Ambiguïté des automates finis

Épreuve pratique d'algorithmique et de programmation
Concours commun des Écoles normales supérieures

Durée de l'épreuve : 3 heures 30 minutes

Juin 2025

ATTENTION !

N'oubliez en aucun cas de recopier votre u_0
à l'emplacement prévu sur votre fiche réponse

Important.

Il vous a été donné un numéro u_0 qui identifie les fichiers d'entrée pour vos codes. Ceux-ci se trouvent dans un répertoire `data/u0/`, où `u0` est remplacé par votre numéro u_0. Les réponses attendues sont généralement courtes et doivent être données sur la fiche réponse fournie à la fin du sujet. À la fin du sujet, vous trouverez en fait deux fiches réponses. La première est un exemple des réponses attendues pour le numéro particulier $\widetilde{u_0}$. Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes. Vous indiquerez vos réponses (correspondant à votre u_0) sur la seconde et vous la remettrez à l'examinateur à la fin de l'épreuve.

En ce qui concerne la partie orale de l'examen, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures !

Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, par exemple : $O(n^2)$, $O(n \log n)$, etc. La lecture de l'instance par le code qui vous a été fourni ne sera pas prise en compte dans la complexité.

Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de ***tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe.***

Enfin, il est recommandé de lire l'intégralité du sujet avant de commencer afin d'effectuer les bons choix de structures de données dès le début.

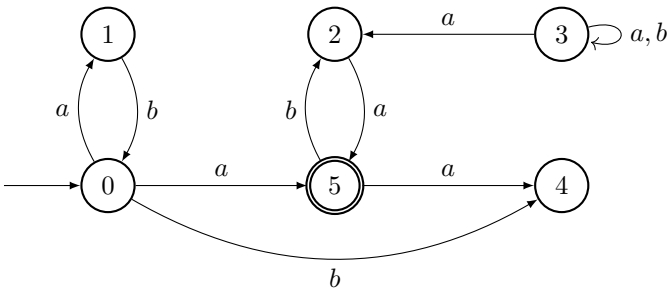
La Partie 1 doit être implémentée en C (à l'aide du fichier `base.c`), et la Partie 2 doit être implémentée en OCaml (à l'aide du fichier `base.ml`). Ces deux parties sont indépendantes.
Les deux fichiers susmentionnés vous sont fournis dans un dossier de base. Ce dossier contient également un sous-dossier `data/` dans lequel se trouvent les jeux de tests pour la partie OCaml.
Il est recommandé de garder une sauvegarde de tous les fichiers fournis, au cas où ceux-ci seraient modifiés par erreur.
Il est demandé de nous fournir sur votre clef USB vos fichiers sources. Ces consignes doivent impérativement être suivies.

Introduction

Dans tout ce sujet on fixe l'alphabet fini $\Sigma = \{a, b\}$. Pour un mot $w \in \Sigma^*$, on note $|w|$ sa longueur et $w[i]$ sa i-ème lettre pour $i \in \{0, 1, \dots, |w| - 1\}$ (ainsi, le mot commence par la lettre $w[0]$).
Rappelons qu'un automate fini sur Σ consiste en :

- un ensemble fini d'états Q – dans nos implémentations en C et en OCaml, il sera toujours de la forme $Q = \{0, 1, \dots, N - 1\}$, et on se contentera donc de spécifier dans la donnée de l'automate le nombre N d'états ;
- un sous-ensemble d'états initiaux $I \subseteq Q$, qui vaudra toujours $I = \{0\}$ ici ;
- et un sous-ensemble d'états finaux $F \subseteq Q$, qui vaudra toujours $F = \{N - 1\}$ ici ;
- un ensemble de transitions $\Delta \subseteq Q \times \Sigma \times Q$.

