

TD8: Induction structurelle

MP2I Lycée Pierre de Fermat

Exercice 1.

Entiers de Péano

On définit inductivement l'ensemble $\mathcal{P}$ des **entiers de Péano** :

- $\mathbf{Z} \in \mathcal{P}$: c'est l'élément appelé “zéro”.
- Si $n \in \mathcal{P}$, alors $\mathbf{S}(n) \in \mathcal{P}$, c'est l'élément appelé “successeur de n ”.

Dans ce système, le nombre que l'on appelle habituellement “deux” serait donc $\mathbf{S}(\mathbf{S}(\mathbf{Z}))$.

Q1. Définir une fonction inductive $\mathbf{int} : \mathcal{P} \rightarrow \mathbb{N}$ mettant en bijection $\mathcal{P}$ et $\mathbb{N}$.

Dans la suite, interdiction d'utiliser la fonction $\mathbf{int}$.

On définit la fonction inductive $\mathbf{add} : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$:

- $\forall b \in \mathcal{P}, \mathbf{add}(\mathbf{Z}, b) = b$
- $\forall a \in \mathcal{P}, \forall b \in \mathcal{P}, \mathbf{add}(\mathbf{S}(a), b) = \mathbf{S}(\mathbf{add}(a, b))$

Q2. Montrer par induction structurelle sur $a \in \mathcal{P}$ que $\forall a, b \in \mathcal{P}, \mathbf{int}(\mathbf{add}(a, b)) = \mathbf{int}(a) + \mathbf{int}(b)$.

Nous allons maintenant montrer que la fonction $\mathbf{add}$ est commutative et associative.

Q3. Montrer la propriété suivante par induction structurelle sur $a \in \mathcal{P}$:

$$\forall a \in \mathcal{P}, \mathbf{add}(a, \mathbf{Z}) = a$$

Q4. Montrer la propriété suivante par induction structurelle sur $a \in \mathcal{P}$:

$$\forall a \in \mathcal{P}, \forall b \in \mathcal{P}, \mathbf{add}(\mathbf{S}(a), b) = \mathbf{add}(a, \mathbf{S}(b))$$

Q5. Montrer que $\forall a, b \in \mathcal{P}, \mathbf{add}(a, b) = \mathbf{add}(b, a)$, i.e. que l'addition est commutative.

Q6. Exprimer et montrer l'associativité de l'addition.

Q7. Définir inductivement la multiplication sur les entiers de Péano, et montrer qu'elle est commutative et associative.

Exercice 2.

Entiers de Péano (partie 2)

On reprend les mêmes notations que l'exercice précédent, et on s'autorise à en utiliser tous les résultats : associativité et commutativité de l'addition et de la multiplication.

Q1. Énoncer et montrer la distributivité de la multiplication sur l'addition.

On définit la relation $\leq$ sur les entiers de Péano comme suit :

$$\forall n, m \in \mathcal{P}, n \leq m \Leftrightarrow \exists k \in \mathcal{P}, m = \mathbf{add}(n, k)$$

Q2. Montrer que cette relation est réflexive, transitive et antisymétrique.

Exercice 3.

On introduit une notation standard pour la définition inductive d'ensembles. On souhaite définir l'ensemble $\mathcal{E}$ des expressions arithmétiques, avec les variables, les constantes entières, l'addition et la multiplication. On pose $\mathcal{X}$ un ensemble infini de noms de variables. On voudrait donc prendre la liste de constructeurs suivante :

$$\text{--- } (\mathbf{Int}, 0, \mathbb{N}) \quad \text{--- } (\mathbf{Var}, 0, \mathcal{X}) \quad \text{--- } (\mathbf{Plus}, 2, \{\bullet\}) \quad \text{--- } (\mathbf{Fois}, 2, \{\bullet\})$$

Pour définir cet ensemble de manière plus condensée, on écrira :

$$e ::= n \mid x \mid e_1 + e_2 \mid e_1 \times e_2$$

où e, e_1, e_2 désignent des expressions arithmétiques, n un entier et $x \in \mathcal{X}$. On note $\mathcal{E}$ l'ensemble des expressions arithmétiques, c'est à dire l'ensemble obtenu par induction à partir des règles précédentes. Dans cet exercice, on utilisera tout de même la notation $\mathbf{Int}(n)$ pour ne pas confondre avec $n \in \mathbb{N}$.

