Info Spés **DS 1 - corrigé**

Langages et automates

1 Codes

1. • $L_1 = \{a, b\}$ est clairement un code : si $w \in L_1^+$, il n'existe qu'une seule façon de le décomposer en produit de ses lettres !
   • $L_2 = ab^*$ est un code : dans une décomposition, le k-ième mot est délimité par le k-ième a (compris) et le $(k+1)$-ième a (non compris).
   • $L_3 = \{a, ab, ba\}$ n'est pas un code : $aba = ab.a = a.ba$.
   • $L_4 = \{a, ba, bb\}$ est un code : un contre-exemple (mot de L_3^+ se décomposant de deux façons différentes selon L_1) minimal commence nécessairement par b (sinon, les décompositions commencent toutes par le mot a, et ce qui suit est un plus petit contre-exemple). Maintenant, si la seconde lettre est a (resp. b), le premier mot intervenant dans toute décomposition ne peut être que ba (resp. bb), et on n'a donc pas le choix du premier mot de la décomposition : ce qui suit est donc un contre-exemple strictement plus petit : absurde.
   • $L_5 = \{aa, baa, ba\}$ est un code : avec des arguments vus plus haut, un contre-exemple minimal commence nécessairement par b, puis baa. Mais alors, la décomposition qui commence par ba se poursuit par aa.Mais alors, la décomposition qui commence par baa doit se poursuivre par aa, mais alors...
   • $L_6 = \{a, abbba, babab, bb\}$ n'est pas un code : la recherche d'un contre-exemple minimal conduit à : $a.bb.babab.abbba = abbba.babab.bb.a$.
   • $L_7 = a^+b^+$ est un code : le premier mot d'une décomposition se termine obligatoirement à la fin de la première séquence de b. Pareil pour le second....
   • $L_8 = a^+b^*$ n'est pas un code puisque $a^2 = a.a \in L^2 \cap L$.
   • $L_9 = \{a^n b^n \mid n \in \mathbb{N}^*\}$ est un code : même argument que pour a^+b^+,ou tout simplement : tout partie d'un code est un code (clair).
2. $\varepsilon^2 = \varepsilon^3$ nous fournit deux décompositions de ε, donc $\{\varepsilon\}$ n'est pas un code. Réciproquement, si $w \neq \varepsilon$, les mots de w^+ sont les puissances de la forme w^p qui ne peut pas se décomposer autrement qu'en produit de p mots w (p est imposé par la longueur du mot).
3. Supposons que X ne soit pas un code : il existe des mots de X^+ se décomposant de deux façons différente selon X. Parmi ces mots, choisissons-en un de taille minimale : il s'écrit $w = w_1...w_k = w'_1...w'_l$, avec les w_i et w'_i dans X. Supposons par exemple : $|w_1| \leq |w'_1|$: on a alors w_1 qui est préfixe de w donc de w'_1, donc $w_1 = w'_1$ d'après l'hypothèse sur X. Ainsi, $w' = w_2...w_k = w'_2...w'_l$ se décompose de deux façons différentes selon X et est de longueur $< |w|$ (car $\varepsilon \notin X$), niant la minimalité de $|w|$: absurde.
4. Même chose en remplaçant "préfixe" par "suffixe" et (w_1, w'_1) par (w_k, w'_l) !!!
5. $a \Rightarrow b$ n'est pas trop compliqué (contraposée) et $b \Rightarrow c$ non plus (idem) !

Pour $c \Rightarrow a$, on va raisonner par l'absurde, en supposant le résultat faux, et en considérant un contre-exemple (w, w'), avec par exemple $|w| + |w'|$ minimale : il existe deux décompositions distinctes d'un même mot :

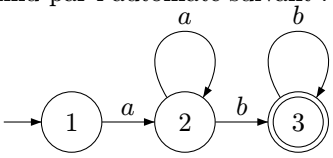
$$m = w_1...w_k = w'_1...w'_l,$$

avec les w_i et w'_i dans $\{w, w'\}$. Par minimalité du contre-exemple, on a $w_1 \neq w'_1$, donc par exemple $w_1 = w$ et $w'_1 = w'$. On peut supposer : $|w_1| \leq |w'_1|$, si bien que w est préfixe de w'. On peut donc écrire $w' = ww''$. Maintenant, w et w'' ne sont pas puissances d'un même mot (sans quoi w et w' le seraient), donc $\{w, w''\}$ est un code (par minimalité du contre-exemple $\{w, w'\}$). En tronquant w de m, on obtient :

