DM n°1 – FDI

Hugo Salou

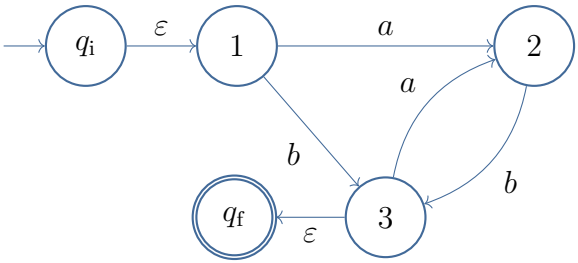
ENS DE LYON

5 octobre 2024

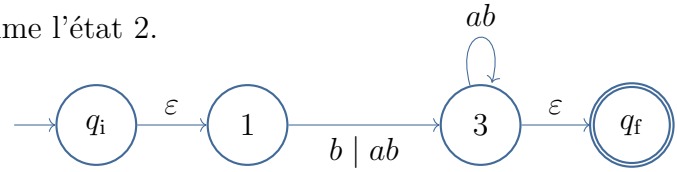
Exercice 1.

On applique l'algorithme d'éliminations d'états sur l'automate. Les différentes étapes sont représentées.

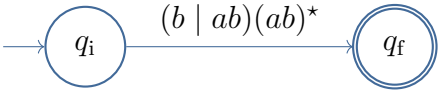
On commence par « détourer » l'automate (on ajoute q_i et q_f).



On supprime l'état 2.



On supprime les états 2 et 3.



On en déduit une expression régulière équivalente à l'automate initial :

$$(b \mid ab)(ab)^\star.$$

Exercice 2.

On procède en deux temps :

1. on réalise un DFA $\mathcal{A}_1$ reconnaissant le langage $\mathcal{L}((aba)^\star) = L$;
2. et on inverse les états finaux et non finaux (donnant $\mathcal{A}_2$).

L'automate déterministe obtenu reconnaitra le langage $\Sigma^\star \setminus L$.

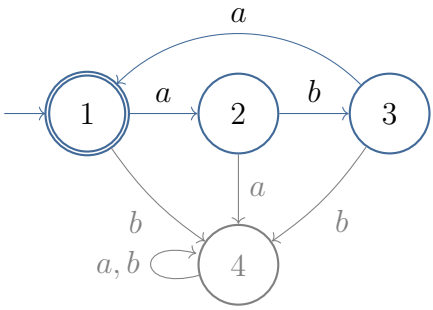


Figure 1 | *Automate fini déterministe pour l'étape 1*

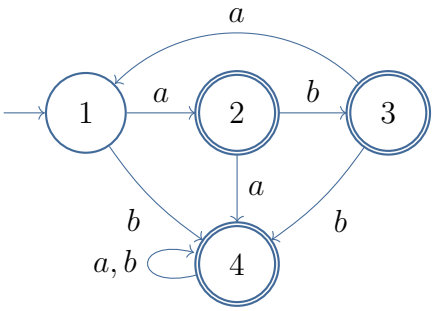


Figure 2 | *Automate fini déterministe pour l'étape 2*

DM n°2 – FDI

Hugo Salou

ENS DE LYON

21 octobre 2024

Exercice 1.

On pose, pour tout $n \in \mathbb{N}$, le mot $v_n = 1^n 2$. Montrons que les mots de la suite $(v_n)_{n\in\mathbb{N}}$ sont deux-à-deux non-équivalents pour la relation de congruence $\equiv_L$.

Considérons les mots $v_n = 1^n 2$ et $v_m = 1^m 2$ pour $n \neq m$. Sans perdre en généralité, supposons $n < m$. Soit z le mot 2^{m-1}. Remarquons que l'on a $v_m z = 1^m 2^m \in L$ mais que $v_n z = 1^n 2^m \notin L$.

Le théorème de Myhill-Nerodes démontre ainsi que l'automate minimal reconnaissant L a un nombre infini d'états. On en conclut que L n'est pas rationnel.

Exercice 2.

