Devoir maison n°2 - À rendre le 07 Mars 2022
Induction : théorie et pratique

Notions abordées

- structure inductive de termes arithmétiques, type OCaml associé
- lecture d'une chaîne de caractères décrivant une expression arithmétique
- évaluation des expressions arithmétiques par différents algorithmes
- fonctions agissant sur les parties d'un ensemble : croissance, continuité, points fixes
- calcul d'un ensemble défini par induction grâce à un plus petit point fixe

Ce projet est composé de deux parties. La première, constituée des exercices 1 à 4, mêle des questions à faire sur machine (🖳) et d'autres à rédiger (✍). Cette partie peut être réalisée seul, seule, en binôme ou en trinôme, cependant les réponses proposées doivent être comprises par tous les membres du groupe. Le code est à rendre sur cahier de prépa, les autres réponses doivent être rendues au format papier à la rentrée. Il est possible de faire le code en groupe, mais de rédiger les réponses séparément. La seconde partie, constituée des exercices 5 à 7, ne comporte que des questions à rédiger, à rendre au format papier à la rentrée. Elle doit être réalisée individuellement afin de vous entraîner à rédiger des preuves dans la continuité de celles du chapitre 7.

Comme précédemment, 🪜 indique une question dont le but est d'accompagner la compréhension, ☐ indique une question qu'il faut faire, qu'on ne peut pas passer, ∗ indique une question un peu difficile, que l'on peut passer si elle l'est trop.

Concernant les rendus papiers :

→ les réponses à la première et la deuxième partie sont à rendre sur des copies différentes

→ les réponses doivent apparaître dans l'ordre de l'énoncé, ce qui ne vous oblige pas à les traiter dans cet ordre, vous pouvez laisser des blancs, et insérer des feuilles

→ les copies doivent être numérotées, les exercices et les numéros de question indiqués

→ les questions notées 🪜 , qui sont là pour vous guider dans l'appropriation des notions, peuvent être omises de ce rendu

Concernant le code à rendre :

→ chaque fonction doit être munie une description, en commentaire sous la signature de la fonction

→ les hypothèses sur les arguments doivent être précisées, au-dessus de la description

→ chaque fonction doit être testée dans un jeu de test qui apparaît dans le code remis

→ chaque fois que cela peut être utile à la compréhension du code proposé, une description de la fonction auxiliaire utilisée sera proposée

Expressions arithmétiques

Exercice 1 Définition des termes arithmétiques

On considère l'ensemble de symboles $\mathbb{O} = \{$ `Add`, `Sub`, `Mul`, `Div` $\}$. Il est nommé $\mathbb{O}$ car ces symboles représentent des opérations, (resp. l'addition, la soustraction, la multiplication et la division). On souhaite définir l'ensemble $\mathbb{T}$ des termes arithmétiques, un terme arithmétique étant :

- ou bien le symbole `Int` encapsulant un entier ;
- ou bien un symbole `Bop` encapsulant un élément de $\mathbb{O}$ et deux éléments de $\mathbb{T}$.

Par exemple $\mathbb{T}$ contient `Int(3)`, mais aussi `Bop(Add, Bop(Sub, Int(3), Int(-1)), Int(2))`.

Question 1 ✍ ☐
Donner les règles permettant la définition de l'ensemble $\mathbb{T}$ par induction.

Question 2 ✍ ☐
Énoncer le principe d'induction sur l'ensemble $\mathbb{T}$ ainsi défini. *On attend le principe d'induction simple, l'analogue des récurrence simples sur les entiers, et non des récurrences fortes.*

Question 3 ✍ ☐
Définir par induction sur $\mathbb{T}$ une fonction $\mathsf{op} : \mathbb{T} \to \mathbb{N}$ donnant le nombre d'opérateurs se trouvant dans un terme arithmétique. Définir de même une fonction $\mathsf{int} : \mathbb{T} \to \mathbb{N}$, donnant le nombre d'entiers se trouvant dans un terme arithmétique.