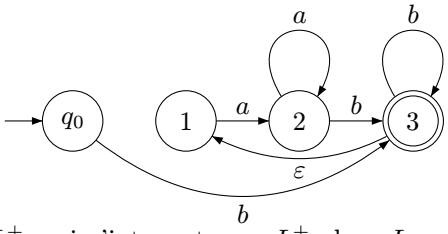
$$m' = w_2...w_k = w''w'_2...w'_l.$$

Si $w_2 = w$, alors m' a deux décompositions dans $\{w, w''\}$ qui est un code : c'est impossible. Ainsi, $w_2 = w' = ww''$, et on obtient à nouveau deux décompositions différentes de m' selon $\{w, w''\}$, fournissant la contradiction souhaitée.

6. On construit un automate déterministe complet reconnaissant L. De façon classique, les mots de L^+ seront reconnus par l'automate $\mathcal{A}'$ construit à partir de $\mathcal{A}$ en ε-transitionnant les acceptants de $\mathcal{A}$ vers i.
Considérons deux décompositions différentes d'un même mot w de L^+, avec $|w|$ minimale :
$$w = w_1...w_k = w'_1...w'_l.$$
Par minimalité de $|w|$, $w_1 \neq w'_1$, donc par exemple $|w_1| < |w'_1|$. Notons q l'état de $\mathcal{A}$ dans lequel on arrive en lisant w_1 depuis q_i : le mot $w_2...w_k$ est l'étiquette d'un chemin de $\mathcal{A}'$ issu de q arrivant à un acceptant et dont la première transition n'est pas une ε-transition : notons L'_q l'ensemble de ces mots.
Si L n'est pas un code, L^+ intersecte donc l'un des L'_q,pour q final. Réciproquement, si $w \in L^+ \cap L'_q$ pour un certain q final, alors on notant w_1 un mot liant q_i à q dans $\mathcal{A}$, on a $w_1 w$ qui se décompose de deux façons différentes dans L^+ (l'une commence par w_1, l'autre par $w_1 w'$, où w' est un préfixe non vide de w reliant q à un état final de $\mathcal{A}'$), de sorte que L n'est pas un code.
Il reste à voir que L'_q est reconnu par l'automate $\mathcal{A}''$ "copie de $\mathcal{A}'$" auquel on ajoute un état initial q_0, avec pour tout $\alpha \in A$: $\delta(q_0, \alpha) = \delta(q, \alpha)$ (ceci pour interdire une ε-transition au début du chemin). Enfin, le caractère vide de l'intersection de deux langages rationels est décidable facilement après déterminisation des automates reconnaissant l'un et l'autre, et en considérant l'automate produit.
Passons à la pratique :
- $L_7 = a^+b^+$ est reconnu par l'automate suivant :

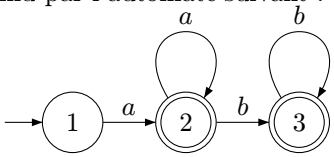


L'_3 est reconnu par :

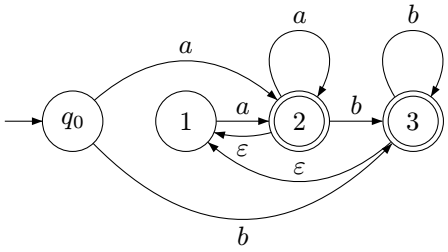


donc $L'_3 = b^+L_7^+$, qui n'intersecte pas L_7^+, donc L_7 est un code.

- $L_8 = a^+b^*$ est reconnu par l'automate suivant :



$L'_3 = b^+L_8^+$, n'intersecte toujours pas L_8^+. Par contre, L'_2 est reconnu par :



donc $w = a \in L'_2 \cap L_8^+$, et L_8 n'est pas un code (avec les notations de la preuve, on voit que $w_1 = a$ relie 1 à 2, ce qui nous fournit le mot $a^2 = a.a$ qui se décompose de deux façons différentes dans L^+).

