Automates, points-fixes, et équivalence de langage

Le sujet comprend douze pages, numérotées de 1 à 12.

Début de l'épreuve.

La première partie introduit les automates et permet de se familiariser avec la notion d'équivalence de langage entre états d'un automate. Un algorithme pour vérifier l'équivalence de deux états y est proposé.

La seconde partie met en place des outils (théorèmes de point fixe, clôtures) qui permettent dans la troisième partie de prouver la correction de cet algorithme, puis d'en proposer des optimisations.

La dernière partie est plus courte, on y considère la question du calcul de la relation d'équivalence entre états dans sa globalité, et on y utilise les outils de la seconde partie pour proposer un algorithme.

Les parties 1 et 2 sont indépendantes ; il est conseillé de les traiter en premier car les parties 3 et 4 en dépendent. Il est permis d'admettre les réponses à certaines questions pour répondre aux suivantes.

1 Automates

On fixe un ensemble fini A appelé *alphabet*, dont les éléments sont appelés *lettres*. Un *mot* est une suite finie de lettres. Un *language* est un ensemble de mots. L'ensemble de tous les mots est noté A^*. On utilise les variables $a, b, c\ldots$ pour dénoter les lettres et les variables $u, v, w\ldots$ pour dénoter les mots. Le mot vide est noté ϵ ; la concaténation de deux mots u, v est notée uv ; une lettre sera parfois confondue avec le mot d'une lettre correspondant. Par exemple, au est le mot constitué de la lettre a suivie du mot u.

Un *automate (déterministe)* est un triplet $\langle X, o, \delta\rangle$ où :

— X est un ensemble d'*états* ;

— $o : X \to \{0, 1\}$ est une fonction indiquant pour chaque état, s'il est acceptant (1) ou non (0) ;

— $\delta : X \times A \to X$ est la fonction de transition, totale, indiquant vers quel état se déplacer lors de la lecture d'une lettre dans un état donné.

Notons que nous faisons le choix de ne pas désigner un état initial ; étant donné un automate, nous nous intéresserons aux langages reconnus à partir de chacun des états de cet automate. On représente un automate graphiquement comme ci-dessous sur la gauche, en utilisant des cercles pour dénoter les états, à bordures doubles quand ils sont acceptants, et des flèches étiquetées par

2.1 Plus grand point fixe

On va démontrer que toute fonction croissante admet un plus grand point fixe. On fixe pour cela une fonction croissante $f : \mathcal{P}(E) \to \mathcal{P}(E)$.

Question 2.5 *Montrer que l'union d'une famille de post-points fixes est un post-point fixe.*

Soit νf l'union de tous les post-points fixes de f :

$$\nu f \stackrel{\text{def}}{=} \bigcup_{X \subseteq f(X)} X$$

Par la question précédente, νf est un post-point fixe : $\nu f \subseteq f(\nu f)$.

Question 2.6 *(i) Montrer que νf est aussi un pré-point fixe.*
(ii) En déduire que c'est le plus grand point fixe (au sens de l'inclusion).

Question 2.7 *Montrer que f admet aussi un plus petit point fixe.*

Si l'on suppose de plus que la fonction f est co-continue, on peut caractériser son plus grand point fixe autrement. Remarquons tout d'abord que $(f^i(E))_{i\in\mathbb{N}}$ est une suite de parties de E.

Question 2.8 *(i) Montrer que la suite $(f^i(E))_{i\in\mathbb{N}}$ est décroissante.*
(ii) Montrer que si f est co-continue, alors $\nu f = \bigcap_{i\in\mathbb{N}} f^i(E)$.

2.2 Plus petite clôture

On ordonne les fonctions point à point : *f est contenue dans g*, noté $f \subseteq g$, si pour tout $X \subseteq E$, $f(X) \subseteq g(X)$.

Une fonction f est :

— *extensive* si $\mathrm{id} \subseteq f$ (c'est à dire, $\forall X, X \subseteq f(X)$) ;

— *saturante* si $f \circ f \subseteq f$ (c'est à dire, $\forall X, f(f(X)) \subseteq f(X)$) ;

— une *clôture* si elle est croissante, extensive, et saturante.

Etant donnée une fonction f, on pose $f^\omega \stackrel{\text{def}}{=} \bigcup_{i\in\mathbb{N}} (f \cup \mathrm{id})^i$.