Avec cette notation, on peut écrire des expressions comme $(\mathbf{Int}(5) + x) \times (y + \mathbf{Int}(8))$. Il faut cependant bien garder en tête que les symboles $+$ et $\times$ ont un sens **syntactique** et pas **sémantique**. Autrement dit, rien ne dit pour l'instant que $+$ correspond à la "vraie somme" : on a simplement écrit $e_1 + e_2$ là où on aurait écrit $\mathbf{Plus}(e_1, e_2)$ avec les notations habituelles du cours.

Q1. Définir inductivement n_{op} la fonction qui à une expression associe son nombre d'opérateurs binaires

Q2. Définir de même la fonction n_{atom} la fonction qui à une expression associe son nombre de constantes et de variables.

Q3. Montrer que pour $e \in \mathcal{E}$, $n_{\text{op}}(e) + 1 = n_{\text{atom}}(e)$.

On appelle **contexte** toute fonction $\sigma : \mathcal{X} \rightarrow \mathbb{N}$. Un contexte permet donc d'associer une valeur à chaque variable. On note $\mathbb{N}^{\mathcal{X}} = \mathcal{F}(\mathcal{X}, \mathbb{N})$ l'ensemble des contextes.

Q4. Définir inductivement une fonction $\mathbf{eval}(e, \sigma)$ qui, étant donné une expression e et un contexte σ , calcule la valeur de e avec le contexte σ .

On définit inductivement la fonction $\mathbf{replace} : \mathcal{E} \times \mathbb{N}^{\mathcal{X}} \rightarrow \mathcal{E}$ suivante :

$$\begin{aligned} \text{--- } \forall x \in \mathcal{X}, \mathbf{replace}(x, \sigma) &= \mathbf{Int}(\sigma(x)) \\ \text{--- } \forall n \in \mathbb{N}, \mathbf{replace}(\mathbf{Int}(n), \sigma) &= \mathbf{Int}(n) \\ \text{--- } \forall e_1, e_2 \in \mathcal{E}, \mathbf{replace}(e_1 + e_2, \sigma) &= \mathbf{replace}(e_1, \sigma) + \mathbf{replace}(e_2, \sigma) \\ \text{--- } \forall e_1, e_2 \in \mathcal{E}, \mathbf{replace}(e_1 \times e_2, \sigma) &= \mathbf{replace}(e_1, \sigma) \times \mathbf{replace}(e_2, \sigma) \end{aligned}$$

Q5. Montrer que pour toute expression $e \in \mathcal{E}$, pour tout contexte σ_0 , on a :

$$\forall \sigma : \mathcal{X} \rightarrow \mathbb{N}, \mathbf{eval}(e, \sigma_0) = \mathbf{eval}(\mathbf{replace}(e, \sigma_0), \sigma)$$

Autrement dit, si l'on remplace les variables d'une expression par leurs valeurs dans σ_0 , puis qu'on évalue la nouvelle expression dans n'importe quel contexte, on obtient la même chose qu'on évalue l'expression dans le contexte σ_0 .

Exercice 4.

Réversibilité terminale

On considère la fonction suivante :

```
1 let rec rev (l: 'a list) : 'a =  
2   match l with  
3   | [] -> []  
4   | x :: q -> rev q @ [x]
```

On admet dans la suite que $\text{rev}(\text{rev } l) = l$ pour toute $l: 'a \text{ list}$. On admet également que l'opérateur de concaténation $@$ est associatif.

Q1. Donner la complexité de rev en fonction de n la taille de la liste d'entrée.

Q2. Proposer une version récursive terminale, plus efficace, et montrer formellement que les deux versions sont équivalentes. (*Indication : on pourra commencer par prouver une propriété sur la fonction auxiliaire.*)

On considère la fonction de fusion du tri-fusion :

```
1 let rec fusion (l1: 'a list) (l2: 'a list) : 'a list =  
2   match (l1, l2) with  
3   | [], [] -> []  
4   | [], _ -> l2  
5   | _, [] -> l1  
6   | x1 :: q1, x2 :: q2 ->  
7     if x1 < x2 then  
8       x1 :: fusion q1 l2  
9     else  
10      x2 :: fusion l1 q2
```

ainsi que la version récursive terminale suivante :

```
1 let rec rev_fusion_concat l1 l2 l3 =  
2   match (l1, l2) with  
3   | [], l | l, [] -> rev l @ l3  
4   | x1 :: q1, x2 :: q2 ->  
5     if x1 < x2 then  
6       rev_fusion_concat q1 l2 (x1::l3)  
7     else  
8       rev_fusion_concat l1 q2 (x2::l3)  
9  
10  let fusion2 l = rev (rev_fusion_concat l l [])
```

