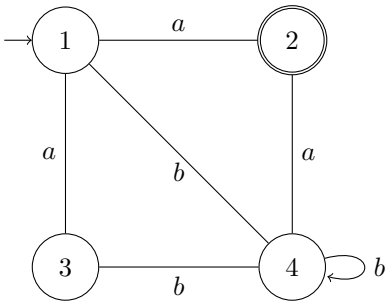
Déterminisation

Donner un automate déterministe complet équivalent à :



Reconnaissance

Dessiner un automate reconnaissant le langage des mots sur $\{a, b\}$ ayant un nombre pair de a et dont le nombre de b est multiple de 3.

Langage non rationnel

Montrer que $L = \{a^p : p \text{ premier } \}$ n'est pas rationnel.

On suppose disposer d'une structure impérative de dictionnaire en Caml de type `('a,'b) dict` avec les primitives suivantes :

Primitive	Type	Description
`new`	`unit -> ('a,'b) dict`	Crée un nouveau dictionnaire vide
`add`	`('a,'b) dict -> 'a -> 'b -> unit`	`add d cl val` associe, dans le dictionnaire `d`, la clef `cl` à la valeur `val`
`find`	`('a,'b) dict -> 'a -> 'b`	`find d cl` lit, dans le dictionnaire `d`, la valeur associée à la clef `cl`
`keys`	`('a,'b) dict -> 'a list`	`keys d` renvoie la liste (dans un ordre arbitraire) des clefs du dictionnaire `d`

`'a` est le type des clefs, et `'b` le type des valeurs associées aux clefs.

On suppose disposer d'une structure persistante d'ensemble d'entiers de type `set` avec les primitives suivantes :

Primitive	Type	Description
`empty`	`set`	L'ensemble vide
`union`	`set -> set -> set`	L'union de 2 ensembles
`inter`	`set -> set -> set`	L'intersection de 2 ensembles
`card`	`set -> int`	Le cardinal d'un ensemble
`equal`	`set -> set -> bool`	Teste l'égalité de 2 ensembles
`mem`	`int -> set -> bool`	`mem i s` renvoie `true` si l'entier `i` appartient à l'ensemble `s` et `false` sinon
`singleton`	`int -> set`	`singleton i` renvoie l'ensemble à un élément ne contenant que `i`

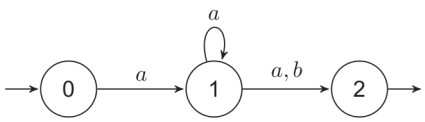
On représente un automate non-déterministe en Caml par le type suivant

```
type auto = {etats : set; init: set;
       trans: (int * char, set) dict ; final: set};;
```

Étant donné un automate `m`,

- `m.init` représente l'ensemble des états initiaux de `m` ;
- `m.final` l'ensemble de ses états finaux,
- `m.trans` sa fonction de transition qui à chaque couple `q,x` associe l'ensemble des états accessibles à partir de l'état `q` en lisant le caractère `x`.

Par exemple, considérons l'automate $\mathcal{M}_1$ suivant :



Si `m1` représente l'automate $\mathcal{M}_1$ en Caml alors `m1.final` est l'ensemble $\{2\}$ et `find m1.trans (1,`a`)` est l'ensemble $\{1;2\}$.

Un automate déterministe est un automate non-déterministe ayant un unique état initial et tel que pour tout état q et toute lettre a, en lisant a à partir de l'état q on peut aller dans au plus un état.