7. Il faut d'abord construire un automate reconnaissant L : en autorisant les ε-transitions, cela génère un automate avec $N = O(|e|)$ états, avec e une expression rationnelle reconnaissant L et $|e|$ la longueur de e "en tout sens raisonnable". Le temps nécessaire à cette construction sera polynomial en N (sans entrer dans les détails, vu ce qui suit !)

   Notons au passage que sans les ε-transitions, la construction naturelle d'un automate reconnaissant $e_1 + e_2$ fait intervenir un automate-produit, ce qui va tuer le caractère linéaire du nombre d'états. L'introduction des ε-transitions apporte donc un plus au niveau de la taille du résultat : les grognons diront qu'il va falloir déterminiser, mais de toute façon, on perd également le caractère déterministe dans la construction de l'étoile ou de la concaténation en l'absence d'ε-transitions.

   Ensuite, on déterminise l'automate, soit un coût en $O(N^{10}2^N)$ pour voir large : le déterminisé $\mathcal{A}_d$ a $N_1 = O(2^N)$ états.

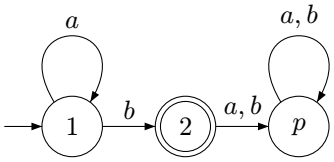
   Pour chacun des L'_q, on doit calculer l'intersection de $\mathcal{L}(\mathcal{A}_d)$ avec $\mathcal{L}(\mathcal{A}_q)$, ce qui demande encore une déterminisation : $O(2^{N_1})$ états à l'arrivée. L'automate produit pour calculer l'intersection a $O(4^{N_1})$ états !

   Ainsi, on obtient une complexité en temps de l'ordre de $2^{2^{O(|e|)}}$, ce qui fait vraiment beaucoup, même si c'est un majorant peut-être un peu grossier !!!

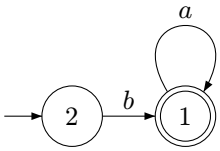
   Un bon sujet de TIPE pourrait être d'implémenter effectivement cet algorithme : je n'ai trouvé aucune référence sur ce problème de décidabilité. Il est peut-être archi connu ainsi qu'une réponse beaucoup plus simple que la mienne, mais ce n'est pas certain.

2 Quelques constructions

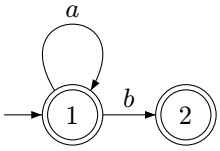
1. Sans trop de mal :



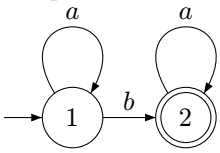
2. Le miroir des mots de la forme $a^n b$, ce sont les mots de la forme ba^n, donc $\overline{L_1} = ba^*$, qui est reconnu par :



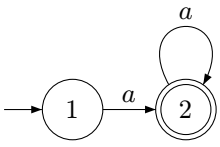
3. Les résultats sont donnés sans justification, mais c'est à peu près clair (faire tout de même les deux inclusions pour s'en convaincre...)
   - $Init(L_1) = a^* + a^*b$, reconnu par :



   - $Min(L_1) = Max(L_1) = L_1$.
   - $Cycle(L_1) = a^*ba^*$, reconnu par



   - $\frac{1}{2}L_1 = \frac{2}{3}L_1 = a^+$, reconnu par



4. Rebelote :
   - $\overline{L_2} = \{b^n a^n \mid n \in \mathbb{N}\}$.
   - $Init(L_2) = \{a^n b^m \mid 0 \leq m \leq n\}$.
   - $Min(L_2) = Max(L_2) = L_2$.
   - $Cycle(L_2) = \{a^i b^n a^j, b^i a^n b^j \mid i + j = n\}$.
   - $\frac{1}{2}L_2 = b^*$.
   - $\frac{2}{3}L_2 = L_2$.

   De ces langages, seul b^* est rationnel : pourquoi ?

5. D'après le théorème de Kleene, il suffit de montrer que la classe des langages reconnaissables est stable par les différentes fonctions. Dans

chacun des cas, on commence donc par fixer un automate $\mathcal{A} = (A, Q, i, \delta, F)$ déterministe complet reconnaissant L.