Q3. On note **rfc** la fonction rev_fusion_concat . Montrer par induction sur les triplets de listes, munis de l'ordre lexicographique, que pour toutes listes L_1, L_2, L_3 , on a :

$$\text{rfc}(L_1, L_2, L_3) = \text{rev}(\text{fusion}(L_1, L_2)) @ L_3$$

Q4. En déduire que **fusion** et **fusion2** sont équivalentes.

Exercice 5.

Preuves sur les listes

On considère les deux fonctions suivantes calculant la longueur d'une liste. La première est récursive non-terminale, et la deuxième est récursive terminale :

```
1 let rec len (l: 'a list) : int =
2   match l with
3   | [] -> 0
4   | x :: q -> 1 + len q
5
6 let rec len2 (l: 'a list) (r: int) : int =
7   match l with
8   | [] -> r
9   | x :: q -> len2 q (r + 1)
```

Q1. Notons L l'ensemble des listes. Montrer par induction sur la structure des listes que $\forall l \in L, \forall r \in \mathbb{N}, \text{len2 } l \ r = \text{len } l + r$

On dit qu'une fonction $f: 'a \rightarrow 'a \rightarrow 'a$ est commutative si pour tout x, y , $f \ x \ y = f \ y \ x$. On dit que f est associative si pour tout x, y, z , $f \ x \ (f \ y \ z) = f \ (f \ x \ y) \ z$.

Q2. Écrire une fonction `append` permettant de concaténer deux listes. Montrer que `append` est associative par induction structurelle.

On définit les fonctions `fold_right` et `fold_left` comme suit :

```
1 let rec fold_right (f: 'a -> 'b -> 'b) (l: 'a list) (d: 'b) : 'b =
2   match l with
3   | [] -> d
4   | x :: q -> f x (fold_right f q d)
5
6 let rec fold_left (f: 'a -> 'b -> 'a) (d: 'a) (l: 'b list) : 'a =
7   match l with
8   | [] -> d
9   | x :: q -> (fold_left f (f d x) q)
```

Q3. Déterminer si les expressions suivantes sont bien typées, et donner leurs valeurs lorsque c'est le cas :

1. `List.fold_right (+) [1;2;3] 0`
2. `List.fold_left (+) [1;2;3] 0`
3. `List.fold_right List.cons [1;2;3] []` (indication : `List.cons : 'a -> 'a list -> 'a list` est identique à l'opérateur `::`)
4. `List.fold_left List.cons [] [1;2;3]`

De manière générale, `fold_right` et `fold_left` n'ont pas le même effet car elles agissent dans des sens opposés sur leur liste d'entrée. Cependant, on a le résultat suivant :

Proposition 1. Soit $f: 'a \rightarrow 'a \rightarrow 'a$ une fonction associative et commutative. Alors pour toute liste l et pour tout élément d , $\text{fold_right } f \ l \ d = \text{fold_left } f \ d \ l$.

Montrons ce résultat. On pose $f: 'a \rightarrow 'a \rightarrow 'a$ une fonction associative et commutative.

Q4. Montrer que pour tout x, y, z , $f \ x \ (f \ y \ z) = f \ y \ (f \ x \ z)$.

Q5. Montrer que pour tout l, d, x , on a $\text{fold_right } f \ l \ (f \ d \ x) = f \ x \ (\text{fold_right } f \ l \ d)$.

Q6. Montrer que pour toute liste l et pour tout élément d , $\text{fold_right } f \ l \ d = \text{fold_left } f \ d \ l$.

Exercice 6.

Induction bien fondée

On considère la fonction suivante :

```
1 (* x et y doivent être positifs ou nuls *)
2 let rec f (x: int) (y: int) : int =
3   if (x, y) = (0, 0) then 0
4   else if y = 0 then f (x-1) (x+1)
5   else f x (y-1) + 1
```

Montrer que pour tout $x, y \in \mathbb{N}$, $f(x, y) = \frac{x(x+3)}{2} + y$.