1. L'automate $\mathcal{M}_1$ est-il déterministe ?
2. Écrire une fonction `max_card : ('a, set) dict -> int` qui étant donné un dictionnaire d'ensembles, renvoie le cardinal maximal des ensembles stockés dans le dictionnaire.
3. Écrire une fonction `est_deterministe : auto -> bool` qui renvoie `true` si l'automate donné en argument est déterministe et `false` sinon.
4. Considérons le code incomplet suivant :

```
let etats_suivants m s x =                                    1
let rec parcours_clefs clefs acc = match clefs with           2
        | [] -> acc                                           3
        | (etat,y)::r -> if  .................                4
                 then  .................                      5
                 else  .................                      6
in parcours_clefs (keys m.trans) empty;;                      7
```

   Compléter le code de sorte que `etats_suivants m s x` renvoie l'ensemble des états accessibles à partir d'un état de l'ensemble `s` en lisant `x` dans l'automate `m`.

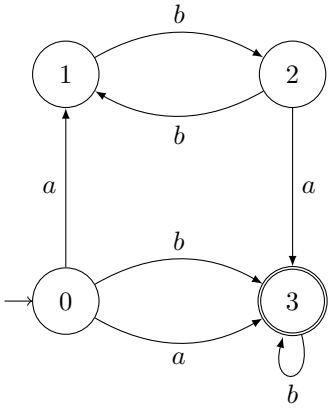
5. Écrire une fonction `reconnu : auto -> string -> bool` qui, étant donné un automate et une chaîne de caractères, renvoie `true` si l'automate reconnaît la chaîne et `false` sinon.
6. Si au lieu d'une structure impérative de dictionnaire nous avions une structure persistante de dictionnaire, quel serait le type de la primitive `add` ? Si nous avions une structure impérative d'ensemble d'entiers et non pas une structure persistante, pourquoi le type de `empty` devrait-il être changé ?

Reconnaissance

Montrer que le langage sur $\{0,1\}$ des écritures en base 2 des multiples de 5 est reconnaissable. Si $n \in \mathbb{N}$ est quelconque, le langage sur $\{0,1\}$ des écritures en base 2 des multiples de n est-il reconnaissable ?

Élimination

À l'aide de l'algorithme d'élimination d'états, donner une expression rationnelle de même langage que l'automate :



Miroir d'un langage

Si $m = m_1...m_n$ est un mot, on définit son miroir $\widetilde{m} = m_n...m_1$.
Si L est un langage, on définit son miroir $\widetilde{L} = \{\widetilde{m} \mid m \in L\}$.

1. Donner une expression rationnel du miroir de $L(a(a+b)^*b)$.
2. Soit e une expression rationnelle de langage L. Définir récursivement une expression rationnelle $\widetilde{e}$ de langage $\widetilde{L}$.
3. Écrire une fonction Caml `miroir : 'a regexp -> 'a regexp` renvoyant le miroir d'une expression rationnelle. On utilisera le type suivant d'expression régulière (`L(a)` désigne une lettre `a`):

```
type 'a regexp =
    | Vide | Epsilon | L of 'a
    | Somme of 'a regexp * 'a regexp
    | Concat of 'a regexp * 'a regexp
    | Etoile of 'a regexp
```

A4 – Automates et palindromes

On fixe un alphabet Σ avec $|\Sigma| > 1$. Un mot $w \in \Sigma^*$ est un *palindrome* s'il s'écrit $w = a_1 \cdots a_n$ et qu'on a $a_i = a_{n-i+1}$ pour tout $1 \leq i \leq n$. On note $\Pi \subseteq \Sigma^*$ le langage des palindromes. Pour un automate fini A sur Σ, on note $L(A)$ le langage reconnu par A.

Question 0. Soit $\Pi_n := \Pi \cap \Sigma^n$. Montrer que pour tout automate fini déterministe complet A, pour tout $n \in \mathbb{N}$, si $L(A) \cap \Sigma^{2n} = \Pi_{2n}$, alors A a au moins $|\Sigma|^n$ états.

Question 1. En déduire que le langage Π n'est pas régulier.

Question 2. Étant donné un automate fini A sur Σ, peut-on calculer un automate A_Π qui reconnaisse $L(A) \cap \Pi$?

