Devoir Non Surveillé 3

À rendre le 25 février 2025

INFORMATIQUE MP2I

Tournez la page S.V.P.

1 Un peu de géométrie (types énumérés en OCaml)

On s'intéresse à des ensembles de points du plan $\mathbb{R}^2$. On se donne les types suivants en OCaml pour les représenter :

```
type point = {x : float; y : float}
type vector = {x : float; y : float}
type set =
  | Set of point list (* Ensemble fini de points *)
  | Line of point * point (* Droite, donnee par deux de ses points *)
  | Circle of point * float (* Cercle, donne par son centre et son rayon *)
type cardinal =
  | Finite of int
  | Infinite
```

De plus, comme nous travaillons avec des nombres flottants, nous définissons une fonction qui nous permet de déterminer si deux nombres sont assez proches pour que l'on considère qu'ils soient égaux. On se fixe aussi une précision ε à cet effet.

```
let epsilon = 0.001

let is_eq (f1 : float) (f2 : float) =
  let df = f1 -. f2 in df > -. epsilon && df < epsilon
```

☐ 1 – Écrire une fonction len (l : 'a list) : int qui calcule la taille d'une liste, et en déduire une fonction card (s : set) : cardinal qui calcule le cardinal d'un ensemble de points.

Réponse 1.

```
let rec len (l : 'a list) : int = match l with
  | [] -> 0
  | _ :: t -> 1 + len t
```

☐ 2 – En supposant que le type int peut représenter tous les entiers sans dépassement de capacité, proposer une relation d'ordre totale $\leq_c$ telle que l'ensemble des éléments de type cardinal munit de $\leq_c$ est un ensemble bien fondé.

Réponse 2. On propose la relation suivante :
$a \leq_c b$ si et seulement si $b =$ `Infinite` ou ($a =$ `Finite`n et $b =$ `Finite`m avec $n \leq m$).
Il s'agit bien d'une relation d'ordre, totale, et bien fondée. (ce n'est pas difficile à montrer mais un peu long à rédiger, donc j'ai choisi de ne pas le faire car ce n'était pas demandé)

On souhaite à présent calculer l'intersection de deux ensembles de points du type set.

☐ 3 – Écrire une fonction vect (a : point) (b : point) : vector qui renvoie le vecteur $\overrightarrow{AB}$, et une fonction det (u : vector) (v : vector) : float qui calcule le déterminant de deux vecteurs $\vec{u}$ et $\vec{v}$, et en déduire une fonction are_aligned (a : point) (b : point) (c : point) : bool qui détermine si trois points A, B, et C sont alignés.

Réponse 3. On applique les formules pour les deux premières fonctions. On remarque que trois points A, B, et C sont alignés si les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AC}$ sont colinéaires, donc de déterminant nul.

```
let vect (a : point) (b : point) : vector =
  {x = b.x -. a.x; y = b.y -. a.y}

let det (u : vector) (v : vector) : float =
  u.x *. v.y -. u.y *. v.x
```

```
let are_aligned (a : point) (b : point) (c : point) : bool =
  let ab, ac = vect a b, vect a c in
  is_eq (det ab ac) 0.
```

☐ 4 – En déduire une fonction lines_intersect (a, b : point ∗ point) (c, d : point ∗ point) : set qui calcule l'intersection des droites (AB) et (CD). On prendra soin de justifier la correction de la fonction en détaillant les calculs qui y ont mené.

Réponse 4. Il faut distinguer trois cas :
- droites parallèles confondues ;
- droites parallèles distinctes ;
- droites sécantes.

Pour trouver le point d'intersection dans le second cas, on peut noter que les points M de la droite (AB) sont tels que $\vec{AM}$ et $\vec{AB}$ ont un déterminant nul. D'où l'équation de la droite (AB) :

$$(x_M - x_A)(y_B - y_A) - (y_M - y_A)(x_B - x_A) = 0 \tag{1}$$

On a une équation similaire pour la droite (CD) :

$$(x_M - x_C)(y_D - y_C) - (y_M - y_C)(x_D - x_C) = 0 \tag{2}$$

$(x_B - x_A)2 - (x_D - x_C)1$ donne une équation linéaire en x_M ne faisant pas intervenir y_M, le coefficient devant x_M étant le déterminant des vecteurs $\vec{AB}$ et $\vec{CD}$.
On obtient de même une équation sur y_M avec $(y_B - y_A)1 - (y_D - y_C)2$.
Après résolution du système on obtient les coordonnées suivantes :