Par exemple, le schéma ci-dessous représente un automate avec $N = 6$ états et les transitions $\Delta = \{(0, a, 1), (0, b, 4), (0, a, 5), (1, b, 0), (2, a, 5), (3, a, 2), (3, a, 3), (3, b, 3), (5, b, 2), (5, a, 4)\}$:



On note $p \xrightarrow{c} q$ pour $(p, c, q) \in \Delta$ lorsque $c \in \{a, b\}$. Un chemin d'un état p à un état q étiqueté par un mot $w \in \Sigma^*$ est une suite d'états $q_0, \dots, q_{|w|}$ telle que :

- $q_0 = p$ et $q_{|w|} = q$;
- $q_i \xrightarrow{w[i]} q_{i+1}$ pour tout $i \in \{0, 1, \dots, |w| - 1\}$.

Ce chemin est acceptant lorsque p est l'état initial et q est l'état final.
On appelle ambiguïté d'un automate sur un mot donné le nombre de chemins acceptants. Par exemple, l'automate dessiné ci-dessus a pour ambiguïté $n+1$ sur $(ab)^n a$ car les chemins acceptants sont de la forme suivante, pour $0 \leqslant i \leqslant n$:

$$\underbrace{0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{a} \cdots \xrightarrow{b} 0}_{i \text{ passages par l'état } 1} \xrightarrow{a} \underbrace{5 \xrightarrow{b} 2 \xrightarrow{a} 5 \xrightarrow{b} \cdots \xrightarrow{a} 5}_{n - i \text{ passages par l'état } 2}$$

En particulier, il y a exactement un chemin acceptant qui ne passe pas par 1 (pour $i = 0$), et un chemin acceptant qui ne passe pas par 2 (pour $i = n$).
Si on supprime la transition $2 \xrightarrow{a} 5$, alors on obtient un autre automate reconnaissant le même langage $(ab)^* a$ mais non ambigu : son ambiguïté est d'au plus 1 quel que soit le mot d'entrée.

1 Partie C

Code fourni Le fichier `base.c` contient un ensemble d'inclusions usuelles (`<stdbool.h>`, `<stdlib.h>`, `<stdio.h>`, ...) en en-tête, suivi d'une déclaration de structure définissant notre type de données pour les automates :

```
struct automate {
    int nb_etats;
    bool *trans_a;
    bool *trans_b;
};
typedef struct automate automate;
```

Les pointeurs `trans_a` et `trans_b` pointent vers des tableaux de taille $N \times N$ où $N =$ `nb_etats`. La valeur du booléen `A.trans_a[i*A.nb_etats + j]` indique si $i \xrightarrow{a} j$, et de même pour `trans_b` et les transitions étiquetées par la lettre b. L'état initial est 0 et l'état final est $N-1$. À titre d'exemple, la variable globale `automate A_ex` dans `base.c` est initialisée à l'encodage de l'automate dessiné dans l'introduction.
De plus, le fichier `base.c` fournit également :

- une fonction `automate gen_automate(int u0, int N)` renvoyant un automate à N états généré pseudo-aléatoirement à partir de la graine u_0 ;
- une fonction `void free_automate(automate A)` permettant de libérer la mémoire dynamiquement allouée lors de la création d'un automate par la fonction précédente ;
- les prototypes des fonctions à implémenter pour les questions qui suivent.

Étant donné un nombre N d'états, on dit qu'un tableau de booléens `etats` de taille N représente un sous-ensemble $E \subseteq Q = \{0, 1, \ldots, N-1\}$ lorsque $E = \{i \in Q \mid$ `etats[i]` vaut `true`$\}$.