Question 3. Pour tout mot $u = b_1 \cdots b_m$ de Σ^*, on note $\overline{u} := b_m \cdots b_1$ son miroir. Étant donné A, peut-on calculer un automate A'_Π qui reconnaisse $\{u \in \Sigma^* \mid u\overline{u} \in L(A)\}$?

Question 4. On appelle Π_{pair} l'ensemble des palindromes de longueur paire, i.e., $\Pi_{\text{pair}} := \bigcup_{n\in\mathbb{N}} \Pi_{2n}$. Proposer un algorithme qui, étant donné un automate fini A sur Σ, détermine si $L(A) \cap \Pi_{\text{pair}}$ est vide, fini, ou infini. Discuter de sa complexité en temps et en espace.

Question 5. Modifier l'algorithme de la question 4 pour calculer la cardinalité de $L(A) \cap \Pi_{\text{pair}}$ quand cet ensemble est fini, en faisant l'hypothèse que l'automate d'entrée A est déterministe. Comment la complexité est-elle affectée ?

Question 6. Modifier l'algorithme des questions 4 et 5 pour qu'il s'applique à $L(A) \cap \Pi$.

Corrigé

Question 0. Soit A un tel automate fini déterministe d'ensemble d'états Q, et fixons $n \in \mathbb{N}$. Considérons la fonction $f : \Sigma^n \to Q$ qui associe à $w \in \Sigma^n$ l'état q (unique) auquel A aboutit en lisant w. Montrons que f est injective. Procédons par l'absurde et supposons que $f(u) = f(v)$ pour $u \neq v$ de Σ^n. On sait que $u\overline{u}$ est un palindrome de longueur $2n$ donc il y a un chemin étiqueté par $\overline{u}$ de $f(u)$ à un état final de A. En combinant ce chemin avec le chemin de l'état initial à q étiqueté par v, on conclut que l'automate accepte $v\overline{u}$. Comme $u \neq v$, c'est pourtant un mot de longueur $2n$ qui n'est pas un palindrome, contradiction. Ainsi, f est injective, donc $|Q| \geq |\Sigma^n| \geq |\Sigma|^n$.

Question 1. Procédons par l'absurde et supposons que Π soit régulier. Soit A un automate fini déterministe complet qui reconnaisse Π, et soit n son nombre d'états. Comme $L(A) = \Pi$ par hypothèse, on sait que $L(A) \cap \Sigma^{2n} = \Pi_{2n}$, ainsi d'après la question précédente A a au moins $|\Sigma|^n \geq 2^n$ états, mais il en a n et on a $n < 2^n$, donc contradiction. Ainsi Π n'est pas régulier.

Question 2. C'est manifestement déraisonnable : pour A un automate reconnaissant le langage régulier Σ^*, la question nous demanderait de calculer un automate qui reconnaisse $\Sigma^* \cap \Pi = \Pi$, et on sait d'après la question précédente qu'il n'en existe pas.

Question 3. De façon un peu contre-intuitive, c'est possible. Posons $A = (Q, I, F, \delta)$ où Q est l'ensemble d'états de A, où I est l'ensemble d'états initiaux, où F est l'ensemble d'états finaux, et où $\delta \subseteq Q \times \Sigma \times Q$ est la relation de transition. On construit d'abord en temps linéaire l'automate $\overline{A} := (Q, F, I, \overline{\delta})$ où $\overline{\delta} := \{(q', a, q) \mid (q, a, q') \in \delta\}$. Il est clair que $\overline{A}$ reconnaît $\overline{L(A)} := \{\overline{u} \mid u \in L(A)\}$, puisqu'il y a une correspondance bijective entre les chemins acceptants dans A et dans $\overline{A}$, et l'effet de cette bijection sur l'étiquette des chemins est l'opération miroir.