Question 4 ✍ ☐
Démontrer que pour tout terme arithmétique $t \in \mathbb{T}$, $\mathsf{op}(t) = \mathsf{int}(t) - 1$.

Question 5 ✍ ☐
Définir par induction sur $\mathbb{T}$ une fonction $\mathsf{eval} : \mathbb{T} \to \mathbb{Z}$ calculant la valeur d'un terme arithmétique. Par exemple, eval(`Bop(Add, Bop(Sub, Int(3), Int(1)), Int(2))`) $= (3 - 1) + 2 = 4$.

Question 6 ✍ ☐
Définir par induction sur $\mathbb{T}$ une fonction $\mathsf{simpl} : \mathbb{T} \to \mathbb{T}$ faisant la simplification consistant à remplacer chaque opération sur deux constantes par la constante valant le résultat de cette opération.
Par exemple, simpl(`Bop(Add, Bop(Sub, Int(3), Int(1)), Bop(Add, Int(1), Int(5))`) vaut `Bop(Add, Int(2), Int(6))`.

Question 7 ✍ ☐
Démontrer que pour tout terme arithmétique $t \in \mathbb{T}$, $\mathsf{eval}(t) = \mathsf{eval}(\mathsf{simpl}(t))$.

Exercice 2 Prise en main du code

Dans le fichier `arith.ml` se trouvant sur cahier de prépa, vous trouverez du code pour démarrer cet exercice avec quelques outils, parmi lesquels :

→ des définitions de types :

- `op` : représentant l'ensemble $\mathbb{O}$
- `composant` : représentant l'ensemble $\mathbb{O} \cup \mathbb{N}$
- `arith` : représentant l'ensemble $\mathbb{T}$

→ des définitions de fonctions pour manipuler les opérations

→ des fonctions de conversion en chaînes de caractères qui pourront être utilisées pour déboguer. *On ne demande pas de comprendre le code de ces fonctions, mais de savoir les utiliser.*

Question 1  

Afin de "prendre en main" les types proposés, définir les objets de type `arith` correspondant aux expressions suivantes $3 + (5 * 6)$, $(1 + 2)/(1 - 2)$ et 42.

Question 2  

Afin de "prendre en main" tous les outils à votre disposition, proposer pour chacune des fonctions outil un jeu de tests. Si des erreurs sont déclenchées dans des cas limites, (par exemple quand on demande de convertir en entier une chaîne de caractères qui ne contient pas que des chiffres), noter à quoi correspondent ces erreurs au cas où elles seraient déclenchées par la suite.

Exercice 3 Lecture d'une chaîne de caractères

Dans cet exercice on s'intéresse au problème de transformer une chaîne de caractères en un terme de type `arith`. On appellera ce processus l'**analyse syntaxique** (ou **parsing** en anglais). On suppose que dans les chaînes de caractères que l'on traite, les opérations sont représentées par les caractères dédiés usuels en programmation : `+`, `-`, `*` et `/`. On dira parfois qu'un caractère est une opération ou un opérateur pour dire que ce caractère est l'un de ces quatre là, qu'il désigne une opération. De plus on suppose que les règles de priorité usuelles sur les opérations arithmétiques n'ont pas cours, et que les expressions sont parenthésées de manière à lever toute ambiguïté. On ne demande donc pas que votre programme lève les ambiguïtés en ajoutant des parenthèses, et son comportement sur une expression mal parenthésée n'est pas garanti. Par exemple, la chaîne de caractères `"1+2*3"` pourra être interprétée par votre programme comme $(1 + 2) \times 3$ ou comme $1 + (2 \times 3)$.

Pour procéder à l'analyse syntaxique de la sous-chaîne de`s` comprise entre les indices `i` et `j`, on procède comme suit.