Question 1 *Implémentez une fonction*

```
void transition(automate A, char c, bool *etats_avant, bool *etats_apres)
```

qui écrit dans le tableau `etats_apres` *— qu'on suppose de taille* `A.nb_etats` *et déjà alloué par le code appelant cette fonction — sans modifier* `etats_avant` *(également de taille* `A.nb_etats`*), de sorte que :*

$$\texttt{etats\_avant} \text{ représente } E \quad \Longrightarrow \quad \texttt{etats\_apres} \text{ représente } \{j \mid \exists i \in E : i \xrightarrow{c} j\}$$

À l'aide de la fonction `transition`*, calculez*

$$\sum_{j=0}^{N-1} x_j 2^j \quad \text{où} \quad x_j = \begin{cases} 1 & \text{si } \exists i \in \{0,1,2,3,4\} : i \xrightarrow{c} j \text{ dans } \texttt{gen\_automate(u0,N)} \\ 0 & \text{sinon} \end{cases}$$

pour votre valeur donnée de u_0 ainsi que les choix des paramètres `c` *et N suivants :*

a) $c = a$, $N = 14$ ***b)*** $c = b$, $N = 15$

Question à développer pendant l'oral 1 *Quelle est la complexité en temps de votre algorithme pour la fonction* `transition` *? Justifiez.*

Dans les questions suivantes, on vous promet que les mots en entrée sont dans $\{a, b\}^*$.

Question 2 *Implémentez une fonction*

```
bool accepte(automate A, char *mot)
```

qui détermine si l'automate accepte le mot. On utilise la représentation habituelle des chaînes de caractères en C avec le caractère nul `'\0'` *comme marqueur de fin de chaîne.*

Indication. *Idéalement, votre solution devrait effectuer $O(1)$ appels à* `malloc` *et ne pas allouer de tableaux de taille variable sur la pile.*
Pour chacun des mots $w \in \{a,b\}^*$ suivants (où on note u^n pour la répétition n fois du mot u), calculer le cardinal de $\{N \in \{50,51,\ldots,99\}$: `gen_automate(u0,N)` accepte $w\}$:

a) $w = abba$ ***b)*** $w = a^{100}b^{100}$ ***c)*** $w = (ba)^{100}$

Question à développer pendant l'oral 2 *Expliquez brièvement votre implémentation de la fonction* `accepte`*, en donnant sa complexité en temps et sa complexité en espace.*

Question 3 *Implémentez une fonction*

```
int ambiguite(automate A, char *mot)
```

qui calcule l'ambiguïté de l'automate sur le mot (telle que définie dans l'introduction) modulo 1000003 (10^6+3), ceci afin d'éviter les débordements d'entiers.
Calculez l'ambiguïté de `gen_automate(u0,1000)` *sur chacun des mots $w \in \{a,b\}^*$ suivants :*

a) $w = abba$ ***b)*** $w = a^{100}b^{100}$ ***c)*** $w = (ba)^{100}$

Question à développer pendant l'oral 3 *Expliquez votre algorithme.*

Question 4 *Implémentez une fonction*

```
int maxamb(automate A, int longueur_max)
```

qui calcule $\max\{$*ambiguïté de $\mathcal{A}$ sur $w \in \{a,b\}^*$: $|w| \leqslant$* `longueur_max`$\}$*, en supposant que cette valeur ne dépasse pas 10^6 (cette supposition sera vérifiée dans les cas d'application ci-dessous).*

Indication. *Une recherche exhaustive suffit ici. Comme précédemment, on veillera à limiter le nombre d'allocations mémoire.*
Calculez les valeurs obtenues pour `A = gen_automate(u0,150)` *et :*

a) `longueur_max = 5` ***b)*** `longueur_max = 10`

Question à développer pendant l'oral 4 *Expliquez brièvement votre implémentation.*

2 Partie OCaml

Rappel sur le module Queue La bibliothèque standard d'OCaml fournit un type `Queue.t` de files mutables (FIFO), prenant en charge notamment les fonctions :

```
Queue.create : unit -> 'a Queue.t          Queue.is_empty : 'a Queue.t -> bool
Queue.push : 'a -> 'a Queue.t -> unit             Queue.pop : 'a Queue.t -> 'a
```

Code fourni Le fichier `base.ml` contient les éléments de code suivants :