On applique l'algorithme de Moore en calculant les classes d'équivalences des relations $\equiv_i$ pour tout $i \in \mathbb{N}$ jusqu'à sa stationnarité :

- ▷ *Étape 0.* les classes sont $\{1, 2, 6\}$ et $\{3, 4, 5\}$;
- ▷ *Étape 1.* les classes sont $\{1, 2\}$, $\{6\}$ et $\{3, 4, 5\}$;
- ▷ *Étape 2.* les classes sont $\{1\}$, $\{2\}$, $\{6\}$ et $\{3, 4, 5\}$;
- ▷ *Étape 3.* les classes sont $\{1\}$, $\{2\}$, $\{6\}$ et $\{3, 4, 5\}$.

On s'arrête à l'étape 3 car on a atteint un point fixe ($\equiv_2 = \equiv_3$).

On en déduit l'automate minimal : chaque état représente une classe d'équivalence pour la congruence de Nerode.

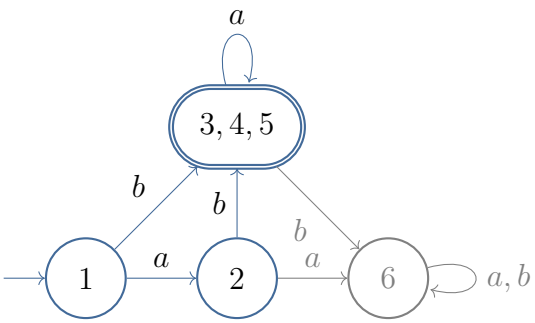


Figure 1 | *Automate minimal équivalent*

DM n°3 – FDI

Hugo Salou

ENS DE LYON

14 novembre 2024

On nomme L le langage

$$L := \{\langle M\rangle \in \Sigma^\star \mid M \text{ s'arrête sur l'entrée } \langle M\rangle\}.$$

Nous allons démontrer que ce langage est

- ▷ indécidable (par réduction au langage de l'arrêt A_{TM} pour les machines de Turing) ;
- ▷ récursivement énumérable ;
- ▷ mais le complémentaire n'est pas récursivement énumérable.

On rappelle que le langage $A_{\mathsf{TM}} := \{\langle M, w\rangle \mid M \text{ accepte } w\}$ est indécidable.

On pose la fonction

$$\begin{aligned} f : \quad \Sigma^\star &\longrightarrow \Sigma^\star \\ \langle N, w\rangle &\longmapsto \langle M_{N,w}\rangle \end{aligned}$$

qui est calculable.

Machine $M_{N,w}$

Entrée Un mot $u \in \Sigma^\star$
On simule N sur l'entrée w.
si N accepte w **alors**
  On accepte.
sinon
  On boucle à l'infini.

L'« implémentation haut niveau » de la machine $M_{N,w}$, donnée ci-dessus à droite, permet de construire une telle machine. En effet, l'étape « on simule N sur l'entrée w » se réalise à l'aide de la machine universelle, et le reste se fait simplement.

On remarque que $M_{N,w}$ s'arrête sur une entrée quelconque si, et seulement si, N accepte w.

Et, on a l'équivalence

$$\begin{aligned} \langle M_{N,w}\rangle \in L &\iff M_{N,w} \text{ s'arrête sur } \langle M_{N,w}\rangle \\ &\iff N \text{ accepte } w \\ &\iff \langle N, w\rangle \in A_{\mathsf{TM}}.. \end{aligned}$$

Ainsi, par réduction, on en déduit que L n'est pas décidable.

Pour démontrer que L est récursivement énumérable, on montre (de manière équivalente) qu'il est Turing-reconnaissable. Considérons la machine de Turing R dont l'implémentation est donnée ci-dessous.

Sur toutes les entrées $\langle M \rangle \in L$, la machine R accepte cette entrée en un temps fini. On en déduit que R reconnait bien L, et donc L est Turing-reconnaissable donc récursivement énumérable.

Machine R

Entrée $\langle M \rangle$
  Simuler M sur $\langle M \rangle$.
  si M accepte $\langle M \rangle$ **alors**
    On accepte.