$$d \cdot x_M = x_C(y_D - y_C)(x_B - x_A) - x_A(y_B - y_A)(x_D - x_C) + (x_B - x_A)(x_D - x_C)(y_A - y_C)$$

$$d \cdot y_M = y_A(x_B - x_A)(x_D - x_C) - y_A(x_B - x_A)(y_D - y_C) + (y_B - y_A)(y_D - y_C)(x_C - x_A)$$

avec $d = (x_B - x_A)(y_D - y_C) - (y_B - y_A)(x_D - x_C)$ le déterminant de $\vec{AB}$ et $\vec{CD}$.
Méthodes alternatives :

- On peut simplifier ces calculs en voyant le calcul comme la résolution d'un système d'équations linéaires et en utilisant la méthode de Cramer.
- On peut calculer les équations paramétriques des droites : M est sur AB s'il existe $\lambda \in \mathbb{R}$ tel que $\vec{AM} = \lambda\vec{AB}$.
- On peut calculer les équations de droites $y = ax + b$ et résoudre le système : attention au cas particulier des droites verticales.

```
let lines_intersect (a, b : point * point) (c, d : point * point) : set =
  let ab = vect a b in
  let cd = vect c d in
  let determinant = det ab cd in
  if is_eq determinant 0.
  then if are_aligned a b c (* droites paralleles *)
       then Line (a, b) (* droites confondues *)
       else Set [] (* droites paralleles non confondues *)
  else (* droites secantes *)
    let x = (c.x *. cd.y *. ab.x -. a.x *. ab.y *. cd.x +. ab.x *. cd.x *. (a.y -. c.y)) /
    let y = (a.y *. ab.x *. cd.y -. c.y *. cd.x *. ab.x +. ab.y *. cd.y *. (c.x -. a.x)) /
    Set [{x = x; y = y}]
```

☐ 5 – Écrire une fonction filter (p : 'a -> bool) (l : 'a list) : 'a list qui prend en entrée un prédicat p une liste ℓ et qui renvoie la liste des éléments x de ℓ vérifiant $p(x)$.

Réponse 5.

```
let rec filter (p : 'a -> bool) (l : 'a list) : 'a list = match l with
  | [] -> []
  | t :: q when p t -> t :: filter p q
  | _ :: q -> filter p q
```

☐ 6 – En déduire une fonction on_line (a, b : point ∗ point) (l : point list) : point list qui renvoie la liste des points de ℓ appartenant à la droite (AB),
et une fonction common_elements (l1 : 'a list) (l2 : 'a list) : 'a list qui renvoie la liste des éléments en qui sont à la fois dans ℓ_1 et dans ℓ_2. Quelle est la complexité de common_elements ?

Réponse 6.

```
let on_line (a, b : point * point) (l : point list) : point list =
  filter (fun x -> are_aligned a b x) l

let rec mem x = function
  | [] -> false
  | t :: q -> t = x || mem x q

let rec common_elements (l1 : 'a list) (l2 : 'a list) : 'a list =
  match l1 with
  | [] -> []
  | t :: q -> let common = common_elements q l2 in
              if mem t l2 then t :: common else common
```

☐ 7 – Écrire une fonction dist (a : point) (b : point) : float qui calcule la distance AB, et en déduire une fonction on_circle (o, r : point ∗ float) (l : point list) : point list qui renvoie la liste des points de ℓ appartenant au cercle de centre O de rayon r.

Réponse 7.

```
let dist (a : point) (b : point) : float =
  let dx, dy = a.x -. b.x, a.y -. b.y in
  sqrt (dx *. dx +. dy *. dy)

let on_circle (o, r : point * float) (l : point list) : point list =
  filter (fun x -> is_eq (dist o x) r) l
```

☐ 8 – Écrire des fonctions pour les cas restants,
et en déduire une fonction intersection (s1 : set) (s2 : set) : set qui calcule $S_1 \cap S_2$.[1]
On prendra soin de justifier la correction de ces fonctions en détaillant les calculs qui y ont mené.