- S'il existe $x \in [i..j]$ tel que `s[x]` est une opération ne se trouvant pas entre deux parenthèses, alors on fabrique le terme `Bop`(o, t_1, t_2) où o est l'opération que représente `s[x]`, t_1 est l'opérande gauche de cette opération, c'est-à-dire le terme représenté par la sous-chaîne `s[i..x-1]` et t_2 est l'opérande droit de cette opération, c'est-à-dire le terme représenté par la sous-chaîne `s[x+1..j]`
- Sinon, si `s[i]` est une parenthèse ouvrante, on attend nécessairement une parenthèse fermante à l'indice `j`, le terme représenté par la sous-chaîne `s[i..j]` est alors le même que celui représenté `s[i+1..j-1]`.
- Sinon, la sous-chaîne `s[i..j]` représente un entier n, auquel cas on fabrique le terme `Int` n.

Question 1 ✍ 🪜
Appliquer manuellement cet algorithme sur les chaînes `"3+(2*(5-1))"` et `"(25-(3+1))/(1+2)"`.

Question 2 💻 ☐
Définir une fonction de signature `trouve_op (s:string) (i:int) (j:int) : int option` qui calcule, s'il existe, l'indice dans `s` d'un opérateur qui n'est pas entre parenthèses dans la sous-chaîne `s[i..j]`, et qui donne la valeur None sinon.
Par exemple pour `s="3/((1+2)*10)"`, `i=3` et `j=10` on attend `Some(8)`, tandis que pour la même chaîne mais `i=3` et `j=7` on attend `None`.

Dans le cas où il y a plusieurs opérateurs hors de parenthèses on s'intéresse à l'indice du premier, sachant que si l'expression est bien parenthésée comme supposé, cela ne peut arriver que pour une opération associative, par exemple dans `"1+2+3+4"`, *et que dans ce cas l'ordre d'évaluation importe peu, de sorte qu'une fonction qui donne l'indice du dernier* `+` *et non pas du premier conduirait à un terme qui aura la même valeur à l'issue de l'évaluation.*

Question 3 💻 ☐
Définir une fonction de signature `parser (s:string) (i:int) (j:int) : arith` qui calcule le terme représenté par la sous-chaîne `s[i..j]` en suivant l'algorithme précédemment décrit.

Question 4 💻 ☐
En déduire une fonction de signature `read (s:string) : arith` qui calcule le terme arithmétique que représente la chaîne de caractères `s`.

Exercice 4 Évaluation(s) des termes arithmétiques

Dans cet exercice on s'intéresse à l'**évaluation** d'une expression arithmétique, c'est-à-dire à déterminer à quelle valeur entière mènent les opérations décrites dans cette expression. On suppose qu'on connaît la structure de cette expression[1], pas seulement son écriture, on travaille donc sur des objets de type `arith` et non des `string`. Par exemple, l'expression qu'on écrit usuellement $12+(16*3)$, c'est-à-dire `Bop(Add,Int 12,Bop(Mul,Int 16, Int 3))` est évaluée à 60.

On propose plusieurs algorithmes d'évaluation, d'abord au moyen de la pile d'appel de OCaml (questions 1 à 3), ensuite au moyen de piles explicites (questions 4 à 7), et enfin par des simplifications successives, (questions 8 à 10).

Question 1 💻 ☐
Définir une fonction de signature `opere (o:op) (a:int) (b:int) : int` qui réalise l'opération `o` sur les entiers `a` et `b`.