On peut démontrer que L n'est pas de complémentaire récursivement énumérable. En effet, par l'absurde, si L l'était, alors $\bar{L}$ serait Turing-reconnaissable. Mais, ceci impliquerai (comme L et $\bar{L}$ seraient Turing-reconnaissable) que L serait décidable (il suffit de lancer les deux machines reconnaissant L et $\bar{L}$ en parallèle pour décider L). Ce qui est ***absurde*** car on a montré l'indécidabilité de L.

Des résultats précédents, on en conclut que

- L est indécidable ;
- L est récursivement énumérable ;
- L n'est pas à complémentaire récursivement énumérable.

DM n°4 – FDI

Hugo Salou

ENS DE LYON

20 décembre 2024

Dans la première solution, on n'utilise pas de quine mais on instaure une propriété très utilie : « pour toute lettre du programme, elle apparait exactement 10 fois ». Pour réaliser cela, on cache beaucoup de caractères dans un commentaire (comme par exemple `f` ou `[`).

```
import sys; print(10 if sys.argv[1] in "\"\\\n#ev
    ↪ .famit␣;p(:n0[rl)1s]gyo" else 0) #
    ↪ eeeeeeevvvvvvvv........
    ↪ fffffffffaaaaaaaammmmmmmmiiiiiitttttttppppppp
    ↪ \\\\\\\(((((((((:::::::::"""""""nnnnnn0000000
    ↪ [[[[[[[[rrrrrr;;;;;;;;llllllll))))))))
    ↪ 1111111ssss]]]]]]]]ggggggggyyyyyyyoooooooo
#
#
#
#
#
#
#
#
#
#
```

Code 1 | *Solution sans quine au DM 4*

C'est pas joli, très peu généralisable, mais ça répond au DM.

J'ai quand même proposé une seconde solution à ce DM une *quine* (sans tricher comme à la proposition précédente), c'est à dire un programme qui s'écrit lui même. La solution est encore plus concise mais utilise des « *string format* » (à la `printf` en C).

```
x="x=%c%s%c;y=x%c(chr(34),x,chr(34),chr(37),chr
    ↪ (10));import␣sys;print(y.count(sys.argv[1])
    ↪ )%c";y=x%(chr(34),x,chr(34),chr(37),chr(10)
    ↪ );import sys;print(y.count(sys.argv[1]))
```

Code 2 | *Solution avec quine au DM 4*

L'idée est simple, dans `x`, on écrit le code source. Il contient trois parties, le début de la définition de `x` (*i.e.* « `x =` »), la chaîne `x`

elle-même (où l'on utilise ici le « `%s` » pour interpoler la valeur de `x` dedans), et enfin le reste du code (la partie qui compte le nombre de caractères, et qui l'affiche en console). Les caractères problématiques, comme les guillemets, les retours à la ligne (parce que le *backslash*), le symbole pour-cent, *etc*, sont interpolés aussi dans la chaîne avec « `%c` » et la fonction `ord` qui prend un entier et renvoie le caractère ASCII à l'indice donné dans la table.

DM n°5 – FDI

Hugo Salou

ENS DE LYON

20 décembre 2024

1. Considérons un ensemble fini E. On montre, par récurrence sur $|E|$, que E est récursif primitif.

- ▷ Dans le cas où $|E| = 0$, alors $E = \emptyset$ alors $\chi_E = \mathbf{0}$, où $\mathbf{0}$ est la fonction constante nulle.
- ▷ Considérons $|E| = n+1$. Soit $x \in E$ et $E' := E \setminus \{x\}$. Par hypothèse de récurrence $\chi_{E'}$ est primitive récursive. En TD, on a vu que la construction « `if ... then ... else` » [1] (avec `true` représenté par 1 et `false` par 0) pouvait être réalisée par une fonction primitive récursive, on s'appuie donc sur cette construction. De même, le test d'égalité peut être réalisé (il suffit de calculer les deux quasi-différences dans $\mathbb{N}$ et de vérifier que les deux sont nulles). On a donc

$$\chi_E(y) = \texttt{if eq}(x, y)\ \texttt{then}\ \mathbf{1}(y)\ \texttt{else}\ \chi_{E'}(y).$$

Ainsi, tout ensemble fini est récursif primitif.