- La déclaration de type synonyme `type automate = (int list * int list) array` : il s'agit d'une représentation par liste d'adjacence, c'est-à-dire qu'un tableau `aut` de ce type et de taille N représente l'automate dont
  - les états sont $\{0, \dots, N-1\}$, l'état initial est 0 et l'état final est $N-1$,
  - $i \xrightarrow{a} j$ lorsque j apparaît dans la liste `fst aut.(i)`,
  - $i \xrightarrow{b} j$ lorsque j apparaît dans la liste `snd aut.(i)`.
- Une définition d'une valeur `a_ex : automate` représentant l'automate de l'introduction.
- Une fonction `read_automate : string -> automate` qui lit un automate à partir d'un fichier dont l'adresse est passée en argument. Elle servira à lire les instances de test stockées dans le dossier `data`. Votre évaluation portera sur les fichiers dans `data/`u_0`/` avec la valeur de u_0 qui vous a été fournie. Les autres fichiers (notamment dans `data/`$\widetilde{u_0}$`/` qui correspond à la fiche réponse type) peuvent être utilisés pour tester vos programmes.
- Une fonction de hachage `hash_ioa : int option array -> int` qui sera utilisée dans certaines questions pour produire les valeurs entières à écrire sur votre fiche réponse.

Cette partie se compose de deux sous-parties indépendantes, reposant toutes deux sur `base.ml` :

- En §2.1 on cherche à tester efficacement si un automate est ambigu ou non.
- En §2.2 on étudie le comportement asymptotique de l'ambiguïté.

2.1 Vers un test d'ambiguïté efficace

On dit qu'un état q est accessible lorsqu'il y a un chemin menant de l'état initial (ici 0) à q. En particulier, l'état initial est lui-même accessible (via un chemin étiqueté par le mot vide). Dans l'automate dessiné en introduction, le seul état non accessible est 3.

Question 5 *Implémentez une fonction*

```
accessibles : automate -> int option array
```

qui calcule un tableau associant à chaque état i :

- *si i est accessible,* `Some`*(longueur minimale d'un chemin de 0 à i) ;*
- *sinon,* `None`*.*

La taille du tableau doit donc être le nombre d'états de l'automate.
Calculez `hash_ioa (accessibles (read_automate f))` *pour* `f` *valant :*

a) `data/`u_0`/petit.txt` ***b)*** `data/`u_0`/moyen.txt` ***c)*** `data/`u_0`/gros.txt`

Question à développer pendant l'oral 5 *Quelle est la complexité en temps de votre algorithme, en fonction du nombre $N = |Q|$ d'états et du nombre $|\Delta|$ de transitions ? (Dans les questions ultérieures de complexité, on attendra aussi une réponse en fonction de $|Q|$ et $|\Delta|$.)*

La question 6 est indépendante du reste du sujet. C'est une variante de la question 5 avec un critère d'optimisation différent : au lieu de minimiser la longueur d'un chemin, on veut minimiser le nombre d'occurrences de la lettre b dans le mot qui l'étiquette. Ainsi, dans l'automate de l'introduction, pour atteindre l'état 4 à partir de l'état initial 0 :

- le chemin le plus court est $0 \xrightarrow{b} 4$;
- le chemin minimisant le nombre de b est $0 \xrightarrow{a} 5 \xrightarrow{a} 4$.

Question 6 *Implémentez une fonction*

```
accessibles_min_b : automate -> int option array
```

qui calcule un tableau associant à chaque état i :
- *si i est accessible,* `Some`*(nombre minimal de b d'un chemin de 0 à i) ;*
- *sinon,* `None`*.*

Calculez `hash_ioa (accessibles_min_b (read_automate f))` *pour* `f` *valant :*

a) `data/`u_0`/petit.txt` **b)** `data/`u_0`/moyen.txt` **c)** `data/`u_0`/gros.txt`

Question à développer pendant l'oral 6 *Expliquez votre algorithme et donnez (en justifiant) sa complexité en temps en fonction de $|Q|$ et $|\Delta|$.*

On dit qu'un état q est utile lorsqu'il y a un chemin acceptant qui passe par q. Cela revient à dire que q est accessible et qu'il y a un chemin de q à l'état final. Dans l'automate de l'introduction, les états utiles sont 0, 1, 2 et 5.