Question 2.9 *(i) Montrer que f^ω est extensive et contient f.*
(ii) Montrer que si f est continue, alors f^ω est une clôture.

Question 2.10 *Montrer que si f est continue, alors pour tout X, $f^\omega(X)$ est le plus petit pré-point fixe de f contenant X.*

Question 2.11 *Déduire des questions précédentes que lorsque f est continue, f^ω est la plus petite clôture contenant f.*

On s'intéresse aux fonctions de $\mathcal{P}(E)$ dans lui-même. On note id la fonction identité, et $f \circ g$ la composition de deux fonctions :

$$\mathrm{id} : X \mapsto X \qquad\qquad f \circ g : X \mapsto f(g(X))$$

On définit également la suite $(f^i)_{i\in\mathbb{N}}$ d'itérées d'une fonction f, par récurrence sur i :

$$f^0 \stackrel{\text{def}}{=} \mathrm{id}$$
$$f^{i+1} \stackrel{\text{def}}{=} f \circ f^i$$

Étant données deux fonctions $f, g : \mathcal{P}(E) \to \mathcal{P}(E)$, on note $f \cup g$ et $f \cap g$ les fonctions

$$f \cup g : X \mapsto f(X) \cup g(X) \qquad\qquad f \cap g : X \mapsto f(X) \cap g(X)$$

Plus généralement, étant donnée une famille $(f_i)_{i\in I}$ de fonctions, on note $\bigcup_{i\in I} f_i$ et $\bigcap_{i\in I} f_i$ les fonctions

$$\bigcup_{i\in I} f_i : X \mapsto \bigcup_{i\in I} f_i(X) \qquad\qquad \bigcap_{i\in I} f_i : X \mapsto \bigcap_{i\in I} f_i(X)$$

Une fonction $f : \mathcal{P}(E) \to \mathcal{P}(E)$ est dite

- *croissante* si pour tous $X, Y \subseteq E$ tels que $X \subseteq Y$, on a $f(X) \subseteq f(Y)$.
- *continue* si pour toute suite croissante $(X_i)_{i\in\mathbb{N}}$ de parties de E, on a $f\left(\bigcup_{i\in\mathbb{N}} X_i\right) = \bigcup_{i\in\mathbb{N}} f(X_i)$.
- *co-continue* si pour toute suite décroissante $(X_i)_{i\in\mathbb{N}}$ de parties de E, on a $f\left(\bigcap_{i\in\mathbb{N}} X_i\right) = \bigcap_{i\in\mathbb{N}} f(X_i)$.

La fonction identité est trivialement croissante, continue, et co-continue. De même, la composée de deux fonctions croissantes (resp. continues, co-continues) est croissante (resp. continue, co-continue), et l'union d'une famille de fonctions croissantes (resp. continues, co-continues) est croissante (resp. continue, co-continue).

À partir de la question suivante, toutes les réponses doivent être soigneusement justifiées.

Question 2.1 *Montrer que toute fonction continue est croissante.*

Question 2.2 *Montrer que toute fonction co-continue est croissante.*

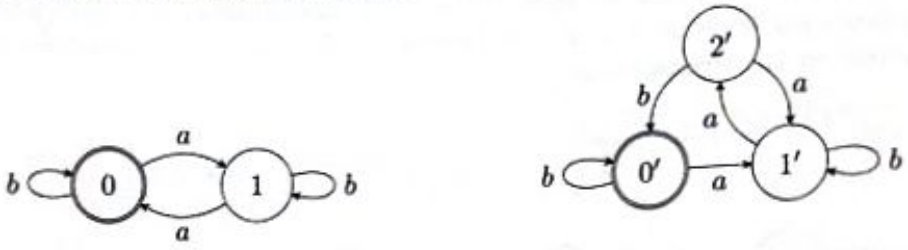
Question 2.3 *Une fonction croissante est-elle toujours continue ? co-continue ? Justifier vos réponses.*

Etant donnée une fonction f, une partie $X \subseteq E$ est :

- un *post-point fixe (de f)* si $X \subseteq f(X)$,
- un *pré-point fixe (de f)* si $f(X) \subseteq X$,
- un *point fixe (de f)* si $X = f(X)$.