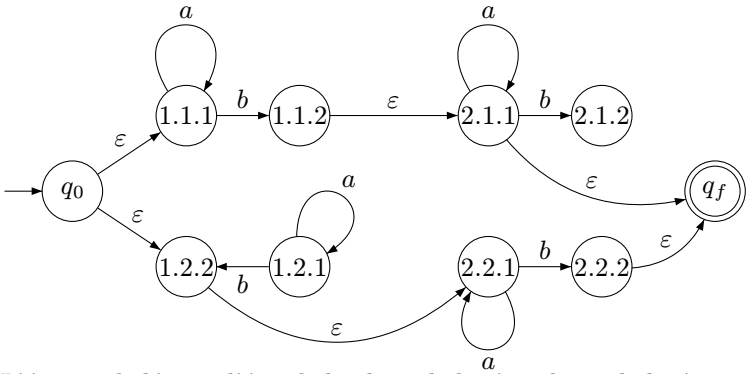
- $\overline{L}$ sera reconnu par l'automate non déterministe $\widetilde{\mathcal{A}} = (A, Q, F, \Delta, \{i\})$ ayant pour ensemble de transition

$$\Delta = \left\{ (q, \alpha, q') \;\middle|\; \delta(q', \alpha) = q \right\}$$

  (on a inversé le sens des transitions dans $\mathcal{A}$...). Pour montrer que $\widetilde{\mathcal{A}}$ reconnaît effectivement $\overline{L}$, on peut montrer par récurrence sur $|w|$ qu'il existe un chemin étiqueté par w entre q_1 et q_2 dans $\mathcal{A}$ si et seulement si il existe un chemin étiqueté par $\widetilde{w}$ dans $\widetilde{\mathcal{A}}$.
  Pour L_1, c'est l'automate donné plus haut (en faisant abstraction du puits, qui n'est pas accessible dans $\widetilde{\mathcal{A}}$).
- Si on déclare acceptants tous les états de Q co-accessibles, c'est-à-dire qui peuvent mener à un état terminal en lisant un certain mot[1], on obtient un automate qui reconnaît $Init(L)$.
  Pour L_1, c'est encore l'automate donné plus haut puisque seuls 1 et 2 sont co-accessibles.
- Si on coupe toutes les transitions issues des états terminaux (ou encore, pour les ronchons, si on les redirige vers un puits), on obtient un automate qui reconnaît $Min(L)$.
  Pour L_1, il n'y a pas de transition issue du seul état terminal, donc on obtient le même automate.
- Dans $\mathcal{A}$, on ne laisse acceptants que les états qui le sont déjà, et qui ne sont pas "strictement co-accessibles", c'est-à-dire à partir desquels on ne peut pas arriver sur un état final en lisant un mot non vide. On obtient alors un automate reconnaissant $Max(L)$.
  Notons que les états strictement co-accessibles s'obtiennent facilement par un parcours de graphe en largeur : c'est donc constructif.
  Pour L_1, le seul état final n'est pas strictement co-accessible, donc on obtient le même automate.
- Pour $Cycle(L)$, ça se complique...
  Notons $|Q| = k$, $Q = q_1, \dots q_k$. On réalise $2k$ copies de l'automate $\mathbb{A}$ reconnaissant L, que l'on dispose en deux colonnes. L'automate $\mathcal{A}'$ est alors construit de la façon suivante : l'état initial que l'on note q_0 pointe par ε-transition vers l'état q_1 du premier automate de la première colonne, vers q_2 du deuxième automate de la première colonne ... Ensuite les états finaux du premier automate de la premiere colonne pointent par ε-transition vers l'état initial q_1 de son voisin (le premier automate de la deuxième colonne) ; les états finaux du deuxième automate de la première colonne pointent par ε-transition vers l'état q_1 de son voisin de droite, ... Les états terminaux de $\mathcal{A}'$ seront q_1 du premier automate de la deuxième colonne, q_2 du deuxième automate de la deuxième colonne, ...
  On vérifie que c'est bien un automate reconnaissant $CYCLE(L)$: un mot étiquetant un chemin vainqueur ayant transité par la $i$ème ligne se décompose en $w_1 w_2$, où w_1 relie q_i à un état final (pour $\mathcal{A}$) f, et

