) On peut simplement calculer la plus grande longueur possible (et un mot témoin) de proche en proche dans l'automate, en complexité linéaire en l'automate.

Des idées similaires sont dans [1].

On peut alternativement voir cet algorithme comme un algorithme de programmation dynamique, c'est même la façon la plus classique de résoudre le problème longest common subsequence.

Question 6. On peut manifestement généraliser l'algorithme précédent à une complexité de $O(\prod_i |u_i| \times |\Sigma|)$.

Question 7. Dans ce cas le problème devient trivial : il suffit de considérer tous les sous-mots de chaque mot d'entrée (il y en a au plus 2^ℓ) et intersecter explicitement ces ensembles. La complexité ainsi obtenue est de l'ordre de $O(n2^\ell)$, elle est linéaire en l'entrée !

Références

[1] A. Conte, R. Grossi, G. Punzi, and T. Uno. A compact DAG for storing and searching maximal common subsequences, 2023. arXiv :2307.13695.

[2] J.-J. Hebrard and M. Crochemore. Calcul de la distance par les sous-mots. *RAIRO-Theoretical Informatics and Applications*, 20(4) :441–456, 1986.