Réponse 8. https://math.stackexchange.com/questions/256100/how-can-i-find-the-points-at-which-two-circl
Pour le cas d'une droite (AB) et d'un cercle, on peut se ramener à l'intersection de deux cercles en prenant le cercle symétrique par rapport à la droite (AB). Attention au cas où le centre du cercle est sur la droite.


```
let circles_intersect (o1, r1 : point * float) (o2, r2 : point * float) : set =
  if is_eq o1.x o2.x && is_eq o1.y o2.y (* Les centres sont confondus *)
  then if is_eq r1 r2
       then Circle (o1, r1) (* Les cercles sont egaux *)
       else Set [] (* Un cercle contient l'autre *)
  else
    let o12 = vect o1 o2 in
    let d = dist o1 o2 in
    assert (d > 0.);
    let l = (r1 *. r1 -. r2 *. r2 +. d *. d) /. 2. /. d in
    let h = sqrt (r1 *. r1 -. l *. l) in
    let x_l, x_r = l *. o12.x /. d, h *. o12.y /. d in
    let y_l, y_r = l *. o12.y /. d, h *. o12.x /. d in
    let p1 : point = {x = x_l +. x_r +. o1.x ; y = y_l -. y_r +. o1.y} in
    let p2 : point = {x = x_l -. y_r +. o1.x ; y = y_l +. y_r +. o1.y} in
    if is_eq p1.x p2.x && is_eq p1.y p2.y
    then Set [p1]
    else Set [p1; p2]

let symmetric (a, b : point * point) (o : point) : point =
  let ra, rb = dist a o, dist b o in
  match circles_intersect (a, ra) (b, rb) with
  | Set [o1; o2] -> if is_eq o.x o1.x && is_eq o.y o1.y then o2 else o1
  | Set [o1] -> o1
  | _ -> assert false

let line_circle_intersect (a, b : point * point) (o, r : point * float) : set =
  if are_aligned a b o
  then let oa = vect o a in
       let doa = dist o a in
       let p1 : point = {x = o.x +. oa.x *. r /. doa;
                         y = o.y +. oa.y *. r /. doa} in
       let p2 : point = {x = o.x -. oa.x *. r /. doa;
                         y = o.y -. oa.y *. r /. doa} in
       if is_eq p1.x p2.x && is_eq p1.y p2.y
       then Set [p1]
       else Set [p1; p2]
  else let o' = symmetric (a, b) o in
       circles_intersect (o, r) (o', r)

let intersection (s1 : set) (s2 : set) : set = match s1, s2 with
  | Set l1, Set l2
    -> Set (common_elements l1 l2)
  | Line l1, Line l2
    -> lines_intersect l1 l2
  | Line ll, Set ls | Set ls, Line ll
    -> Set (on_line ll ls)
  | Set ls, Circle c | Circle c, Set ls
```

1. Si vous n'arrivez pas à calculer l'intersection dans un ou plusieurs des cas restants, remplacez ce calcul par **assert false**

```
    -> Set (on_circle c ls)
  | Line l, Circle c | Circle c, Line l
    -> line_circle_intersect l c
  | Circle c1, Circle c2
    -> circles_intersect c1 c2
```

On souhaite à présent étendre notre type set pour inclure les unions de droites, ou les unions de cercles, ou les unions d'unions de droites et d'unions de cercles, etc.
On modifie donc à présent notre déclaration de type :

```
type set =
  | Set of point list (* Ensemble fini de points *)
  | Line of point * point (* Droite, donnee par deux de ses points *)
  | Circle of point * float (* Cercle, donne par son centre et son rayon *)
  | Union of (set * set) (* Union de deux ensembles de points *)
```

☐ 9 – Proposer un ajout à la fonction card pour gérer le nouveau cas.

Réponse 9.

```
let add_card c1 c2 = match c1, c2 with
  | Finite n1, Finite n2 -> Finite (n1 + n2)
  | _ -> Infinite

let rec card = function
  | Set l -> Finite (len l)
  | Union (s1, s2) -> add_card (card s1) (card s2)
  | _ -> Infinite
```

☐ 10 – Proposer un ajout à la fonction intersection pour gérer le nouveau cas. On pourra utiliser la distributivité de l'intersection par rapport à l'union.