Question 7 *En réutilisant la fonction* `accessibles`*, implémentez une fonction*

```
utiles : automate -> int option array
```

qui calcule un tableau associant à chaque état i :
- *si i est utile,* `Some`*(longueur minimale d'un chemin acceptant passant par i) ;*
- *sinon,* `None`*.*

La taille du tableau doit donc être le nombre d'états de l'automate.
Calculez `hash_ioa (utiles (read_automate f))` *pour* `f` *valant :*

a) `data/`u_0`/petit.txt` **b)** `data/`u_0`/moyen.txt` **c)** `data/`u_0`/gros.txt`

Question à développer pendant l'oral 7 *Expliquez votre algorithme, en précisant s'il suffit ou non de remplacer les appels à* `accessibles` *par* `accessibles_min_b` *pour calculer, pour chaque état, le nombre minimal de b dans un chemin acceptant passant par cet état.*

Question 8 *Implémentez une fonction*

```
ambigu : automate -> int option
```

qui renvoie :
- `Some`*(longueur minimale d'un mot qui a une ambiguïté $\geqslant 2$) s'il existe un tel mot ;*
- *sinon (ce qui signifie que l'automate est non ambigu),* `None`*.*

Indication. *On pourra exploiter le résultat de la fonction* `utiles` *sur un automate judicieusement construit à partir de l'automate d'entrée.*
Calculez `ambigu (read_automate f)` *pour* `f` *valant :*

a) `data/`u_0`/q8a.txt` **b)** `data/`u_0`/q8b.txt`

c) `data/`u_0`/q8c.txt` **d)** `data/`u_0`/q8d.txt`

Question à développer pendant l'oral 8 *Expliquez brièvement votre algorithme. Quelle est sa complexité en temps ?*

2.2 Degré asymptotique d'ambiguïté d'un automate

Cette dernière partie vise à calculer le degré $\deg(\mathcal{A}) \in \mathbb{N} \cup \{+\infty\}$ d'un automate fini $\mathcal{A}$. Nous en donnerons plus loin (après la question 10) une définition combinatoire. L'intérêt principal de la notion de degré réside dans le théorème suivant (admis ici, et qui ne sera pas nécessaire pour traiter le sujet) :

$$\underbrace{\mathrm{maxamb}(\mathcal{A}, n)}_{\text{cf. question 4}} = \begin{cases} \Theta(n^{\deg(\mathcal{A})}) & \text{si } \deg(\mathcal{A}) \in \mathbb{N} \\ 2^{\Theta(n)} & \text{si } \deg(\mathcal{A}) = +\infty \end{cases} \qquad \text{lorsque } n \to +\infty$$

(rappel : $f(n) = \Theta(g(n))$ signifie que $f(n) = O(g(n))$ et $g(n) = O(f(n))$). Remarquons que cela signifie, en particulier, que maxamb croît toujours soit polynomialement, soit exponentiellement.

Indication. Il est possible de les traiter les questions 9 et 10 indépendamment ; elles seront ensuite utilisées ensemble dans le calcul de $\deg(\mathcal{A})$. Cela dit, leurs solutions sont susceptibles de faire intervenir des algorithmes communs, et donc de se prêter à de la réutilisation de code. Ces questions admettent toutes les deux des solutions en temps linéaire, c'est-à-dire $O(|Q| + |\Delta|)$, mais tout algorithme en temps polynomial qui fonctionne sur les exemples de taille moyenne suffira à rapporter une partie des points.