On construit ensuite (en temps quadratique en A) l'automate produit $A \times \overline{A}$ défini comme suit : $A \times \overline{A} := (Q \times Q, I \times F, F', \delta \times \overline{\delta})$ où la relation de transition est définie comme $\overline{\delta} := \{((q, \overline{q'}), a, (q', \overline{q}) \mid (q, a, q') \in \delta \wedge (\overline{q'}, a, \overline{q}) \in \overline{\delta}$, et où les états finaux sont $\{(q, q) \mid q \in Q\}$. Il est clair que l'automate produit peut atteindre l'état $(q, \overline{q})$ en lisant un mot $w \in \Sigma^*$ si et seulement si l'automate A peut atteindre l'état q en lisant w (dans la première composante) et l'automate $\overline{A}$ peut atteindre l'état $\overline{q}$ en lisant w (dans la seconde composante), ce qui est le cas, par définition de $\overline{A}$, si et seulement s'il y a un chemin dans A étiqueté par $\overline{u}$ de $\overline{q}$ à un état final de F. En particulier, l'automate produit peut atteindre l'état (q, q) en lisant un mot $w \in \Sigma^*$ si et seulement si l'automate A peut atteindre q en lisant w et l'automate A a un chemin étiqueté par $\overline{u}$ de q à un état final. Ainsi, la définition de F' assure que l'automate produit accepte un mot $w \in \Sigma^*$ si et seulement s'il existe un état q tel que A a un chemin acceptant pour $w\overline{w}$ qui passe par un certain état q après la lecture de w, c'est-à-dire si et seulement si A accepte $w\overline{w}$. Ceci établit que la construction est correcte.

[Cette construction est donnée dans [ALR+09], Lemma 2.]

Question 4. La construction de l'automate produit A'_Π de la question précédente s'effectue en temps quadratique en A. Il est clair que $L(A) \cap \Pi_{\text{pair}}$ a la même cardinalité que $L(A'_\Pi)$, puisque la fonction $f : \Sigma^* \to \Pi_{\text{pair}}$ définie par $f(u) := u\overline{u}$ pour tout $u \in \Sigma^*$ définit une bijection entre $L(A'_\Pi)$ et $L(A) \cap \Pi_{\text{pair}}$. Ainsi, il suffit de déterminer si $L(A'_\Pi)$ est vide, fini, ou infini.

On détermine d'abord si $L(A'_\Pi)$ est non-vide en vérifiant qu'il existe un chemin d'un état initial de A'_Π à un état final de A'_Π, en temps linéaire, par exemple avec un DFS.

On détermine ensuite si A'_Π contient un cycle d'états accessibles et co-accessibles. Pour ce faire, on détermine les états accessibles et co-accessibles par un parcours de graphe, et ensuite on utilise un DFS pour déterminer si on rencontre un cycle dans ces sommets. Ceci est toujours en temps linéaire, et conclut.

Question 5. Dans le cas où l'automate produit est déterministe, on peut compter le nombre de mots qu'il accepte par de la programmation dynamique. Spécifiquement, le nombre de mots acceptés à partir d'un état q est 1 ou 0 selon que q est final ou non, plus le nombre de mots acceptés à partir des états vers lesquels q a des transitions sortantes. Noter que, comme l'automate est déterministe, les étiquettes de ces transitions sont différentes, ainsi les ensembles de mots concernés sont disjoints, et c'est effectivement correct de faire la somme comme expliqué.

On n'a en fait pas vraiment besoin que l'automate produit soit déterministe : il suffit qu'il soit *inambigu*, c'est-à-dire que tout mot accepté a un unique chemin acceptant. Dans ce cas, la construction présentée reste correcte, parce que si un état q a des transitions étiquetées par la même lettre vers deux états distincts q_1 et q_2, alors la définition de l'inambiguïté impose que l'ensemble des mots acceptés à partir de q_1 et celui des mots acceptés à partir de q_2 sont disjoints.