Réponse 10.

```
let intersection (s1 : set) (s2 : set) : set = match s1, s2 with
  | Set l1, Set l2
    -> Set (common_elements l1 l2)
  | Line l1, Line l2
    -> lines_intersect l1 l2
  | Line ll, Set ls | Set ls, Line ll
    -> Set (on_line ll ls)
  | Set ls, Circle c | Circle c, Set ls
    -> Set (on_circle c ls)
  | Line l, Circle c | Circle c, Line l
    -> line_circle_intersect l c
  | Circle c1, Circle c2
    -> circles_intersect c1 c2
  | Union (s3, s4), s5 | s5, Union (s3, s4)
    -> Union (intersection s3 s5, intersection s4 s5)
```

2 Introduction aux fonctions de hachages

Dans cette partie, on pose $\Sigma = [\![0, 255]\!]$ l'alphabet composé des valeurs de type **char**. On note Σ^* l'ensemble des chaînes de caractères, c'est-à-dire des suites finies d'éléments de Σ, ainsi que ε la chaîne de caractère vide "". Pour $u \in \Sigma^*$ et $a \in \Sigma$, on note $u \cdot a$ le mot composé des lettres de u suivi de la lettre a. On cherche à construire une fonction $h : \Sigma^* \mapsto \Sigma$ et à étudier ses propriétés.

☐ 11 – Dans un premier temps, nous posons h définit comme suit :

$$h(\varepsilon) = 0, \qquad h(u \cdot a) = (128 \times (h(u) + a)) \mod 256$$

Donner les valeurs possibles pour $h(u)$, avec $u \in \Sigma^*$.

Réponse 11. $h(u)$ est un multiple de 128 et est inférieur à 256. Il ne peut prendre que les valeurs 0 et 128. Ces deux valeurs sont atteignables : $h(\varepsilon) = 0$ et $h(1) = 128$.

On considère à présent la fonction h définit comme suit :

$$h(\varepsilon) = 0, \qquad h(u \cdot a) = (17 \times (h(u) + a)) \mod 256$$

☐ 12 – Écrire une fonction **int** hash(**char** u[]) qui prend en entrée une chaîne de caractères u et qui renvoie $h(u)$.

Réponse 12. J'ai remplacé `char` par `unsigned char` pour éviter les valeurs négatives. Ce n'était pas attendu.

C

```c
int hash(unsigned char u[]){
  int h = 0;
  for (int i = 0; u[i] != '\0'; i = i + 1){
    h = (17 * (h + u[i])) % 256;
  }
  return h;
}
```

☐ 13 – Montrer que 17 est inversible dans $\mathbb{Z}/256\mathbb{Z}$ et donner son inverse.

Réponse 13. $17 \times 15 = 255$, donc $17 \times (-15) = -255 \equiv 1[256]$, l'inverse de 17 modulo 256 est $256 - 15 = 241$.

☐ 14 – Montrer que pour tout mot $u \in \Sigma^*$, si a est tiré aléatoire uniformément dans Σ, alors $h(u \cdot a)$ est distribué uniformément dans $[\![0, 255]\!]$.
Reformulation : montrer que pour toute valeur $x \in [\![0, 255]\!]$ et tout mot $u \in \Sigma^*$, il existe un unique a tel que $h(u \cdot a) = x$.

Réponse 14. Il suffit de prendre $a = 241x - h(u)) \mod 256$, ainsi, $h(u \cdot a) = (17 \times (h(u) + a)) \mod 256 = (17 \times (241x)) \mod 256 = x$.

☐ 15 – Donner deux chaînes de caractères u et v distinctes ($u \neq v$) telles que $h(u) = h(v)$.

Réponse 15. $u = \varepsilon$ et $v = 0$ conviennent.
On peut aussi citer $u = 1 \cdot 1$ et $v = 18$ si on veut éviter les 0.

☐ 16 – Soit $u_0, u_1, \ldots, u_n$ $n + 1$ lettres de Σ. Notons $u = \varepsilon \cdot u_0 \cdot u_1 \cdot \ldots \cdot u_{n-1}$ et $u' = \varepsilon \cdot u_1 \cdot u_2 \cdot \ldots \cdot u_n$. Exprimez $h(u')$ en fonction de $h(u)$, u_0 et u_n.