[1] en pratique, quand on construit un automate, on a en général seulement le puits qui n'est pas co-accessible

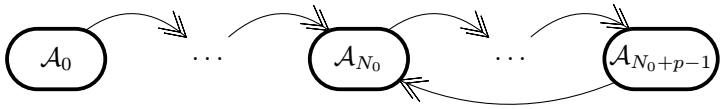
w_2 relie q_1 à q_i. w_2w_1 relie donc q_1 à f dans $\mathcal{A}$, donc $w_2w_1 \in L$ donc $w_1w_2 \in Cycle(L)$. La réciproque se fait sans problème.
Je n'ai pas eu le courage de faire le dessin pour le cas général, mais pour le cas particulier de L_1, puisqu'on a que deux états, c'est faisable :



L'état $i.j.k$ désigne l'état k du clone de la $j$ème ligne de la $i$ème colonne (ouf !).
NB : Les ε-transitions aboutissant à q_f sont un moyen élégant de ne mettre qu'un seul état final.
On vérifie que cet automate reconnaît effectivement $a^*ba^* + a^*b = a^*ba^*$ (Yessss...)

- Pour $\frac{1}{2}L$, ç'est un peu plus fin : après avoir lu un mot u de taille k dans $\mathcal{A}$, on aimerait savoir s'il existe un mot de taille k reliant l'état dans lequel on est à un état final de $\mathcal{A}$. On pourrait donc faire des transitions dans des copies $(\mathcal{A}_k)_{k\in\mathbb{N}}$ de $\mathcal{A}$, en remplaçant la transition (q_1, α, q_2) de $\mathcal{A}$ par la transition $(q_1^k, \alpha, q_2^{k+1})$ de $\mathcal{A}^\infty$ (la réunion de tous les $\mathcal{A}_k$) pour tout $k \in \mathbb{N}$, soit schématiquement :

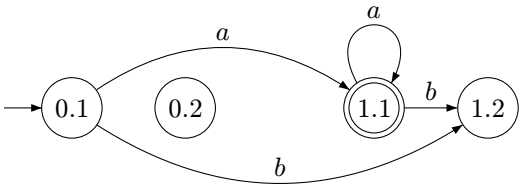
On déclarerait ensuite initial q_i^0, et acceptants dans cet "automate infini" les états q^k, pour chaque $q \in Q_k = \{q \in Q \mid \exists w \in A^k;\ \delta(q, w) \in F\}$.
Tout le problème est de "replier" cet automate. Pour cela, on note qu'il existe forcément $N_0, p \in \mathbb{N}$ avec $p > 0$ tels que $Q_{N_0} = Q_{N_0+p}$. Mais Q_{k+1} est complètement défini à partir de Q_k (il s'agit des états à l'origine d'une transition arrivant dans un état de Q_k). Ainsi, la suite (Q_n) va être cyclique à partir de N_0. On peut donc replier notre automate, en ne prenant que les clones $\mathcal{A}_0, ..., \mathcal{A}_{N_0+p-1}$, et en dirigeant les transitions issues de $\mathcal{A}_{N_0+p-1}$ vers $\mathcal{A}_{N_0}$:



Ainsi, dans ce macro-automate, en lisant u depuis l'état q_i^0,on arrive dans l'état q^j, avec $q = \delta(q_i, u)$ et j tel que $Q_j = Q_{|u|}$. Cet automate va

donc accepter exactement les u de $\frac{1}{2}L$.

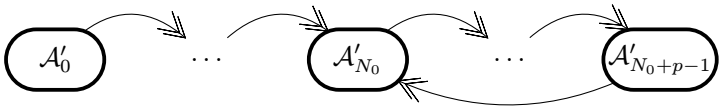
Pour L_1, on a $Q_0 = Q = \{2\}$, et $Q_k = \{1\}$ pour tout $k \geq 1$, donc le cycle est assez court, ce qui est heureux !



On a noté $i.j$ l'état j de $\mathcal{A}_i$. Cet automate reconaît clairement a^+, ce qui est le résultat souhaité (Yes yes...).