Question 2.4 *(i) Montrer que toute fonction admet un post-point fixe et un pré-point fixe.*
(ii) Donner une fonction qui n'admette pas de point fixe.

Nous allons exécuter cet algorithme en partant de la paire d'états $\langle 0, 0' \rangle$ de l'automate suivant :



On représente sur une ligne les valeurs des variables R, $\langle x, y \rangle$, et F au point situé entre les lignes 2.1 et 2.2, en soulignant la paire en cours de traitement, $\langle x, y \rangle$. À la première itération, R est vide et on extrait la paire $\langle 0, 0' \rangle$ de F, qui est donc vide :

$$\underline{\langle 0, 0' \rangle}$$

On vérifie que $o(0) = o(0')$, et on insère les successeurs de la paire $\langle 0, 0' \rangle$ à la fin de F : $\langle 1, 1' \rangle$ selon la lettre a, puis $\langle 0, 0' \rangle$ selon la lettre b. La paire soulignée passe dans R et on souligne la nouvelle paire à traiter :

$$\langle 0, 0' \rangle \ \underline{\langle 1, 1' \rangle} \ \langle 0, 0' \rangle$$

on vérifie que $o(1) = o(1')$, on insère les successeurs de la paire $\langle 1, 1' \rangle$ ($\langle 0, 2' \rangle$ puis $\langle 1, 1' \rangle$) ; la paire soulignée passe dans R et on souligne la suivante :

$$\langle 0, 0' \rangle, \langle 1, 1' \rangle \ \underline{\langle 0, 0' \rangle} \ \langle 0, 2' \rangle, \langle 1, 1' \rangle$$

cette paire est déjà dans R, on la barre et on passe à la suivante :

$$\langle 0, 0' \rangle, \langle 1, 1' \rangle, \cancel{\langle 0, 0' \rangle}, \ \underline{\langle 0, 2' \rangle} \ \langle 1, 1' \rangle$$

on a $o(0) \neq o(2')$, l'algorithme renvoie faux.

Question 1.5 *Exécuter l'algorithme 1 pour la paire d'états $\langle 0, 3 \rangle$ de l'automate de la question 1.3. Utiliser la représentation précédente, et ne donner que la dernière ligne. Faire de même en partant de la paire $\langle 2, 5 \rangle$.*

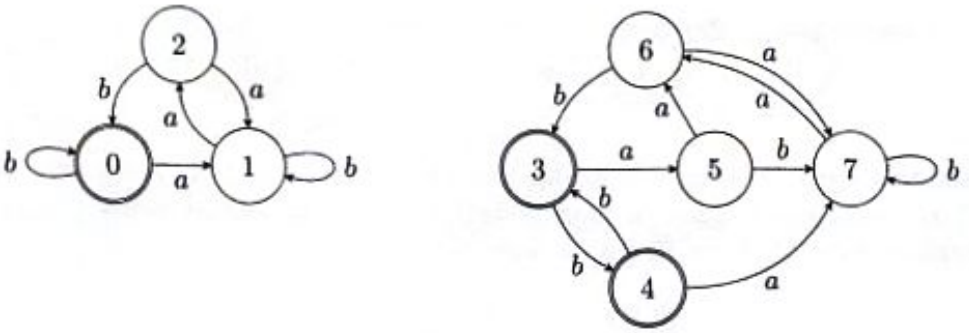
Nous démontrerons la correction de cet algorithme dans la partie 3.

2 Fonctions d'ensembles, de relations

Soit E un ensemble, potentiellement infini. On note $\mathcal{P}(E)$ l'ensemble des parties de E. Notons que pour toutes parties $X, Y \subseteq E$, on a $X \subseteq Y$ si et seulement si $X \cup Y = Y$.

Une suite $(X_i)_{i \in \mathbb{N}}$ de parties de E est dite *croissante* (resp. *décroissante*) si pour tout indice i, $X_i \subseteq X_{i+1}$ (resp. $X_{i+1} \subseteq X_i$).

Question 1.3 *Considérons l'automate suivant. Bien que cet automate contienne deux parties disjointes, on le considère comme un automate à huit états. Donner une description intuitive des langages reconnus par les états 0, 1, et 2 de l'automate. Les états 3 à 7 reconnaissent aussi ces trois langages ; quelle est la correspondance ?*



Deux états x, y sont *équivalents*, noté $x \simeq y$, lorsqu'ils reconnaissent le même langage ($L(x) = L(y)$).