On appelle $a\Sigma^*$-cycle en un état q un chemin de q vers lui-même sur un mot du langage $a\Sigma^*$, c'est-à-dire commençant par un a (rappel : $\Sigma = \{a, b\}$ est notre alphabet) : $q \xrightarrow{a} q' \to \cdots \to q$. Par exemple dans l'automate de l'introduction on a notamment :

— en 0, un $a\Sigma^*$-cycle $0 \xrightarrow{a} 1 \xrightarrow{b} 0$;
— en 3, un $a\Sigma^*$-cycle $3 \xrightarrow{a} 3 \xrightarrow{a} 3 \xrightarrow{a} 3$.

Question 9 *Implémentez une fonction*

```
a_cycles : automate -> bool array
```

qui renvoie un tableau indiquant, pour chaque état i, s'il existe un $a\Sigma^*$-cycle en i.
Pour les valeurs suivantes de `f`*, calculez :*

```
hash_ioa (Array.map (fun x -> if x then Some 1 else None)
                    (a_cycles (read_automate f)))
```

a) `data/`u_0`/petit.txt` ***b)*** `data/`u_0`/moyen.txt` ***c)*** `data/`u_0`/gros.txt`

Question à développer pendant l'oral 9 *Expliquez votre algorithme et sa complexité en temps.*

Question 10 *Implémentez une fonction*

```
max_b : automate -> int option
```

qui renvoie :
— *si l'automate n'admet pas de chemin acceptant,* `None` *;*
— *si le nombre de b dans un chemin acceptant est non borné,* `None` *;*
— *sinon,* `Some(`*nombre maximal de b dans un chemin acceptant*`)`*.*
Calculez `max_b (read_automate f)` *pour* `f` *valant :*

a) `data/`u_0`/q10a.txt` ***b)*** `data/`u_0`/q10b.txt`
c) `data/`u_0`/q10c.txt` ***d)*** `data/`u_0`/q10d.txt`

Question à développer pendant l'oral 10 *Expliquez votre algorithme et sa complexité.*

Nous en venons maintenant à la définition du degré d'un automate.

— Tout d'abord, étant donné un automate $\mathcal{A}$ avec un ensemble Q d'états, on définit un autre automate $\mathcal{A}'$ comme suit :
- les états de $\mathcal{A}'$ sont les triplets d'états de $\mathcal{A}$ (l'ensemble d'états Q^3 de $\mathcal{A}'$ n'est donc pas de la forme $\{0,\dots,N-1\}$ mais on décrit ici une construction abstraite) ;
- $(p,q,q) \xrightarrow{a} (p,p,q)$ pour toute paire $(p,q) \in Q^2$ telle que $p \neq q$;
- $(p_1,p_2,p_3) \xrightarrow{b} (q_1,q_2,q_3)$ dans $\mathcal{A}' \iff \exists c \in \{a,b\} : \forall i \in \{1,2,3\},\ p_i \xrightarrow{c} q_i$ dans $\mathcal{A}$;
- l'état initial et l'état final n'ont pas d'importance car on ne s'intéresse qu'aux cycles.

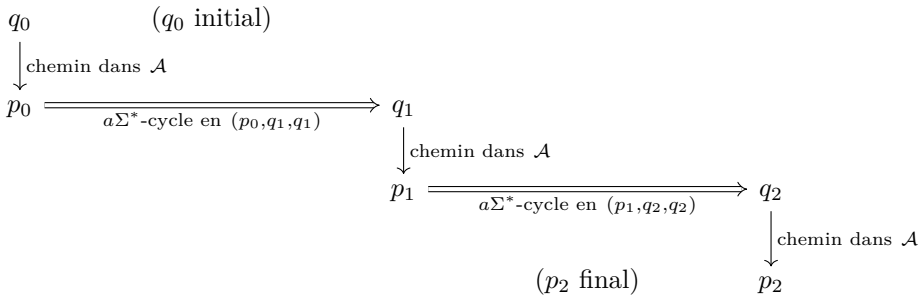
Par exemple, dans $(\mathcal{A}_{\text{ex}})'$ où $\mathcal{A}_{\text{ex}}$ est l'automate donné dans l'introduction, on a :

$$(0,2,2) \xrightarrow{a} \underbrace{(0,0,2) \xrightarrow{b} (1,5,5)}_{\text{provient d'un triplet de } a\text{-transitions (et non de } b\text{-transitions) dans } \mathcal{A}_{\text{ex}}} \xrightarrow{b} (0,2,2)$$