Il suffit alors d'observer que, si A est déterministe (ou, en fait, s'il est inambigu), alors $\overline{A}$ est toujours inambigu (même si pas forcément déterministe). Maintenant, le produit $A \times \overline{A}$ avec l'ensemble d'états finaux indiqué est également inambigu : si un mot $u \in \Sigma^*$ avait un chemin vers (q, q) et vers (q', q') dans l'automate produit avec $q \neq q'$, alors on saurait que $u\overline{u}$ a deux chemins acceptants distincts dans A (un où l'état intermédiaire est q, un autre où c'est q'), ce qui contredirait le fait que A soit inambigu. Ainsi, si A est inambigu on peut construire le produit avec la même complexité qu'avant, il est inambigu, et on compte la taille du langage qu'il accepte comme expliqué. Comme cet ensemble est en bijection avec $L(A) \cap \Pi_{\text{pair}}$ (comme prouvé à la question 4), on a établi le résultat.

[L'hypothèse que l'automate d'entrée est déterministe ou inambigu est probablement indispensable, parce que compter le nombre de mots acceptés par un automate quelconque est #P-difficile : voir [KSM95] Theorem 2.1.]

Question 6. L'automate produit A' de la question 4 ne gère intuitivement que les u qui se complètent en un palindrome de longueur paire. Ceci dit, pour chaque $a \in \Sigma$, on peut adapter l'ensemble d'états finaux de l'automate produit pour construire un automate A'_a qui reconnaisse exactement $\{u \in \Sigma^* \mid ua\overline{u} \in L(A)\}$, c'est-à-dire le langage des mots u qui se complètent en un palindrome de longueur impaire accepté par A en ajoutant a comme lettre centrale, puis le miroir de u. On définit A'_a pour chaque $a \in \Sigma$ exactement comme A' mais avec l'ensemble d'états finaux $F'_a := \{(q, q') \mid (q, a, q') \in \delta\}$, ce qui est clairement correct : un mot $u \in \Sigma^*$ a un chemin acceptant dans A'_a finissant en (q, q') si et seulement s'il y a un chemin d'un état initial de A à q étiqueté par u, une transition dans A étiquetée par a de q à q', et un chemin de q' à un état final de A étiqueté par $\overline{u}$ dans A. Du reste, si A est inambigu alors tous ces automates produits le sont, par le même raisonnement qu'à la question précédente. On peut ainsi appliquer la construction de la question précédente à A' et à A'_a pour chaque $a \in \Sigma$, ce qui ne fait que rajouter un facteur $|\Sigma|$ à la complexité : on peut obtenir la cardinalité du nombre de palindromes reconnus en sommant les quantités pour A' et pour chaque A'_a (en traitant ∞ de la manière attendue). Ceci est correct parce que les palindromes de $\Pi \cap L(A)$ se partitionnent entre ceux de longueur paire et ceux de longueur impaire dont la lettre centrale est $a \in \Sigma$ pour chaque a. (En revanche, noter que le langage de A' et celui des A'_a ne sont pas forcément disjoints, vu qu'il est tout à fait possible, par exemple, que A accepte $u\overline{u}$ et $ua\overline{u}$ pour diverses valeurs de $a \in \Sigma$. Autrement dit, il faut bien sommer la cardinalité de ces langages même si leur union n'est pas disjointe, pour la raison qu'on vient d'expliquer.)

Références

[ALR+09] Terry Anderson, John Loftus, Narad Rampersad, Nicolae Santean, and Jeffrey Shallit. Detecting palindromes, patterns and borders in regular languages. *Information and Computation*, 207(11), 2009. `https://www.sciencedirect.com/science/article/pii/S0890540109000650`.

A7 – Langages continuables et mots primitifs

On fixe un alphabet fini Σ et on suppose $|\Sigma| > 1$. Dans ce sujet, on considérera des automates sur l'alphabet Σ qui seront toujours supposés déterministes complets.

Un mot non-vide $w \in \Sigma^*$ est dit *primitif* s'il n'existe pas de mot $u \in \Sigma^*$ et d'entier $p > 1$ tel que $w = u^p$.