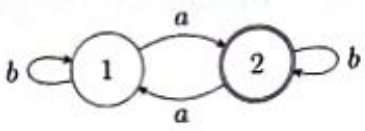
Question 1.4 *Décrire la relation $\simeq$ sur l'ensemble d'états $[0;7]$ pour l'automate de la question 1.3.*

Nous allons étudier l'algorithme 1 ci-dessous, permettant de tester l'équivalence de deux états dans un automate donné. On ignore dans un premier temps les invariants I1 et I2, et I3 qui sera défini à la question 3.7. À la ligne 2.2, la sémantique de l'instruction `continuer` consiste à revenir à l'entrée de la boucle (ligne 2), sans exécuter les lignes 2.3 à 2.5. On suppose que la variable F est implémentée par une file "premier entré premier sorti", et que pour la boucle interne (ligne 2.4), les éléments sont considérés selon l'ordre alphabétique.

Algorithme 1 Un algorithme pour tester l'équivalence de deux états x_0 et y_0 d'un automate $\langle X, o, \delta \rangle$ donné.

```
(1) R := ∅; F := {⟨x0, y0⟩};
(2) tant que F n'est pas vide,
        // I1: ⟨x0, y0⟩ ∈ R ∪ F
        // I2: R ⊆ p(R ∪ F)
        // I3:
 (2.1)  extraire une paire ⟨x, y⟩ de F;
 (2.2)  si ⟨x, y⟩ ∈ R, continuer;
 (2.3)  si o(x) ≠ o(y), renvoyer faux ;
 (2.4)  pour tout a ∈ A, ajouter ⟨δ(x, a), δ(y, a)⟩ à F;
 (2.5)  ajouter ⟨x, y⟩ à R;
(3) renvoyer vrai;
```

les lettres de l'alphabet pour dénoter les transitions. Ici, il s'agit d'un automate dont l'ensemble d'états est $\{1,2\}$, sur l'alphabet $\{a,b\}$; les fonctions o et δ correspondantes sont données sur la droite.



x	$\delta(x,a)$	$\delta(x,b)$	$o(x)$
1	2	1	0
2	1	2	1

On étend la fonction de transition aux mots comme suit, par récurrence sur les mots :

$$\delta^* : X \times A^* \to X$$
$$\begin{cases} \delta^*(x,\epsilon) & \stackrel{\text{def}}{=} x \\ \delta^*(x,au) & \stackrel{\text{def}}{=} \delta^*(\delta(x,a),u) \end{cases}$$

(Dans ce sujet, on note $\stackrel{\text{def}}{=}$ l'égalité par définition.) Intuitivement, $\delta^*(x,u)$ est l'état atteint en lisant le mot u à partir de l'état x. On en déduit la fonction associant à chaque état x de l'automate le langage $L(x)$ reconnu par cet état.

$$L(x) \stackrel{\text{def}}{=} \{u \mid o(\delta^*(x,u)) = 1\}$$

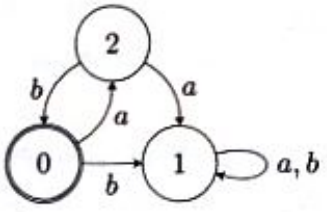
Notons que par définition, on a pour tout état x, toute lettre a, et tout mot u :

$$\epsilon \in L(x) \Leftrightarrow o(x) = 1$$
$$au \in L(x) \Leftrightarrow u \in L(\delta(x,a))$$

Pour les questions de cette partie (1.1 à 1.5), il n'est pas demandé de justifier les réponses.