Exercice 7.

Mi casa es tu casa

On considère la fonction d'Ackermann, définie sur $\mathbb{N}^2$ par :

$$A(m, n) = \begin{cases} n + 1 & \text{si } m = 0 \\ A(m - 1, 1) & \text{si } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{sinon} \end{cases}$$

Q1. Montrer que cette fonction termine sur toutes les entrées $(m, n) \in \mathbb{N}^2$.

Q2. Montrer que pour $m, n \in \mathbb{N}$, $A(m, n) \geq m + n + 1$

Q3. Montrer que pour $n \in \mathbb{N}$, $A(2, n) \geq 2n$

Q4. Montrer que pour $n \in \mathbb{N}$, $A(3, n) \geq 2^n$

Q5. Proposer une minoration de $A(4, n)$.

Cette fonction grandit extrêmement vite : $A(3, 2)$ vaut 29 mais $A(4, 2)$ vaut $2^{65536} - 3$. On peut montrer qu'il est impossible de la calculer uniquement avec des boucles for et des opérations arithmétiques : il est nécessaire de passer par des boucles while ou des fonctions récursives. Vous pouvez vous amuser à l'implémenter en OCaml, à la tracer avec `#trace`, et à constater le nombre immense d'appels même sur de toutes petites entrées.

Exercice 8.

Le retour de Knaster-Tarski

L'objectif de cet exercice est de trouver une preuve du principe d'induction structurale faible, passant par la théorie des treillis et le théorème de Knaster Tarski (voir TD7). On rappelle :

Définition 1. Un ordre $(X, \leq)$ est un treillis complet si et seulement si toute partie $A \subseteq X$ non vide admet une borne inférieure notée $\bigwedge A$ et une borne supérieure notée $\bigvee A$.

Théorème 1 ((Knaster-Tarski)). Soit $(X, \leq)$ un treillis complet. Soit $f : X \rightarrow X$ croissante. Alors f admet un plus petit point fixe.

Cependant, la preuve vue au TD précédent ne permet pas de construire un point fixe facilement. Nous allons étudier un cas où le plus petit point fixe s'écrit comme la limite d'une certaine suite, et en extraire une preuve élégante du raisonnement par induction structurale.

Définition 2. Soit $(X, \leq)$ un treillis complet. On dit qu'une fonction $f : X \rightarrow X$ est **continue** si pour toute partie $A \subseteq X$ totalement ordonnée, on a :

$$f(\bigvee A) = \bigvee (f(A))$$

Q1. Montrer que toute fonction continue est croissante, mais pas l'inverse.

Dans la suite, on pose $(X, \leq)$ un treillis complet et $f : X \rightarrow X$ une fonction continue.

Q2. On pose $p_0 = \perp$, et $p_{n+1} = f(p_n)$ pour $n \in \mathbb{N}$. Montrer que la suite $(p_n)_{n \in \mathbb{N}}$ est croissante.

Q3. Montrer que $p = \bigvee \{p_n \mid n \in \mathbb{N}\}$ est un point fixe de f , et que c'est le plus petit.

La manière dont on construit le point fixe ressemble fortement à la manière dont on a défini les ensembles construits par induction : on itère une bonne fonction sur un élément de base, et on considère la limite de la suite obtenue. On peut en fait utiliser le théorème de Knaster-Tarski pour montrer le principe d'induction faible :

Proposition 2. Soit X un ensemble construit par induction avec les règles $\mathcal{C} = \mathcal{B} \cup \mathcal{I}$. Soit $\mathcal{Q}$ une propriété sur X telle que :

- $\forall (S, 0, P) \in \mathcal{B}, \forall p \in P, \mathcal{Q}(\mathbf{S}(p))$ est vraie
- $\forall (S, r, P) \in \mathcal{I}, \forall p \in P, \forall x_1, \dots, x_r \in X, (\forall i \in \llbracket 1, r \rrbracket, \mathcal{Q}(x_i)) \Rightarrow \mathcal{Q}(\mathbf{S}(p, x_1, \dots, x_r))$

Autrement dit, telle que $\mathcal{Q}$ est vraie sur les éléments de base, et stable par les règles inductives. Alors, $\mathcal{Q}(x)$ est vraie pour tout $x \in X$.