Question 0. Le mot *abaaabaa* est-il primitif ? Le mot *ababbaabbbababbab* (de longueur 17) est-il primitif ?

Question 1. Proposer un algorithme naïf qui, étant donné un mot, détermine s'il est primitif, et discuter de sa complexité en temps et en espace.

Question 2. Donner un exemple d'un langage régulier infini qui ne contienne aucun mot primitif.

Question 3. Donner un exemple d'un langage régulier infini qui ne contient que des mots primitifs.

Question 4. Un langage régulier L est dit *continuable* s'il a la propriété suivante : pour tout $u \in \Sigma^*$, il existe $v \in \Sigma^*$ tel que $uv \in L$. Donner un exemple de langage régulier infini non continuable. Existe-t-il des langages réguliers continuables dont le complémentaire soit infini ?

Question 5. Étant donné un automate A, proposer un algorithme pour déterminer si le langage $L(A)$ qu'il reconnaît est continuable. Justifier sa correction et discuter de sa complexité en temps et en espace.

Question 6.

(a) Montrer que tout langage régulier continuable contient une infinité de mots primitifs.

(b) Étant donné un langage régulier continuable L, donner une borne supérieure sur la taille du plus petit mot primitif de L.

(c) La réciproque de la question (a) est-elle vraie ?

Corrigé

Question 0. Le mot $abaaabaa$ n'est pas primitif puisqu'on peut l'écrire $(abaa)^2$.

Le second mot proposé est primitif : comme sa longueur est un nombre premier, la seule factorisation possible serait comme l'exponentiation d'un mot de longueur 1, ce qui n'est pas possible car tous ses caractères ne sont pas identiques.

Question 1. On suppose sans perte de généralité que le mot d'entrée est non-vide.

Étant donné un mot $w \in \Sigma^*$, il suffit de tester, pour chaque $1 \leq p \leq |w|/2$, si p divise $|w|$, et si oui on fait le test suivant : pour chaque $1 \leq i \leq |w| - p$, vérifier si $w_i = w_{i+p}$. Il est clair que ces tests réussissent pour un p si et seulement si on peut écrire w comme la puissance d'un mot de longueur $|w|/p$.

La complexité en espace est constante (en plus du mot w). La complexité en temps est en $O(|w|^2)$: le seul point possiblement litigieux serait le test de divisibilité, mais son temps d'exécution est clairement dominé par $O(|w|)$.

Question 2. On peut prendre par exemple $(aa)^*$, ou $(aa)(aa)^*$ si on veut éviter de parler du mot vide.

Question 3. On peut prendre par exemple ab^+ : il est clair que tout mot w de ce langage est primitif parce qu'il contient un seul a, et si on avait une écriture $w = u^p$ avec $p > 1$ le mot u devrait contenir $1/p$ occurrences de a, or ce nombre doit être entier, contradiction.

Question 4. Le langage régulier ab^* est infini mais n'est pas continuable (prendre $u := b$).

Il existe des langages réguliers continuables dont le complémentaire est infini, par exemple le langage des mots de longueur paire.

Question 5. Il suffit d'observer la caractérisation suivante : un automate déterministe complet représente un langage continuable si et seulement si tous ses états accessibles sont co-accessibles, c'est-à-dire que tout état accessible par un chemin depuis l'état initial a un chemin vers un état final. Cette propriété peut être testée en temps et mémoire linéaire en la taille de l'automate : on effectue d'abord un parcours en largeur depuis l'état initial pour identifier les états accessibles, on effectue un second parcours en largeur depuis les états finaux en suivant l'inverse des transitions pour identifier les états co-accessibles, et on vérifie que tous les états accessibles sont bien co-accessibles. (En particulier, si l'automate n'était pas complet initialement, alors il n'est pas continuable, sauf si le puits que l'on a ajouté n'est pas accessible, i.e., aucun des états auquel il manquait des transitions sortantes était accessible.)