Question 1.1 *Pour l'automate à deux états dessiné en haut de la page, donner une description intuitive des langages reconnus par les états* 1 *et* 2 *(c'est à dire,* $L(1)$ *et* $L(2)$*).*

Question 1.2 *Donner une description précise des langages reconnus par les trois états de l'automate suivant.*



2.3 Relations et fonctions de relations

On fixe dans cette partie un ensemble I. Une *relation* est une partie de $I \times I$. On s'intéresse aux fonctions des relations dans les relations. Autrement dit, on spécialise la partie précédente au cas $E = I \times I$. On note 1 la relation identité, $R \cdot S$ la composition de deux relations R, S, et $R^\intercal$ la transposée d'une relation R :

$$1 \stackrel{\text{def}}{=} \{\langle i, i\rangle \mid i \in I\}$$
$$R \cdot S \stackrel{\text{def}}{=} \{\langle i, k\rangle \mid \exists j \in I,\ \langle i, j\rangle \in R \wedge \langle j, k\rangle \in S\}$$
$$R^\intercal \stackrel{\text{def}}{=} \{\langle j, i\rangle \mid \langle i, j\rangle \in R\}$$

Etant donnée une relation R, on notera parfois $x\ R\ y$ pour $\langle x, y\rangle \in R$.

Question 2.12 *Comment qualifie-t-on usuellement une relation R telle que $1 \subseteq R$? telle que $R^\intercal \subseteq R$? telle que $R \cdot R \subseteq R$?*

Soient $r, s, t : \mathcal{P}(I \times I) \to \mathcal{P}(I \times I)$ les trois fonctions suivantes :

$$r : R \mapsto 1 \qquad s : R \mapsto R^\intercal \qquad t : R \mapsto R \cdot R$$

Question 2.13 *Qu'est-ce qu'un pré-point fixe pour r ? pour s ? pour t ? pour $r \cup t$? pour $r \cup s \cup t$?*

Question 2.14 *Montrer que les trois fonctions r, s et t sont continues.*

Au vu des questions 2.10 et 2.11, on appelle usuellement la fonction t^ω *clôture transitive* : c'est une clôture, et $t(R)$ est la plus petite relation transitive contenant R. De manière similaire, les fonctions $(r \cup t)^\omega$ et $(r \cup s \cup t)^\omega$ sont respectivement les fonctions de clôture réflexive-transitive et réflexive-symétrique-transitive.

3 Equivalence de deux états

On cherche désormais des algorithmes permettant de tester l'équivalence de deux états dans un automate fini donné. Nous allons tout d'abord montrer que l'algorithme proposé en partie 1 est correct, et de complexité quadratique. On fixe dans toute cette partie un automate $\langle X, o, \delta\rangle$, dont l'ensemble d'états X est fini, de taille n.

3.1 Un algorithme quadratique

Soit $p : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ la fonction suivante entre relations sur l'ensemble X des états :

$$p : R \mapsto \{\langle x, y\rangle \mid o(x) = o(y) \wedge \forall a \in A,\ \delta(x, a)\ R\ \delta(y, a)\}$$

Pour toutes relations R, S, en déroulant cette définition, on a

$$S \subseteq p(R) \quad \text{si et seulement si} \quad \forall x, y \in X,\ x\ S\ y \Rightarrow o(x) = o(y) \wedge \forall a \in A,\ \delta(x, a)\ R\ \delta(y, a)$$

On appelle *bisimulation* tout post-point fixe de p : toute relation R telle que $R \subseteq p(R)$.

Question 3.1 *Donner la plus petite bisimulation contenant la paire d'états $\langle 0,3\rangle$ de l'automate de la question 1.3. Montrer qu'il n'existe pas de bisimulation contenant la paire d'états $\langle 2,5\rangle$.*

Question 3.2 *Montrer que la fonction p est croissante.*

D'après la question 2.5, l'union de toutes les bisimulations est une bisimulation : c'est le plus grand point fixe νp de la fonction p.

Question 3.3 *(i) Montrer que la relation d'équivalence entre états ($\simeq$) est une bisimulation.*
(ii) Montrer que pour toute bisimulation R, $R \subseteq \simeq$.
(iii) En déduire que la plus grande bisimulation est l'équivalence de language : $\nu p = \simeq$.

Afin de prouver que deux états sont équivalents, il suffit donc de trouver une bisimulation contenant ces deux états. Inversement, pour prouver que deux états ne sont pas équivalents, il suffit de trouver un mot accepté par l'un mais pas l'autre. C'est exactement ce que fait l'algorithme de la partie 1.