Réponse 16. Dans le cas général, on a $h(\varepsilon \cdot u_0 \cdot \ldots \cdot u_{n-1}) = \sum_{i=0}^{n-1} 17^{n-i} \times u_i \mod 256$. Par récurrence :
— $h(\varepsilon) = 0 = \sum_{i=0}^{-1} 17^{-i} u_i \mod 256$.

— Si $h(\varepsilon \cdot u_0 \cdot \ldots \cdot u_{n-1}) = \sum_{i=0}^{n-1} 17^{n-i} \times u_i \mod 256$, alors

$$\begin{aligned} h(\varepsilon \cdot u_0 \cdot \ldots \cdot u_{n-1} \cdot u_n) &= 17(u_n + \sum_{i=0}^{n-1} 17^{n-i} \times u_i) \mod 256 \\ &= 17 \sum_{i=0}^{n} 17^{n-i} \times u_i \mod 256 \\ &= \sum_{i=0}^{n} 17^{n+1-i} \times u_i \mod 256. \end{aligned}$$

On a donc :

$$\begin{aligned} h(\varepsilon \cdot u_1 \cdot \ldots \cdot u_{n-1} \cdot u_n) &= \sum_{i=0}^{n-1} 17^{n-i} u_{i+1} \mod 256 \\ &= \sum_{j=1}^{n} 17^{n-j+1} u_j \mod 256 \\ &= 17u_n - 17^{n+1}u_0 + \sum_{j=0}^{n-1} 17^{n-j+1} u_j \mod 256 \\ &= 17u_n - 17^{n+1}u_0 + 17h(\varepsilon \cdot u_0 \cdot \ldots \cdot u_{n-1}) \mod 256 \end{aligned}$$

☐ 17 – On note a^n le mot défini par :

$$a^0 = \varepsilon, \qquad a^{n+1} = a^n \cdot a$$

Déterminer en justifiant $\{h(a^n) \mid n \in \mathbb{Z}\}$ pour les valeurs de a suivantes :

1. $a = 0$ (remarque : impossible en C car le caractère 0 signe la fin de la chaîne de caractère)
2. $a = 1$
3. $a = 17$
4. a quelconque.

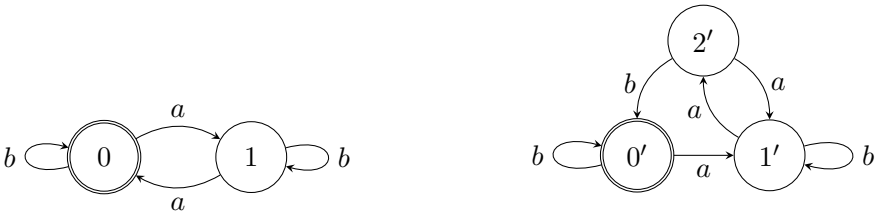
Réponse 17. En utilisant l'identité démontrée précédemment :
$h(a^n) = \sum_{i=0}^{n-1} 17^{n-i} \times a = a \sum_{i=0}^{n-1} 17^{n-i} = a \sum_{j=1}^{n} 17^j$. On peut remarquer que $\sum_{j=1}^{n} 17^j$ parcours toutes les valeurs entre 0 et 255 lorsque n varie de 0 à 255 (j'ai tout testé avec un petit programme).

1. $\{0\}$
2. $[\![0, 255]\!]$
3. $[\![0, 255]\!]$
4. $[\![0, 255]\!]$ si a est impair.
   Si $a = s2^k$, avec s impair, alors les multiples de 2^k qui sont dans $[\![0, 255]\!]$.

3 ENS - Info-Mathématiques 2022 - Section 2

La partie qui nous intéresse est la section 2. Ci-après sont inclues les pages correspondante. Le début de la première page et la fin de la dernière sont à ignorer.

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. $X_{i+1} \subseteq X_i$).

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.

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.*

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 :

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

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 :

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

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 $R \subseteq p(R)$.