Le but du reste de l'exercice est de trouver une preuve de cette propriété. On pose X un ensemble construit par induction avec les règles $\mathcal{C} = \mathcal{B} \cup \mathcal{I}$, et $\mathcal{Q}$ une propriété sur X telle que :

- $\forall (S, 0, P) \in \mathcal{B}, \forall p \in P, \mathcal{Q}(\mathbf{S}(p))$ est vraie
- $\forall (S, r, P) \in \mathcal{I}, \forall p \in P, \forall x_1, \dots, x_r \in X, (\forall i \in \llbracket 1, r \rrbracket, \mathcal{Q}(x_i)) \Rightarrow \mathcal{Q}(\mathbf{S}(p, x_1, \dots, x_r))$

On souhaite montrer que $\forall x \in X, \mathcal{Q}(x)$ est vraie.

On considère la fonction suivante :

$$\Phi : \begin{array}{ccc} \mathcal{P}(X) & \longrightarrow & \mathcal{P}(X) \\ A & \longmapsto & A \cup B \cup \{S(p, x_1, \dots, x_r) \mid (S, r, P) \in \mathcal{I}, x_1, \dots, x_r \in A, p \in P\} \end{array}$$

Avec $B = \{S(p) \mid (S, 0, P) \in \mathcal{B}, p \in P\}$ l'ensemble des éléments de base de X .

Q4. Que vaut $\Phi(\emptyset)$? et $\Phi(\Phi(\emptyset))$? De manière générale, que vaut $\Phi^n(\emptyset)$?

Q5. Montrer que Φ est croissante et continue.

Q6. En appliquant le résultat des questions 3 et 4, montrer que X est le seul point fixe de Φ .

Q7. On considère maintenant l'ensemble $Q = \{x \in X \mid \mathcal{Q}(x) \text{ est vraie}\}$. Montrer que $Q = X$ et conclure.

Exercice 9.

Existence des fonctions inductives

Prélude : singleton On remarque que pour tout ensemble X , le produit $\{\bullet\} \times X$ est trivialement en bijection avec X via la correspondance $(\bullet, x) \leftrightarrow x$. Dans la suite de cet exercice, lorsqu'un singleton apparaît dans un produit cartésien, on le fera disparaître afin de simplifier les notations.

Nous avons admis en cours l'existence des fonctions définies par induction, mais ce n'est pas immédiat. Par exemple, si l'on considère L l'ensemble des listes d'entiers, la fonction $\text{len} : L \rightarrow \mathbb{N}$ qui calcule la longueur d'une liste vérifie :

- $\text{len}([]) = 0$
- $\text{len}(x :: q) = 1 + \text{len}(q)$ pour $x \in \mathbb{N}, q \in L$

mais rien ne dit que les deux égalités ci-dessus caractérisent entièrement la fonction len .

On considère dans tout cet exercice un ensemble X construit par induction à partir d'un ensemble $\mathcal{C} = \{(C, r, P)\}$ de règles de constructions. On rappelle qu'à partir d'une règle (C, r, P) , on peut construire tous les éléments notés $C(p, x_1, \dots, x_r)$ avec $p \in P$ et $x_1, \dots, x_r \in X$. On partitionne $\mathcal{C}$ en deux ensembles : les règles de base $\mathcal{B}$ ($r = 0$) et les règles inductives $\mathcal{I}$ ($r > 0$).

L'objectif de cet exercice est de justifier que définir une fonction sur les différents constructeurs suffit à définir une **unique** fonction sur X tout entier, via le théorème suivant :

Théorème 2. Soient Y et Z des ensembles quelconques. Soit $(f_C)_{(C,r,P) \in \mathcal{C}}$ une famille de fonctions indexée par les règles de constructions de X , avec pour chaque règle $(C, r, P) \in \mathcal{C}$, $f_C : Z \times P \times Y^r \rightarrow Y$.

Alors, il existe une unique fonction $f : Z \times X \rightarrow Y$ telle que :

$$\forall (C, r, P) \in \mathcal{P}, \forall z \in Z, \forall p \in P, \forall x_1, \dots, x_r \in X, f(z, C(p, x_1, \dots, x_r)) = f_C(z, p, f(z, x_1), \dots, f(z, x_r))$$

Dans ce théorème, Z est l'ensemble des paramètres non récurrents de la fonction que l'on définit, et Y est l'ensemble d'arrivée.