Démontrons que cette caractérisation est correcte. Supposons d'abord que A ait un état q accessible mais non-co-accessible. Soit $u \in \Sigma^*$ l'étiquette d'un chemin de l'état initial de A à q. On sait alors qu'il n'existe pas de $v \in \Sigma^*$ tel que $uv \in L$, puisqu'un tel v témoignerait de l'existence d'un chemin de q à un état final de A. Ainsi, u est un contre-exemple au caractère continuable de $L(A)$.

Réciproquement, supposons que tous les états accessibles de A soient co-accessibles. Soit $u \in \Sigma^*$ quelconque, et soit q l'état auquel A aboutit après lecture de u depuis l'état final (cet état existe parce que l'automate est complet) : l'existence de u montre que q est accessible. Ainsi, q est co-accessible. Soit v l'étiquette d'un chemin de q à un état final de A. On voit alors que uv est l'étiquette d'un chemin de l'état initial de A à un état final de A, c'est-à-dire que $uv \in L(A)$. Ainsi, on a bien montré que $L(A)$ est continuable.

Question 6.

(a) Soient a, b deux lettres distinctes de Σ. Soit L un langage régulier continuable. Posons A un automate déterministe complet tel que $L(A) = L$, et supposons quitte à éliminer les états inaccessibles que tous les états de A sont accessibles. Comme $L(A)$ est un langage continuable, on sait que A satisfait le critère de la question 5 : plus précisément, comme tous les états de A sont accessibles, A doit être complet et tous ses états doivent être co-accessibles.

Soit n le nombre d'états de A. Pour chaque $i > 0$, écrivons $w'_i := ba^i$, écrivons q_i l'état auquel aboutit l'automate en lisant w'_i depuis son état initial (cet état existe car A est complet), et soit $x_i \in \Sigma^*$ un mot qui étiquette un chemin allant de l'état q_i à un état final de A : un tel chemin existe car tous les états de A sont co-accessibles, et on peut clairement toujours assurer que $|x_i| < n$. Pour tout $i > 0$ on définit $w_i := w'_i x_i$. Par définition, on a $w_i \in L(A)$ pour tout $i > 0$. Montrons à présent que, pour tout $i \geq n - 1$, le mot w_i est primitif, ce qui suffit à conclure la première partie de l'énoncé. En effet, procédons par l'absurde et supposons qu'on puisse écrire $w_i = u^p$ avec $p > 1$. On sait alors que la première lettre de w_i, qui est un b, est la même que la lettre $1 + |w_i|/p$: cette quantité est $\leq 1 + |w_i|/2$. Or on sait que toutes les lettres de w_i aux positions $\{2, \ldots, i + 1\}$ sont des lettres de w'_i qui sont des a, donc il faut que $1 + |w_i|/2$ soit $\geq i + 2$, i.e., que $|w_i| \geq 2(i + 1)$. Mais on sait que $|w_i| = 1 + i + |x_i|$, donc il faut que $|x_i| \geq i + 1$. Mais on a $|x_i| < n \leq i + 1$, donc on a une contradiction. Ainsi, le mot w_i est bien primitif.

(b) La construction de (a) montre en particulier que le mot w_{n-1} est primitif, et sa longueur est de $1 + (n - 1) + (n - 1)$ au plus, i.e., $2n - 1$. Autrement dit, tout langage régulier continuable L contient un mot primitif de longueur $\leq 2n - 1$, où n est le nombre d'états d'un automate déterministe reconnaissant L.

(c) La réciproque est fausse, car le langage ba^+ contient une infinité de mots primitifs mais n'est pas continuable.

[Les questions (a) et (b) sont une adaptation de la preuve de la Proposition 5 de [IKSY88].]

Références

[IKSY88] M Ito, M Katsura, HJ Shyr, and SS Yu. Automata accepting primitive words. In *Semigroup Forum*, volume 37, pages 45–52. Springer, 1988. `https://link.springer.com/article/10.1007/BF02573122`.