Question 2 💻 ☐
Définir une fonction récursive `eval` qui implante la fonction eval définie en question 5 de l'Exercice 1. *On ne cherche pas ici à avoir une fonction récursive terminale.*

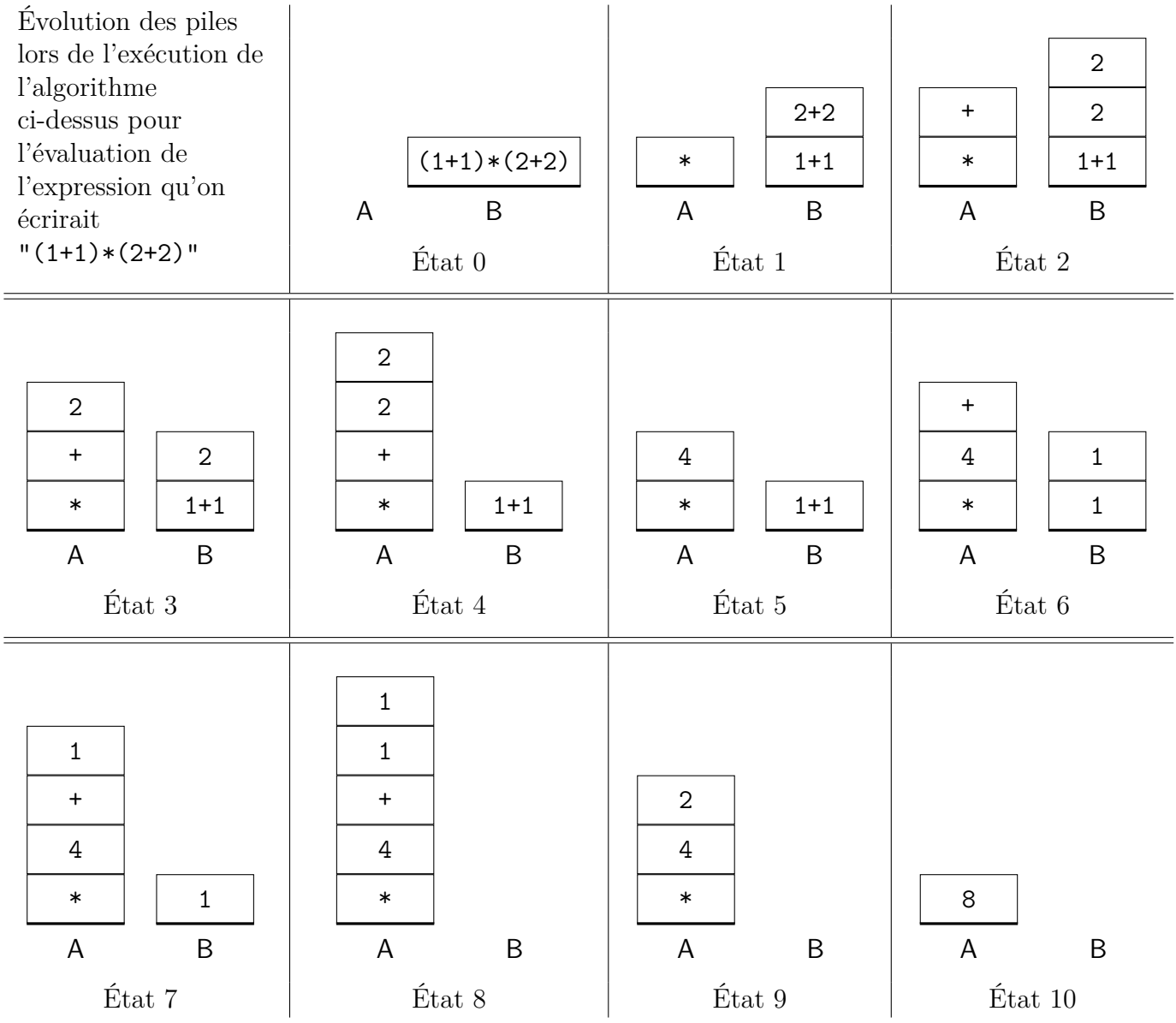
Question 3 ✍ ☐
Représenter par une suite d'égalités menant de l'expression initiale au résultat, la suite de calculs engendrée par l'appel `eval (read "(1+(2-1))*(2*25)")`. *Par souci de lisibilité, on représentera les expressions de manière usuelle, non comme des termes de* $\mathbb{T}$ *ni des éléments de type* `arith`.