— À partir de là, on peut définir l'ensemble $K(\mathcal{A})$ des $k \in \mathbb{N}$ pour lesquels il existe une suite de $2k+2$ états $q_0, p_0, \dots, q_k, p_k$ dans $\mathcal{A}$ telle que :
- q_0 est initial et p_k est final dans $\mathcal{A}$;
- pour $i \in \{0,\dots,k\}$, il existe un chemin de q_i à p_i dans $\mathcal{A}$;
- pour $i \in \{0,\dots,k-1\}$, il existe un $a\Sigma^*$-cycle en (p_i, q_{i+1}, q_{i+1}) dans $\mathcal{A}'$.

Par exemple, $1 \in K(\mathcal{A}_{\text{ex}})$ car pour $q_0 = 0$ initial, $p_0 = 0$, $q_1 = 2$ et $p_1 = 5$ final, on a :
- deux chemins $0 \xrightarrow{\text{mot vide}} 0$ et $2 \xrightarrow{aba} 5$ (par exemple) dans $\mathcal{A}_{\text{ex}}$;
- un $a\Sigma^*$-cycle en $(0,2,2)$ dans $(\mathcal{A}_{\text{ex}})'$, déjà donné plus haut.

Le diagramme ci-dessous illustre la situation où $2 \in K(\mathcal{A})$ — et on peut vérifier que cette situation ne se produit pas pour $\mathcal{A}_{\text{ex}}$ (en fait, $K(\mathcal{A}_{\text{ex}}) = \{0,1\}$) :



— Enfin, on définit le degré de l'automate $\mathcal{A}$ comme suit (notamment, $\deg(\mathcal{A}_{\text{ex}}) = 1$) :

$$\deg(\mathcal{A}) = \sup(K(\mathcal{A}) \cup \{0\}) \in \mathbb{N} \cup \{+\infty\}$$

Question 11 *En utilisant cette définition et les fonctions* `a_cycles` *et* `max_b`*, implémentez*

```
degre : automate -> int option
```

qui renvoie **`Some`***(degré de l'automate) si le degré est fini, ou bien* **`None`** *sinon.*
Calculez `degre (read_automate f)` *pour* `f` *valant :*

a) `data/`u_0`/q11a.txt` ***b)*** `data/`u_0`/q11b.txt`

c) `data/`u_0`/q11c.txt` ***d)*** `data/`u_0`/q11d.txt`

Fiche réponse type : Ambiguïté des automates finis

$\widetilde{u_0}$: 0

Question 1

a) 80

b) 4 676

Question 2

a) 11

b) 32

c) 29

Question 3

a) 483

b) 833 631

c) 438 370

Question 4

a) 17

b) 15 816

Question 5

a) 3 306

b) 9 108

c) 792

Question 6

a) 4 326

b) 4 638

c) 1 164

Question 7

a) 129

b) 1 339

c) 9 102

Question 8

a) None

b) Some 14

c) Some 18

d) Some 20

Question 9

a) 3 685

b) 5 023

c) 4 383

Question 10

a) Some 67

b) Some 71

c) None

d) Some 39029

Question 11

a) Some 5

b) Some 2

c) None

d) Some 6

Fiche réponse : Ambiguïté des automates finis

Nom, prénom, u_0 : .....

Question 1

a)

b)

Question 2

a)

b)

c)

Question 3

a)

b)

c)

Question 4

a)

b)

Question 5

a)

b)

c)

Question 6

a)

b)

c)

Question 7

a)

b)

c)

Question 8

a)

b)

c)

d)

Question 9

a)

b)

c)

Question 10

a)

b)

c)

d)

Question 11

a)

b)

c)

d)