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 $\mathbf{int} : \mathcal{P} \to \mathbb{N}$ mettant en bijection $\mathcal{P}$ et $\mathbb{N}$.

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

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

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

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 **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. *Expressions*

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 :

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

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 **syntaxique** 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 $\sigma : \mathcal{X} \to \mathbb{N}$. 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 $\mathbf{replace} : \mathcal{E} \times \mathbb{N}^{\mathcal{X}} \to \mathcal{E}$ suivante :

— $\forall x \in \mathcal{X}, \mathbf{replace}(x, \sigma) = \mathbf{Int}(\sigma(x))$

— $\forall n \in \mathbb{N}, \mathbf{replace}(\mathbf{Int}(n), \sigma) = \mathbf{Int}(n)$

— $\forall e_1, e_2 \in \mathcal{E}, \mathbf{replace}(e_1 + e_2, \sigma) = \mathbf{replace}(e_1, \sigma) + \mathbf{replace}(e_2, \sigma)$

— $\forall e_1, e_2 \in \mathcal{E}, \mathbf{replace}(e_1 \times e_2, \sigma) = \mathbf{replace}(e_1, \sigma) \times \mathbf{replace}(e_2, \sigma)$

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

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

Autrement dit, si l'on remplace les variables d'un 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 évaluant l'expression dans le contexte σ_0.

Exercice 4. *Récursivité terminale*

Q1. Définir une fonction `occ: 'a -> 'a list -> int` permettant de compter le nombre d'occurrences d'une valeur donnée dans une liste. On utilisera une fonction auxiliaire `occ_plus: 'a -> 'a list -> int -> int` récursive terminale.

Q2. Montrer par induction sur la structure des listes que pour toute liste l, pour tout x, et pour tous $a, b \in \mathbb{N}$, `(occ_plus x l a) + b = occ_plus x l (a+b)`.

Q3. Définir une fonction récursive terminale `est_triee: 'a list -> bool` vérifiant si une liste est triée dans l'ordre croissant. On ne passera pas par une fonction auxiliaire.

On suppose qu'on a également défini une fonction `est_triee_rev: 'a list -> bool` vérifiant si une liste est triée dans l'ordre décroissant.

On dit qu'une fonction `f: 'a list -> 'a list` est un **tri** si elle vérifie :

Pout toute liste `l`, `est_triee (f l)` et pour tout élément `x`, `occ x (f l) = occ x l`

Dans les tris, nous allons utiliser deux fonctions de la librairie standard d'OCaml :

— `List.rev: 'a list -> 'a list` permet de renverser une liste.

— `List.append: 'a list -> 'a list -> 'a list` permet de concaténer deux listes.

Q4. Proposer des propriétés sur `List.rev` faisant intervenir `occ_plus`, `est_triee`, et `est_triee rev`.

Q5. Faire de même avec `List.append`.

Dans la suite, on pourra utiliser librement les propriétés proposées aux questions précédentes.

On considère le tri par insertion, exprimé via des fonctions récursives terminales :

```
1  let rec insert (x: 'a) (l: 'a list) (r: 'a list) : 'a list =
2    match l with
3    | [] -> x :: List.rev r
4    | y :: q -> if x < y then List.append (List.rev r) (x :: y :: q)
5                else          insert x q (y::r)
6
7  let rec insert_sort_into (l: 'a list) (res: 'a list) : 'a list =
8    match l with
9    | [] -> res
10   | x :: q -> insert_sort q (insert x res [])
11
12 let insert_sort l = insert_sort_into l []
```

Nous allons commencer par montrer la correction de la fonction `insert`. Notons que pour une fonction d'insertion simple, sans récursivité terminale, la spécification de la fonction serait simplement "Si `l` est triée, alors `insert x l`" est triée. Ici, il nous faut trouver une spécification plus générale, car la fonction `insert` prend en entrée un accumulateur.

Q6. Montrer que pour toute liste `l` et pour tout élément `x`, on a : TODO TODO TODO

`occ_plus x (insert x l r) n = occ_plus x l n + occ_plus x r 0 + 1`.

Q7. En déduire que pour toute liste `l` et pour tout élément `x`, `occ x (insert x l []) = occ x l + 1`.

Q8. Montrer que pour toute liste `l`, pour tout élément `x`, pour tout élément `y` différent de `x`, `occ x (insert y l []) = occ x l`.

Q9. On admet pour cette question que pour toute liste `l`, pour tout élément `x`, `est_triee l` implique que `est_triee (insert x l [])`. Énoncer et montrer la correction du tri insertion.

Q10. Montrer la propriété admise à la question précédente. *Pour cette question, vous pourrez être amenés à devoir introduire de nouvelles propriétés inductives sur les listes.*

Q11. Donner la complexité asymptotique de la fonction `insert_sort` en fonction de n la taille de la liste à trier.

Intéressons nous maintenant au tri fusion.

Q12. Définir une fonction récursive terminale `fusion: 'a list -> 'a list -> 'a list` fusionnant deux listes triées. Énoncer et montrer une propriété de cette fonction par rapport à `occ`.

Q13. Énoncer et montrer une propriété de `fusion` par rapport à `est_triee`.

Q14. Définir une fonction `tri_fusion` implémentant le tri fusion. Montrer la correction de cette fonction.

Exercice 5.

Preuves sur les listes 1

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

```
let rec len (l: 'a list) : int =
  match l with
  | [] -> 0
  | x :: q -> 1 + len q

let rec len2 (l: 'a list) (r: int) : int =
  match l with
  | [] -> r
  | x :: q -> len2 q (r + 1)
```

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

On dit qu'une fonction `f: 'a -> 'a -> '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 :

```
let rec fold_right (f: 'a -> 'b -> 'b) (l: 'a list) (d: 'b) : 'b =
  match l with
  | [] -> d
  | x :: q -> f x (fold_right f q d)

let rec fold_left (f: 'a -> 'b -> 'a) (d: 'a) (l: 'b list) : 'a =
  match l with
  | [] -> d
  | 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 -> 'a -> 'a` une fonction associative et commutative. Alors pour toute liste l et pour tout élément d, `fold_right f l d = fold_left f d l`.

Montrons ce résultat. On pose `f: 'a -> 'a -> '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 `fold_right f l (f d x) = f x (fold_right f l d)`.

Q6. Montrer que pour toute liste l et pour tout élément d, `fold_right f l d = 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 $n \in \mathbb{N}, A(3, n) \geq 2^n$

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 amusez 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 structurelle 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. Soit $(X, \leq)$ un treillis complet. Soit $f : X \to X$ 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 structurelle.

Définition 2. Soit $(X, \leq)$ un treillis complet. On dit qu'une fonction $f : X \to X$ est continue [1] si et seulement 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 $f : X \to X$ une fonction continue.

Q2. On pose $p_0 = \bot$, 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

— $\forall (S, r, P) \in \mathcal{I}, \forall p \in P, \forall x_1, \ldots, x_r \in X, (\forall i \in [\![1, r]\!], \mathcal{Q}(x_i)) \Rightarrow \mathcal{Q}(\mathbf{S}(p, x_1, \ldots, x_r))$

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

— $\forall (S, r, P) \in \mathcal{I}, \forall p \in P, \forall x_1, \ldots, x_r \in X, (\forall i \in [\![1, r]\!], \mathcal{Q}(x_i)) \Rightarrow \mathcal{Q}(\mathbf{S}(p, x_1, \ldots, x_r))$

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

On considère la fonction suivante :

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

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

1. La terminologie technique est *chain-continuous*, car les parties totalement ordonnées d'un ensemble sont appelées des chaînes.

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.

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 $\bot \leq p$.