[1] c'est comme si on connaissait déjà son arbre de syntaxe

On cherche maintenant à proposer une version récursive terminale de cette fonction d'évaluation. L'idée est de décomposer l'expression arithmétique en petits morceaux qu'on ne peut plus décomposer, qu'on appelle **composants**, qui sont des opérations ou des entiers. Dès qu'on le peut, on traite ces composants, c'est-à-dire qu'on applique une opération aux deux entiers correspondant à ses opérandes. Afin de gérer d'une part les sous-termes restant à traiter, (*i.e.* à décomposer puis évaluer), et d'autre part les composants déjà extraits de l'expression, on maintient deux piles, que l'on note respectivement A et B. Initialement, B contient un seul terme, l'expression entière, et A ne contient aucun composant. Puis on applique l'une des 4 règles suivantes, données par ordre de priorité, jusqu'à ce que la pile (A) ne contienne plus qu'un seul entier et que la pile (A) soit vide.

1. Si le haut de la pile (A) décrit une opération numérique faisable, c'est-à-dire s'il est constitué d'un entier, d'un entier et d'une opération, alors on dépile ces trois éléments, on applique l'opération sur les deux entiers, et on empile le résultat au sommet de (A).
   Passage des états 4 à 5, 8 à 9 et 9 à 10 dans l'exemple ci-dessous.
2. Si le haut de la pile (B) contient un terme qui est l'application d'une opération binaire à deux sous-termes, alors l'opération est empilée sur (A) et les sous-termes sont empilés sur (B).
   Passage des états 0 à 1, 1 à 2 et 5 à 6 dans l'exemple ci-dessous.
3. Si le sommet de (B) est un terme réduit à un entier, alors celui-ci est transvasé dans (A)
   Passage des états 2 à 3, 3 à 4, 6 à 7 et 7 à 8 dans l'exemple ci-dessous.

Évolution des piles lors de l'exécution de l'algorithme ci-dessus pour l'évaluation de l'expression qu'on écrirait `"(1+1)*(2+2)"`



Question 4 ✍ 🪜

Représenter l'évolution des piles lors de l'exécution de l'algorithme décrit ci-avant pour l'évaluation de l'expression qu'on écrirait `"((3*5)+2)-10"`. *On pourra au préalable représenter l'arbre de syntaxe de cette expression, afin de bien voir sa structure.*

Question 5 🖳 ☐

Implanter cet algorithme dans une fonction de signature `eval_tr : arith -> int`. Dans cette fonctions les piles seront de simples listes OCaml, il n'est pas nécessaire de définir un type `pile` pour l'occasion.

Question 6 ✍ ☐

Dans l'algorithme décrit ci-avant, certaines transformations opèrent sur (B) et font grandir (A), tandis que d'autres opèrent sur (A) et laissent (B) inchangée, ce qui rend la terminaison de l'algorithme non évidente. C'est pourquoi on cherche un **variant**, c'est-à-dire dans ce cadre une expression sur (A) et (B) à valeurs dans un espace ordonné bien fondé dont la valeur décroît strictement à chaque étape de l'algorithme. Donner un tel variant et le prouver.

On pourra chercher une fonction f_A (resp. f_B) de l'espace des piles de composants (resp. l'espace des piles de termes arithmétiques) dans un espace ordonné $(\mathbb{A}, \preceq_{\mathbb{A}})$ (resp. $(\mathbb{B}, \preceq_{\mathbb{B}})$) puis considérer l'ordre lexicographique sur $\mathbb{A} \times \mathbb{B}$ dérivé à partir de $\preceq_{\mathbb{A}}$ et $\preceq_{\mathbb{B}}$.