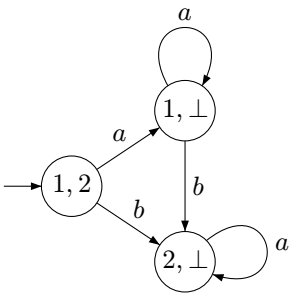
- Pour $\frac{2}{3}L$, ça devient monstrueux, puisqu'il faut remonter le temps en quelque sorte : en lisant u dans notre automate à construire, on veut savoir s'il existe un état q_1 de $\mathcal{A}$ accessible en $|u|$ lettres, depuis lequel en lisant u, on arrive dans un état "$|u|$-coaccessible". La dernière condition n'est pas un problème. Pour la première, il y a deux problèmes : connaître les états "$|u|$-accessibles", et connaître les $\delta(q,u)$ depuis chacun de ces états.
On va résoudre le premier problème comme pour le problème de co-accessibilité. Pour le second, on va utiliser un automate qui garde en mémoire les mouvements dans $\mathcal{A}$ depuis chaque état (cf le problème de $\sqrt[N]{L}$)

Pour $k \in \mathbb{N}$, on définit donc I_k l'ensemble des états k-accessibles, c'est-à-dire de la forme $\delta(q_i, u)$ pour un certain $u \in A^k$. I_{k+1} est l'ensemble des états accessibles depuis un état de I_k en une transition. Ainsi, pour les mêmes arguments que précédemment, (I_n) est cyclique, et même : $\big((I_k, Q_k)\big)_{k\in\mathbb{N}}$ est cyclique. On va noter N_0 et p deux entiers (avec $p > 0$) tels que $(I_{N_0+p}, Q_{N_0+p}) = (I_{N_0}, Q_{N_0})$.
Avec $Q = [\![1, n]\!]$, notons $\mathcal{A}' = \big(A, [\![1, n]\!]^n, (1, ..., n), \delta'\big)$ le macro-automate (sans état final) regroupant les mouvements dans $\mathcal{A}$ depuis tous les états, en posant $\delta'\big((i_1, ..., i_n), \alpha\big) = \big(\delta(i_1, \alpha), ..., \delta(i_n, \alpha)\big)$.
On va considérer le macro-(macro-automate) :

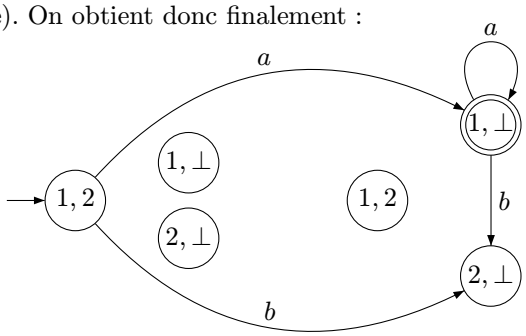


Il n'y a plus qu'à déclarer initial $(1, ..., n)^0$, et acceptants les états $(i_1, ..., i_n)^k$ tels que il existe un état q k-accessible depuis lequel, en lisant u, on arrive dans un état k-coaccessible, c'est-à-dire : il existe $j \in I_k$ tel que $i_j \in Q_k$.
Tout est en place pour pouvoir affirmer la tête haute et sans sourciller que cet automate reconnaît clairement $\frac{2}{3}L$.

Pour L_1, on a $I_0 = \{1\}$ et $I_k = \{1, 2\}$ pour tout $k \geq 1$. Par ailleurs le macro-automate est :



avec $\perp$ désignant le puits (ça fait vachement plus informaticien-logicien-pipologue). On obtient donc finalement :



Le langage reconnu est bien a^+ (yes yes yes yes ! ! !)

6. Un début de lassitude me gagnant, je me contente de donner les expressions rationnelles (qui c'est qui m'a foutu une question comme ça à la fin ? ? ? ? ?)
   - $\overline{L_3} = b^*a^*$.
   - $Init(L_3) = a^*b^*$.
   - $Min(L_3) = \{\varepsilon\}$.
   - $Max(L_3) = \emptyset$.
   - $Cycle(L) = a^*b^*a^* + b^*a^*b^*$.
   - $\frac{1}{2}L = a^*b^*$.
   - $\frac{2}{3}L = a^*b^*$.

On restera longuement en émoi face aux qualités pédagogiques et calligraphiques de ce corrigé.