Question 3.4 *Montrer que le premier invariant (I1) est satisfait à l'entrée de la boucle, et préservé à chaque itération.*

Question 3.5 *Faire de même pour le second invariant (I2).*

Question 3.6 *Que peut on en déduire à la sortie de la boucle (ligne 3) ?*

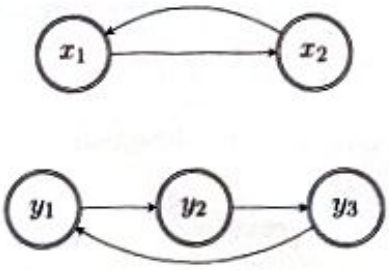
Question 3.7 *Donner un troisième invariant (I3) impliquant que les états de départ ne sont pas équivalents lorsque l'on sort de la boucle prématurément (ligne 2.3). Démontrer sa préservation. (Un tel invariant ne s'exprime pas de manière aussi concise que I2, mais reste relativement simple.)*

Question 3.8 *Donner une quantité décroissant strictement à chaque itération non-triviale (lorsque l'instruction* `continuer` *n'est pas exécutée). En déduire la terminaison de l'algorithme.*

Question 3.9 *En déduire que l'algorithme est correct et a une complexité dans le pire cas en $O(kn^2)$ où n est le nombre d'états de l'automate et k le nombre de lettres de l'alphabet. (On ne comptera que le nombre d'appels aux fonctions o et δ.)*

Question 3.10 *La variable F peut être implémentée par n'importe quelle structure de données de file (premier entrant premier sorti) ou de pile (premier entrant dernier sorti). Ces choix correspondent à deux stratégies d'exploration de l'automate ; lesquelles ? Dans quel cas ce choix a une importance sur le temps d'exécution de l'algorithme ? Donner un exemple.*

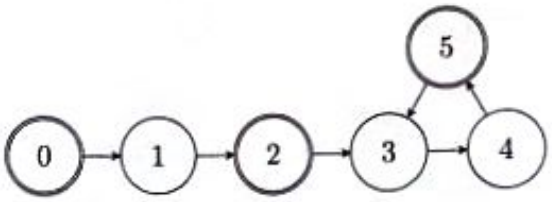
Considérons l'automate suivant, sur un alphabet réduit à une lettre de telle sorte que l'on peut omettre l'étiquetage des transitions, et dont tous les états sont acceptants.



Question 4.3 *Montrer que pour tout $i \in \mathbb{N}$, $p^i(X \times X)$ est une relation d'équivalence.*

Question 4.4 *Montrer que l'on a $\simeq = p^{n-1}(X \times X)$, où n est comme précédement le nombre d'états de l'automate.*

Question 4.5 *Calculer les termes de la suite $(p^i(X \times X))_{i \in \mathbb{N}}$ pour l'automate ci-dessous. (On rappelle qu'une relation d'équivalence peut être vue comme une partition en classes d'équivalence, on pourra donc représenter ainsi de telles relations.) En déduire un automate à quatre états reconnaissant les langages reconnus par les six états de l'automate de départ.*



Il s'agit maintenant de calculer cette suite efficacement. Soit P_0 la relation et d la fonction suivantes :

$$P_0 \stackrel{\text{def}}{=} \{\langle x, y\rangle \mid o(x) = o(y)\} \qquad d(R) \stackrel{\text{def}}{=} R \cap \{\langle x, y\rangle \mid \forall a \in A,\ \delta(x,a)\ R\ \delta(y,a)\}$$

Question 4.6 *Montrer que pour tout entier i, $d^i(P_0) = p^{i+1}(X \times X)$.*

Question 4.7 *Proposer un algorithme prenant en entrée un automate fini (X, o, δ) et renvoyant la relation $\simeq$, de complexité $O(kn^2 \log n)$ dans le pire cas. On détaillera les structures de données utilisées.*

Question 4.8 *En déduite un algorithme prenant en entrée un automate et renvoyant un automate de taille minimale dont les états reconnaissent les différents langages reconnus par ceux de l'automate de départ.*

On appelle un tel algorithme un algorithme de *minimisation*.

Question 4.9 *Raffiner la routine calculant d pour obtenir un algorithme de minimisation de complexité $O(kn^2)$ dans le pire cas.*

L'algorithme ainsi obtenu est dû à Edward Moore, il a ensuite été amélioré par John Hopcroft pour obtenir une complexité en $(O(kn \log n))$.

Fin du sujet.