Question 7 ✍ ☐

Démontrer que l'algorithme précédent admet l'invariant suivant : la taille de la pile (A) est inférieure ou égale à $2o + 1$ où o est le nombre d'éléments de la pile (A) qui sont des opérations. Donner, pour chaque $n \in \mathbb{N}$, un terme arithmétique contenant n opérations sur lequel l'exécution de l'algorithme précédent produit à une étape une pile (A) de taille $2n + 1$.

Question 8 🖳 ☐

Définir une fonction de signature `pt_fixe (f: 'a -> 'a) (x0: 'a) : 'a` qui calcule le premier élément x de la suite $(\texttt{f}^{(n)}(\texttt{x0}))_{n\in\mathbb{N}}$ tel que $x = \texttt{f}(x)$, autrement dit le premier point fixe de `f` que l'on rencontre en itérant `f` à partir de `x0`. *Si un tel élément x n'existe pas, la fonction pourra ne pas terminer.*

Question 9 🖳 ☐

Définir une fonction récursive `simpl` qui implante la fonction mathématique eval définie en question 6 de l'Exercice 1.

Question 10 🖳 ☐

Déduire des deux questions précédentes une fonction de signature `eval_fp : arith -> int` permettant l'évaluation d'un terme arithmétique.

Ordres et point fixes

Pour toute cette partie, on fixe un ensemble S et $\mathcal{E} = \wp(S)$ l'ensemble de ses parties, ordonné par la relation d'inclusion. On s'efforcera de typographier en minuscule (x, y, z) les éléments de S, en majuscules d'imprimerie (X, Y, Z) les éléments de $\mathcal{E}$, en majuscules calligraphiques ($\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$) les ensembles d'éléments de $\mathcal{E}$. Pour une fonction $\varphi \in \mathcal{F}(\mathcal{E}, \mathcal{E})$, on note $\text{FP}(\varphi)$ l'**ensemble des points fixes** de φ, *i.e.* $\{X \in \mathcal{E} \mid \varphi(X) = X\}$. On s'intéresse dans cette partie à ces points fixes et en particulier à leur plus petit élément, sous réserve d'existence, qui permet ensuite de calculer des ensembles définis par induction.

Exercice 5 Questions préliminaires

Question 1 ✍ ☐
Montrer que pour tout $\mathcal{X} \subseteq \mathcal{E}$, la borne supérieure de $\mathcal{X}$ est $\bigcup_{X \in \mathcal{X}} X$, autrement dit montrer que $\sup(\mathcal{X}) = \{x \in S \mid \exists X \in \mathcal{X},\, x \in X\}$.

On admet de même que $\bigcap_{X \in \mathcal{X}} X$ est la borne inférieure de tout sous-ensemble $\mathcal{X}$ de $\mathcal{E}$. Ainsi chaque sous-ensemble $\mathcal{X}$ de $\mathcal{E}$ admet une borne supérieure et une borne inférieure.

On dit qu'une fonction $\varphi \in \mathcal{F}(\mathcal{E}, \mathcal{E})$ est **continue** si pour tout sous-ensemble $\mathcal{X}$ non vide de $\mathcal{E}$, $\varphi(\bigcup_{X \in \mathcal{X}} X) = \bigcup_{X \in \mathcal{X}} \varphi(X)$. Cela revient à dire que l'image de la borne supérieure de $\mathcal{X}$ est la borne supérieure des images par φ des éléments de $\mathcal{X}$.

Question 2 ✍ ☐
Montrer que pour $S = \{0, 1, 2\}$ et $\varphi = \begin{pmatrix} \wp(\{0,1,2\}) & \to & \wp(\{0,1,2\}) \\ X & \mapsto & X \cup \{1\} \end{pmatrix}$, la fonction φ est continue.

Question 3 🖳 ☐
Montrer que pour $S = \{0, 1, 2\}$ et $\varphi = \begin{pmatrix} \wp(\{0,1,2\}) & \to & \wp(\{0,1,2\}) \\ \{0,1\} & \mapsto & \{0,1,2\} \\ X \neq \{0,1\} & \mapsto & X \end{pmatrix}$, φ n'est pas continue.