4 ENS - Info-Mathématiques 2022 - Section 2 - Correction

Question 2.1 Soit f une fonction continue, $X, Y \subseteq E$ tels que $X \subseteq Y$.
On pose $X_0 = X$ et $\forall n \in \mathbb{N}, X_{n+1} = Y$.
$(X_n)_{n\in\mathbb{N}}$ est une suite croissante donc $f\left(\cup_{i\in\mathbb{N}} X_i\right) = \cup_{i\in\mathbb{N}} f\left(X_i\right)$ par continuité de f.
$f(X) \subseteq \cup_{i\in\mathbb{N}} f\left(X_i\right) = f\left(\cup_{i\in\mathbb{N}} X_i\right) = f(Y)$.
Donc f est croissante.

Question 2.2 Soit f une fonction co-continue, $X, Y \subseteq E$ tels que $X \subseteq Y$.
On pose $X_0 = Y$ et $\forall n \in \mathbb{N}, X_{n+1} = X$.
$(X_n)_{n\in\mathbb{N}}$ est une suite décroissante donc $f\left(\cap_{i\in\mathbb{N}} X_i\right) = \cap_{i\in\mathbb{N}} f\left(X_i\right)$ par co-continuité de f.
$f(X) = f\left(\cap_{i\in\mathbb{N}} X_i\right) = \cap_{i\in\mathbb{N}} f\left(X_i\right) \subseteq f(Y)$.
Donc f est croissante.

Question 2.3 Soit $E = \mathbb{N}$.
On pose $f : X \mapsto \emptyset$ si X est de cardinal fini et $X \mapsto \mathbb{N}$ si X est de cardinal fini.
f est croissante car si X est fini et $x \subseteq Y$, alors Y est fini.
La suite $X_i = \{j \in \mathbb{N} \mid j < i\}$ est croissante.
$f\left(\cup_{i\in\mathbb{N}} X_i\right) = f(\mathbb{N}) = \mathbb{N}$
$\cup_{i\in\mathbb{N}} f\left(X_i\right) = \cup_{i\in\mathbb{N}} \emptyset = \emptyset$
Donc f n'est pas continue.

La suite $Y_i = \{j \in \mathbb{N} \mid j >= i\} = \mathbb{N} \setminus X_i$ est décroissante.
$f\left(\cap_{i\in\mathbb{N}} X_i\right) = f(\emptyset) = \emptyset$
$\cap_{i\in\mathbb{N}} f\left(X_i\right) = \cap_{i\in\mathbb{N}} \mathbb{N} = \mathbb{N}$
Donc f n'est pas co-continue.

Question 2.4 (i) Soit f une fonction.
$f(E) \subseteq E$ donc E est pré-point fixe de f.
$\emptyset \subseteq f(\emptyset)$ donc $\emptyset$ est un post-point fixe de f.
(ii) On pose $f : E \mapsto \emptyset, X \neq E \mapsto E$. Cette fonction n'admet de point fixe que si $E = \emptyset$.

Question 2.5 Soit I une famille de post-points fixes.
Soit $x \in \cup_{X\in I} X$.
$\exists X \in I . x \in X \subseteq f(X)$.
Or $f(X) \subseteq f\left(\cup_{X\in I} X\right)$ par croissance de f.
Donc $x \in f\left(\cup_{X\in I} X\right)$.
$\cup_{X\in I} X$ est un post-point fixe.

Question 2.6 (i) On a $\nu f \subseteq f(\nu f)$, or f est croissante donc $f(\nu f) \subseteq f(f(\nu f))$.
$f(\nu f)$ est un post-point fixe, donc $f(\nu f) \subseteq \nu f$ par définition de νf.
νf est un pré-point fixe.
(ii) νf est un post-point fixe et un pré-point fixe donc c'est un point fixe.
Tout point fixe X est ausis post-point fixe, donc inclus dans νf par définition de νf.
νf est donc le plus grand point fixe au sens de l'inclusion.

Question 2.7 On reprend les questions précédentes avec $\mu f \stackrel{\text{def}}{=} \cap_{f(X)\subseteq X} X$, l'intersection de tous les pré-points fixes.
Pour tout pré-point fixe X, $f\left(\cap_{f(X)\subseteq X} X\right) \subseteq f(X)$ par croissance de f.
Soit $x \in f(\mu f) = f\left(\cap_{f(X)\subseteq X} X\right)$, alors pour tout pré-point fixe X, $x \in f(X) \subseteq X$, donc $x \in \cap_{f(X)\subseteq X} X = \mu f$.
$f(\mu f) \subseteq \mu f$ par croissance de f, donc $f(\mu f)$ est un pré-point fixe donc un point fixe.
Tout point fixe est un pré-point fixe donc contient μf, et μf est donc le plus petit point fixe.