En notant $E = \{x_1, \dots, x_n\}$, le raisonnement par récurrence donne la fonction primitive récursive :

$$\begin{aligned}\chi_E(y) = {} & \texttt{if eq}(x_1, y)\ \texttt{then}\ \mathbf{1}(x)\ \texttt{else} \\ & \texttt{if eq}(x_2, y)\ \texttt{then}\ \mathbf{1}(x)\ \texttt{else} \\ & \texttt{if eq}(x_3, y)\ \texttt{then}\ \mathbf{1}(x)\ \texttt{else} \\ & \qquad\qquad \vdots \\ & \texttt{if eq}(x_n, y)\ \texttt{then}\ \mathbf{1}(x)\ \texttt{else} \\ & \mathbf{0}(x).\end{aligned}$$

Montrons que P_k est récursif primitif en montrant que $k\mathbb{N}$ l'est. En TD, on a implémenté la fonction `div` calculant le quotient modulo un nombre précisé en argument, et `mult` calculant le produit. On pose donc

$$\chi_{k\mathbb{N}}(x) = \texttt{eq}(\texttt{mult}(\texttt{div}(x, k), k), x).$$

1. Pour la lisibilité, j'écrirais cette construction comme une commande, et non comme un appel à une fonction (c'est un petit abus de notations).

En effet, on a

$$x \in k\mathbb{N} \iff k \mid x \iff \left\lfloor \frac{x}{k} \right\rfloor \times k = x.$$

(On aurait aussi pu passer par la fonction `mod` définie en TD.)

2. Supposons A et B primitifs récursifs. Alors, les ensembles $A \cap B$, $A \cup B$ et A^{C} le sont. En effet, on a :

- ▷ $\chi_{A\cap B}(x) = \texttt{mult}(\chi_A(x), \chi_B(x))$;
- ▷ $\chi_{A\cup B}(x) = \texttt{sub}(\texttt{add}(\chi_A(x), \chi_B(x)), \chi_{A\cap B}(x))$;
- ▷ $\chi_{A^{\mathrm{C}}}(x) = \texttt{sub}(\mathbf{1}(x), \chi_A(x))$.

Ces relations proviennent des relations entre fonctions indicatrices :

$$\begin{aligned} \chi_{A\cap B} &= \chi_A \times \chi_B \\ \chi_{A\cup B} &= \chi_A + \chi_B - \chi_{A\cap B} \\ \chi_{A^{\mathrm{C}}} &= 1 - \chi_A \end{aligned}$$

On en conclut que les ensembles primitifs récursifs sont clos par les opérations ensemblistes $\cap$, $\cup$ et $-^{\mathrm{C}}$.

3. On a

$$\begin{aligned} g(x) = \ & \texttt{if } \chi_{A_1}(x) \texttt{ then } f_1(x) \texttt{ else} \\ & \texttt{if } \chi_{A_2}(x) \texttt{ then } f_2(x) \texttt{ else} \\ & \texttt{if } \chi_{A_3}(x) \texttt{ then } f_3(x) \texttt{ else} \\ & \qquad\qquad \vdots \\ & \texttt{if } \chi_{A_k}(x) \texttt{ then } f_k(x) \texttt{ else} \\ & f(x). \end{aligned}$$

(La fonction définie en question 1 est un cas particulier où l'on a posé $A_i = \{x_i\}$ (et donc $\chi_{A_i}(x) = \texttt{eq}(x, x_i)$), $f_i = \mathbf{1}$ et $f = \mathbf{0}$, on retrouve ainsi la même fonction primitive récursive.) Formellement, on procède par récurrence sur k.

- ▷ Pour $k = 0$, on pose $g = f$, qui est primitive récursive.

- Pour $k > 0$, on considère g' définie « cas par cas » (par hypothèse de récurrence) sur les k cas $((A_1, f_1), \ldots, (A_k, f_k))$, primitive récursive, et on a

$$g(x) = \texttt{if } \chi_{A_{k+1}}(x) \texttt{ then } f_{k+1}(x) \texttt{ else } g'(x),$$

  qui est aussi primitive récursive, d'où l'hérédité.

Ceci permet de conclure que les fonctions primitives récursives sont closes par schéma de définition par cas.