Question 4 🖳 ☐
Montrer qu'une fonction continue $\varphi \in \mathcal{F}(\mathcal{E}, \mathcal{E})$ est aussi croissante, *i.e.* $\forall (X, Y) \in \mathcal{E}^2, X \subseteq Y \Rightarrow \varphi(X) \subseteq \varphi(Y)$. *On pourra s'intéresser à la borne supérieure d'une paire* $\{X, Y\}$, *avec* $(X, Y) \in \mathcal{E}^2$.

Exercice 6 Existence du plus petit point fixe

Dans cet exercice, on fixe $\varphi \in \mathcal{F}(\mathcal{E}, \mathcal{E})$ une fonction continue de $\mathcal{E}$ dans $\mathcal{E}$. Le but est de montrer que $\text{FP}(\varphi)$, l'ensemble de ces points fixes, admet pour plus petit élément $A = \bigcup_{n \geq 0} \varphi^{(n)}(\emptyset)$, où $\varphi^{(n)}$ désigne la composée de φ avec elle-même n fois ($\varphi^{(0)} = id_{\mathcal{E}}, \varphi^{(1)} = \varphi, \varphi^{(2)} = \varphi \circ \varphi ...$).

Question 1 🖳 ☐
Montrer que pour toute fonction $\varphi \in \mathcal{F}(\mathcal{E}, \mathcal{E})$, la suite $(\varphi^{(n)}(\emptyset))_{n \in \mathbb{N}}$ est croissante dans $\mathcal{E}$.

Question 2
Montrer que A est un point fixe de φ, *i.e.* que $\varphi(A) = A$.

Question 3
Montrer que A est un minorant de $\text{FP}(\varphi)$, l'ensemble des points fixes de φ.

Question 4
Conclure

Exercice 7 Application : définitions par induction

On fixe dans cet exercice B un sous-ensemble de S et $(f_i)_{i\in[1..n]}$ une famille de fonctions de $\mathcal{F}(S,S)$.
On définit la fonction $\varphi = \begin{pmatrix} \wp(S) & \to & \wp(S) \\ Y & \mapsto & B \cup Y \cup \{f_i(y) \mid y \in Y, i \in [1..n]\} \end{pmatrix}$.

Question 1
Soit $X \subseteq S$. On dit que X est **stable** par les f_i pour $i \in [1..n]$, ssi $\forall x \in X$, $\forall i \in [1..n]$, $f_i(x) \in X$.
Montrer que si $B \subseteq X$, alors X est stable par les $(f_i)_{i\in[1..n]}$ si et seulement si $X \in \text{FP}(\varphi)$.

Question 2
Montrer que φ est une fonction continue de $\wp(S)$ dans $\wp(S)$.

Question 3
En déduire que $\bigcup_{n\in\mathbb{N}} \varphi^{(n)}(\emptyset)$ est l'ensemble défini par induction à partir de B et des $(f_i)_{i\in[1..n]}$, c'est-à-dire le plus petit sous-ensemble de S contenant B et stable par les $(f_i)_{i\in[1..n]}$.

Question 4
Montrer que si S est un ensemble fini, alors il existe $n_0 \in \mathbb{N}$ tel que $\varphi^{(n_0)}(\emptyset) \in \text{FP}(\varphi)$.

Question 5
En déduire un algorithme calculant l'ensemble défini par induction à partir de B et des $(f_i)_{i\in[1..n]}$ dans le cas où S est fini.
On attend ici une description en français ou en pseudo-code de l'algorithme.

Question 6
Pour $S = [0..10]$, $B = \{1\}$, $f_1 = x \mapsto x + 3$ et $f_2 = x \mapsto 2x$, donner les itérés de la fonction φ sur $\emptyset$.