Par exemple, si l'on considère un ensemble E , et l'ensemble $L(E)$ des listes d'éléments de E , défini à partir des règles :

- $([], 0, \{\bullet\})$ (liste vide $[]$)
- $(::, 1, E)$ (liste non vide $x :: q$ avec $x \in E$ et $q \in L(E)$)

On prend $Y = \mathbb{N}$ et $Z = \{\bullet\}$. On considère les deux fonctions suivantes :

$$\begin{aligned} f_{[]} : \begin{array}{ccc} \{\bullet\} & \rightarrow & \mathbb{N} \\ \bullet & \mapsto & 0 \end{array} \\ f_{::} : \begin{array}{ccc} E \times \mathbb{N} & \rightarrow & \mathbb{N} \\ (e, n) & \mapsto & 1 + n \end{array} \end{aligned}$$

Alors, le théorème précédent affirme l'existence d'une unique fonction $f : L(E) \rightarrow \mathbb{N}$ telle que :

- $f([]) = f_{[]}(\bullet) = 0$
- Pour tout $x \in E$, pour tout $q \in L(E)$, $f(x :: q) = f_{::}(x, f(q)) = 1 + f(q)$

On reconnaît la fonction `length: 'a list -> int` qui calcule la taille d'une liste!

Q1. On considère E un ensemble quelconque. Déterminer la fonction $f : E \times L(E) \rightarrow L(E)$ dont l'existence est garantie par l'application du théorème 1 aux deux fonctions suivantes (avec $Y = L(E)$ et $Z = E$) :

$$f_{[]} : \begin{array}{ccc} E & \rightarrow & L(E) \\ e & \mapsto & [] \end{array} \quad f_{::} : \begin{array}{ccc} E \times E \times L(E) & \rightarrow & L(E) \\ (e, x, t) & \mapsto & \begin{cases} t & \text{si } e = x \\ x :: t & \text{si } e \neq x \end{cases} \end{array}$$

Q2. Montrer l'existence de la fonction `map: ('a -> 'b) -> 'a list -> 'b list`.

Intéressons nous maintenant à la preuve du théorème 1. Soient Y et Z des ensembles quelconques. Soit $(f_C)_{(C,r,P) \in \mathcal{C}}$ une famille de fonctions indexée par les règles de constructions de X , avec pour chaque règle $(C, r, P) \in \mathcal{C}$, $f_C : Z \times P \times Y^r \rightarrow Y$.

Pour commencer, nous allons montrer l'unicité. Soient $f, g : Z \times X \rightarrow Y$ deux fonctions telles que :

$$\forall (C, r, P) \in \mathcal{P}, \forall z \in Z, \forall p \in P, \forall x_1, \dots, x_r \in X, f(z, C(p, x_1, \dots, x_r)) = f_C(z, p, f(z, x_1), \dots, f(z, x_r))$$

et

$$\forall (C, r, P) \in \mathcal{P}, \forall z \in Z, \forall p \in P, \forall x_1, \dots, x_r \in X, g(z, C(p, x_1, \dots, x_r)) = f_C(z, p, g(z, x_1), \dots, g(z, x_r))$$

Q3. Montrer par induction structurale sur X que $\forall x \in X, \forall z \in Z, f(z, x) = g(z, x)$.

Montrons maintenant l'existence. On note $X_0, \dots, X_n, \dots$ les différents ensembles définis lors de la construction de X . On note également $h : X \rightarrow \mathbb{N}$ la fonction hauteur.

Q4. Définir une fonction $f : Z \times X \rightarrow Y$ satisfaisant la condition du théorème. *Indication : on pourra définir par récurrence une suite de fonctions $f_n : Z \times X_n \rightarrow Y$ permettant de construire f par étage.*

Indications

- Exercice 1 Question 4 : partez des deux côtés de l'égalité à montrer dans l'étape d'induction, et faites les se rejoindre en déroulant la définition de l'addition.
- Exercice 2 Question 1 : Une induction sur a fait l'affaire. Dans l'étape d'induction, commencez par dérouler les définitions des fonctions, puis voyez quels termes on veut regrouper ensemble.
- Exercice 8 Question 3 : commencez par justifier que $p_0 \leq p_1$, puis que $p_1 \leq p_2$.
- Exercice 8 Question 4 : pour montrer que c'est un plus petit point fixe : prenez un autre point fixe p , et partez du fait que $\perp \leq p$.