Question 2.8 (i) Par récurrence :

- $f^1(E) = f(E) \subseteq E = f^0(E)$.
- Si $f^{i+1}(E) \subseteq f^i(E)$, alors par croissance de f, on a $f^{i+2}(E) \subseteq f^{i+1}(E)$.

Donc $\left(f^i(E)\right)_{i\in\mathbb{N}}$ est décroissante.
(ii) Si f est co-continue, on a $f(\cap_{i\in\mathbb{N}} f^i(E)) = \cap_{i\in\mathbb{N}} f^{i+1}(E)$.
$\cap_{i\in\mathbb{N}} f^{i+1}(E) \subseteq E$ donc $\cap_{i\in\mathbb{N}} f^{i+1}(E) = \cap_{i\in\mathbb{N}} f^i(E)$, et $\cap_{i\in\mathbb{N}} f^i(E)$ est un point-fixe.
Soit X un point fixe. On montre par récurrence que $\forall i \in \mathbb{N}, X \subseteq f^i(E)$:
- $X \subseteq E = f^0(E)$.
- Si $X \subseteq f^i(E)$, alors par croissance de f, $f(X) \subseteq f^{i+1}(E)$, or X est point fixe donc $f(X) = X \subseteq f^{i+1}(E)$.

Donc $X \subseteq \cap_{i\in\mathbb{N}} f^i(E)$, et $\cap_{i\in\mathbb{N}} f^i(E)$ est le plus grand point fixe, c'est-à-dire νf.

Question 2.9 (i) $(f \cup \mathrm{id})^0 = \mathrm{id}$, donc f^ω est extensive.
$(f \cup \mathrm{id})^1 = f \cup \mathrm{id}$ contient f et est contenu dans f^ω.
Donc f^ω contient f.
(ii) Si f est continue, alors elle est croissante. Il ne reste plus qu'à montrer que f est saturante pour qu'elle soit une clôture.
Soit $X \subseteq E$, $\forall i \in \mathbb{N}, (f \cup \mathrm{id})^i(X) \subseteq (f \cup \mathrm{id})^i(X) \cup f \circ (f \cup \mathrm{id})^i(X) \subseteq (f \cup \mathrm{id})^{i+1}(X)$.
Donc la suite $\left((f \cup \mathrm{id})^i(X)\right)_{i\in\mathbb{N}}$ est croissante.
$\forall j \in \mathbb{N}$, $(f \cup \mathrm{id})^j$ est continue comme composée de fonctions continues.
Donc $(f \cup \mathrm{id})^j(f^\omega(X)) = (f \cup \mathrm{id})^j(\cup_{i\in\mathbb{N}}(f \cup \mathrm{id})^i(X)) = \cup_{i\in\mathbb{N}}(f \cup \mathrm{id})^{i+j}(X) \subseteq f^\omega(X)$.
Donc $f^\omega \circ f^\omega(X) = \cup_{j\in\mathbb{N}}(f \cup \mathrm{id})^j(f^\omega(X)) \subseteq f^\omega(X)$. D'où f^ω est saturante.

Question 2.10 $\forall i \in \mathbb{N}, f \circ (f \cup \mathrm{id})^i(X) \subseteq (f \cup \mathrm{id})^{i+1}(X) \subseteq f^\omega(X)$, donc $f(f^\omega(X)) \subseteq f^\omega(X)$
De plus, $X \subseteq f^\omega(X)$, donc $f^\omega(X)$ est un pré-point fixe de f contenant X.
Soit f continue, et soit Y un pré-point fixe de f contenant X.
On montre par récurrence que $\forall i \in \mathbb{N}, (f \cup \mathrm{id})^i(X) \subseteq Y$:
- $(f \cup \mathrm{id})^0(X) = X \subseteq Y$ par définition de Y.
- Si $(f \cup \mathrm{id})^i(X) \subseteq Y$, alors par croissance de $f \cup \mathrm{id}$ (qui est continue donc croissante car composée de fonctions continues), $(f \cup \mathrm{id})^{i+1}(X) \subseteq (f \cup \mathrm{id})(Y) = Y$ car Y est un pré-point fixe de f.