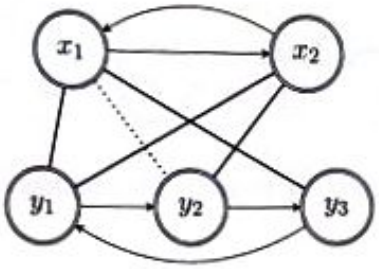
Question 3.20 *Combien d'itérations nécessite l'algorithme 1 pour prouver que x_1 et y_1 sont équivalents ?*

Question 3.21 *Proposer une optimisation permettant de terminer en temps constant sur cette famille d'entrées, et prouver sa correction via le cadre précédemment mis en place.*

Reconsidérons maintenant l'automate de la question 3.11. Au début de la cinquième itération, on a :

$$R = \{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_1, y_3\rangle, \langle x_2, y_1\rangle\} \qquad F = \{\langle x_1, y_2\rangle\}$$

ce qui peut être représenté ainsi :



Question 3.22 *Proposer une optimisation permettant de terminer lors de cette cinquième itération et prouver sa correction via le cadre précédemment mis en place.*

Question 3.23 *Toujours pour cette optimisation, donner une nouvelle quantité permettant de prouver que l'on effectue au plus n itérations non-triviales (n étant comme auparavant le nombre d'états de l'automate).*

Il reste toutefois à implémenter de façon efficace le test en ligne 2.2. Nous ne nous en chargeons pas ici ; cela peut-être fait en complexité amortie "presque constante". L'algorithme ainsi obtenu pour tester l'équivalence de deux états d'un automate est du à John Hopcroft et Richard Karp, sa complexité est "presque linéaire" : $O(kn\alpha(n))$ où k est le nombre de lettres et n le nombre d'états, et $\alpha(\cdot)$ est une fonction à croissance extrêmement faible (en fait, une inverse de la fonction d'Ackermann).

4 Relation globale d'équivalence

On s'intéresse maintenant au calcul, dans un automate $\langle X, o, \delta\rangle$ donné, de l'ensemble des paires d'états équivalents, c'est à dire de la relation $\simeq$ dans sa globalité.

Question 4.1 *En utilisant l'algorithme de John Hopcroft et Richard Karp, donner un algorithme simple pour calculer la relation $\simeq$, de complexité $O(kn^3\alpha(n))$.*

Cette approche n'est cependant pas optimale. Pour faire mieux, on va utiliser la caractérisation alternative du plus grand point fixe d'une fonction.

Question 4.2 *En utilisant les questions 2.8 et 3.3, et sans supposer que l'automate est fini, montrer que l'on a $\simeq = \bigcap_{i\in\mathbb{N}} p^i(X \times X)$.*

Nous allons proposer des conditions sur la fonction g permettant d'assurer la correction d'une telle variante, puis nous chercherons des fonctions g satisfaisant ces conditions.

L'invariant I2 précédemment utilisé n'est plus valide pour cet algorithme. De même, à la sortie de la boucle principale, R n'est plus nécessairement une bisimulation.

On appelle *bisimulation modulo g* tout post-point fixe de $p \circ g$.

Question 3.13 *Sous l'hypothèse d'une condition très simple sur la fonction g, donner un nouvel invariant I2' garantissant en sortie de boucle que R est une bisimulation modulo g. Démontrer que I2' est satisfait avant la boucle et préservé lors de chaque itération.*

Question 3.14 *Montrer que si g est extensive, alors la quantité donnée à la question 3.8 permet toujours de garantir la terminaison.*

Il nous faut maintenant trouver une condition garantissant que toute bisimulation modulo g est bien contenue dans l'équivalence de langage.

Nous nous plaçons pour cela à nouveau dans le cadre de la partie 2 : on fixe une fonction croissante $f : \mathcal{P}(E) \to \mathcal{P}(E)$, dont le plus grand point fixe nous intéresse.

Une fonction $h : \mathcal{P}(E) \to \mathcal{P}(E)$ est dite *compatible (avec f)* si h est croissante et $h \circ f \subseteq f \circ h$. L'identité est trivialement compatible.

Question 3.15 *Montrer que la composition de deux fonctions compatibles est compatible.*

Question 3.16 *Montrer que l'union d'une famille de fonctions compatibles est compatible.*

Question 3.17 *Montrer que si h est compatible, alors il en est de même de h^ω.*

Question 3.18 *Soit h une clôture compatible avec f.*

(i) Montrer que pour tout post-point fixe X de $f \circ h$, $h(X)$ est un post-point fixe de f.