On a donc $f^\omega \subseteq Y$, et f^ω est le plus petit pré-point fixe de f contenant X.

Question 2.11 Soit f continue. D'après la question 2.9, f^ω est une clôture contenant f.
Soit g une autre clôture contenant f. Soit $X \subseteq E$.
$g(X)$ contient X car g est extensive.
$f(g(x)) \subseteq g(g(x))$ car g contient f.
$g \circ g(X) \subseteq g(X)$ car g est saturante.
Donc $f(g(X)) \subseteq g(X)$, et $g(X)$ est un pré-point fixe de f contenant X.
Donc $f^\omega(X) \subseteq g(X)$ par la question 2.10, donc $f^\omega \subseteq g$.
f^ω est la plus petite clôture contenant f.

Question 2.12 Une relation R telle que $1 \subseteq R$ est appelée réflexive.
Une relation R telle que $R^T \subseteq R$ est appelée symétrique.
Une relation R telle que $R \cdot R \subseteq R$ est appelée transitive.

Question 2.13 Un pré-point fixe pour r est une relation réflexive, pour s symétrique, pour t transitive.
L'union des fonctions se traduit par la conjonction des propriétés.
Un pré-point fixe pour $r \cup t$ est donc une relation réflexive et transitive (un pré-ordre).
Un pré-point fixe pour $\cup t \cup s$ est une relation réflexive transitive et symétrique (une relation d'équivalence).

Question 2.14 Soit $(X_n)_{n\in\mathbb{N}}$ une suite croissante.
$\forall n \in \mathbb{N}$, on a $r(X_n) = 1$ donc $\cup_{n\in\mathbb{N}} r(X_n) = 1$, et $r(\cup_{n\in\mathbb{N}} X_n) = 1$.
Donc r est continue.
Soit $\langle j, i\rangle \in \cup_{n\in\mathbb{N}} s(X_n)$, alors $\exists n \in \mathbb{N}, \langle j, i\rangle \in s(X_n)$.
Donc $\langle i, j\rangle \in X_n \subseteq \cup_{n\in\mathbb{N}} X_n$, et $\langle j, i\rangle \in s(\cup_{n\in\mathbb{N}} X_n)$.
Soit $\langle j, i\rangle \in s(\cup_{n\in\mathbb{N}} X_n)$, alors $\langle i, j\rangle \in \cup_{n\in\mathbb{N}} X_n$.
Donc $\exists n \in \mathbb{N}, \langle i, j\rangle \in X_n$, et $\langle j, i\rangle \in s(X_n) \subseteq \cup_{n\in\mathbb{N}} s(X_n)$.
Donc s est continue.
Soit $\langle i, k\rangle \in \cup_{n\in\mathbb{N}} t(X_n)$, alors $\exists n \in \mathbb{N}, \langle i, k\rangle \in t(X_n)$.
Donc $\exists j \in I, \langle i, j\rangle \in X_n \wedge \langle j, k\rangle \in X_n$.

$\langle i,j\rangle \in \cup_{n\in\mathbb{N}} X_n$ et $\langle j,k\rangle \in \cup_{n\in\mathbb{N}} X_n$, donc $\langle i,j\rangle \in t\left(\cup_{n\in\mathbb{N}} X_n\right)$.
Soit $\langle i,j\rangle \in t\left(\cup_{n\in\mathbb{N}} X_n\right)$, alors $\exists j \in I, \langle i,j\rangle \in \cup_{n\in\mathbb{N}} X_n \wedge \langle j,k\rangle \in \cup_{n\in\mathbb{N}} X_n$.
Donc $\exists n_1, n_2 \in \mathbb{N}, \langle i,j\rangle \in X_{n_1} \wedge \langle j,k\rangle \in X_{n_2}$.
En posant $n = \max(n_1, n_2)$, on a $\langle i,j\rangle \in X_n \wedge \langle j,k\rangle \in X_n$ par croissance de la suite $(X_n)_{n\in\mathbb{N}}$.
Donc $\langle i,k\rangle \in t(X_n) \subseteq \cup_{n\in\mathbb{N}} t(X_n)$.
Donc t est continue.