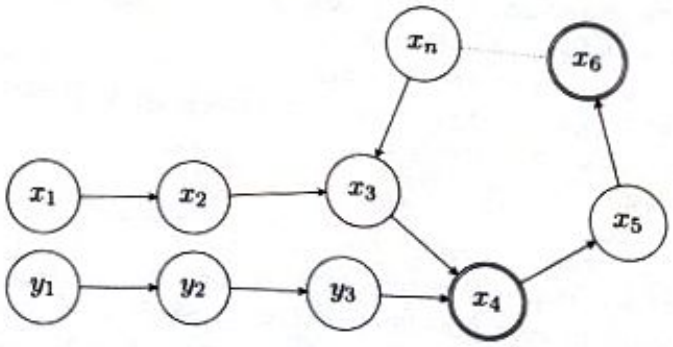
(ii) En déduire que $\nu(f \circ h) \subseteq \nu f$.

(iii) L'inclusion réciproque est-elle vérifiée ?

On revient maintenant dans le cadre des automates déterministés.

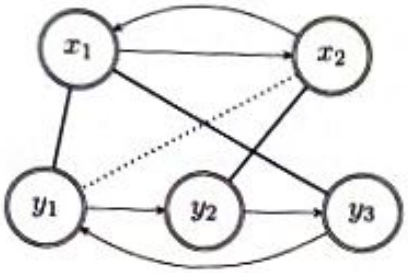
Question 3.19 *En utilisant les questions précédentes, donner une condition simple sur la fonction g garantissant la correction de l'algorithme optimisé.*

Nous allons maintenant exhiber des fonctions spécifiques pouvant être employées pour g.
Considérons un automate de la forme suivante :



L'alphabet n'a qu'une seule lettre, c'est pourquoi les transitions ne sont pas étiquetées. Il y a $n+3$ états dont seulement deux acceptants.

Si l'on commence l'exécution de l'algorithme précédent à partir de la paire $\langle x_1, y_1 \rangle$, on insère successivement les paires $\langle x_1, y_1 \rangle$, $\langle x_2, y_2 \rangle$, $\langle x_1, y_3 \rangle$, ... ce que l'on peut représenter comme ci-dessous, la paire représentée en pointillée étant la prochaine devant être traitée :



Question 3.11 *Représenter comme ci-dessus la relation R obtenue lorsque l'algorithme termine. Combien de paires contient cette relation ?*

Question 3.12 *En généralisant l'exemple précédent, donner une séquence d'automates et de paires d'états dans ces automates démontrant que la borne quadratique en le nombre d'états peut effectivement être atteinte, asymptotiquement. On se contentera de traiter le cas d'un alphabet à une seule lettre, et on considérera comme précédemment des automates dont tous les états sont acceptants.*

3.2 Un algorithme linéaire ?

Soit $g : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ une fonction. L'algorithme 2 est une variante de l'algorithme 1, où la seule modification apparaît en ligne 2.2 : au lieu de tester $\langle x, y \rangle \in R$, on teste $\langle x, y \rangle \in g(R)$. L'algorithme 1 s'obtient donc en prenant la fonction identité pour g.

Algorithme 2 Optimisation générique de l'algorithme 1, paramétrée par la fonction g utilisée à la ligne 2.2.

```
(1) R := ∅; F := {⟨x0, y0⟩};
(2) tant que F n'est pas vide,
        // I1: ⟨x0, y0⟩ ∈ R ∪ F
        // I2':
        // I3:
 (2.1)  extraire une paire ⟨x, y⟩ de F;
 (2.2)  si ⟨x, y⟩ ∈ g(R), continuer;
 (2.3)  si o(x) ≠ o(y), renvoyer faux ;
 (2.4)  pour tout a ∈ A, ajouter ⟨δ(x, a), δ(y, a)⟩ à F;
 (2.5)  ajouter ⟨x, y⟩ à R;
(3) renvoyer vrai;
```

Si l'on choisit une fonction extensive pour g, ce nouvel algorithme peut terminer plus rapidement : certaines paires ne sont plus insérées dans la file F. Cela peut bien entendu fausser le résultat : par exemple, si g est la fonction constante valant $X \times X$ partout, l'algorithme renvoie toujours vrai.