

MATHÉMATIQUES POUR L'INFORMATIQUE
EXAMEN FINAL 2008

Induction structurelle

Exercice 1. (environ 12 points) Considérons un ensemble E_1 défini par induction structurelle à partir de l'ensemble de bases \mathcal{B}_1 et de l'ensemble d'opérateurs d'induction \mathcal{I}_1 . Considérons de même un ensemble E_2 défini par induction structurelle à partir de l'ensemble de bases \mathcal{B}_2 et de l'ensemble d'opérateurs d'induction \mathcal{I}_2 .

Prouver que l'ensemble $E_1 \times E_2$ peut se définir par induction structurelle à partir de l'ensemble des bases $\mathcal{B}_1 \times \mathcal{B}_2$ et d'un ensemble d'opérateurs d'induction que vous préciserez.

Note La difficulté ici consiste à déterminer un bon ensemble d'opérateurs d'induction. Il est bien entendu absurde d'utiliser $\mathcal{I}_1 \cup \mathcal{I}_2$ ou $\mathcal{I}_1 \times \mathcal{I}_2$ qui n'ont pas le bon ensemble de définition. Par contre, il est possible de ruser un peu en modifiant un peu les opérateurs d'induction pour leur permettre de gérer les arguments supplémentaires qui apparaissent.

Démonstration. Commençons par construire nos opérateurs d'induction.

Soit i_1 dans \mathcal{I}_1 une fonction d'arité a_1 , c'est-à-dire une fonction de $E_1^{a_1}$ vers E_1 . Nous allons étendre i_1 en un ensemble de fonctions d'arité a_1 sur $E_1 \times E_2$, c'est-à-dire une fonction de $(E_1 \times E_2)^{a_1}$ vers $E_1 \times E_2$. Pour ce faire, pour tout k de $\llbracket 1, a_1 \rrbracket$, notons i_1^k la fonction de $(E_1 \times E_2)^{a_1}$ vers $E_1 \times E_2$ définie par $(x_1, y_1), (x_2, y_2), \dots, (x_{a_1}, y_{a_1}) \mapsto (i_1(x_1, x_2, \dots, x_{a_1}), y_k)$. Il existe k telles fonctions, qui chacune permettra d'appliquer l'opérateur i_1 aux x et de garder des opérations qui ont déjà été effectuées sur les y . Regroupons-les en un ensemble $i_1^< = \{i_1^k/k \in \llbracket 1, a_1 \rrbracket\}$.

De la même manière, définissons pour tout i_2 de \mathcal{I}_2 et d'arité a_2 les fonctions $i_2^k: ((x_1, y_1), (x_2, y_2), \dots, (x_{a_2}, y_{a_2})) \mapsto (x_k, i_2(y_1, y_2, \dots, y_{a_2}))$ et regroupons-les en un ensemble $i_2^> = \{i_2^k/k \in \llbracket 1, a_2 \rrbracket\}$

Soit alors $\mathcal{I}_{1 \times 2}$ l'ensemble défini comme $\bigcup_{i_1 \in \mathcal{I}_1} i_1^< \cup \bigcup_{i_2 \in \mathcal{I}_2} i_2^>$. Cet ensemble est bien un ensemble de fonctions de $(E_1 \times E_2)^k$ vers $E_1 \times E_2$, pour certaines valeurs de k , et que nous pouvons donc utiliser comme ensemble d'opérateurs d'induction pour définir un sous-ensemble de $E_1 \times E_2$. Reste à prouver que nous définissons bien l'ensemble demandé.

Soit E l'ensemble défini par induction à partir de l'ensemble de bases $\mathcal{B}_1 \times \mathcal{B}_2$ et de l'ensemble des opérateurs d'induction $\mathcal{I}_{1 \times 2}$. Prouvons que $E = E_1 \times E_2$.

Inclusion. Ce cas est trivial, on pouvait l'évacuer en quelques phrases. Cela dit, dans un souci d'exhaustivité, présentons-le ici en détails.

Prouvons par induction structurelle sur E que $E \subseteq E_1 \times E_2$. Soit pour tout x dans E l'hypothèse $\mathcal{H}(x)$ "x est un élément de $E_1 \times E_2$ ".

Initialisation. Soit b dans $\mathcal{B}_1 \times \mathcal{B}_2$. Comme $\mathcal{B}_1 \subseteq E_1$ et $\mathcal{B}_2 \subseteq E_2$, alors b est bien un élément de $E_1 \times E_2$. Ce qui prouve l'initialisation.

Héritage. Considérons un élément z de E qui peut s'écrire $i(z_1, \dots, z_n)$ où z_1, \dots, z_n sont des éléments de E tels que $\mathcal{H}(z_1), \dots, \mathcal{H}(z_n)$ soient vraies. Prouvons que $\mathcal{H}(z)$ est vraie elle aussi.

Par hypothèse d'induction, pour tout z_k , comme $\mathcal{H}(z_k)$ est vraie, z_k peut s'écrire (x_k, y_k) avec $x_k \in E_1$ et $y_k \in E_2$. De plus, par définition de $\mathcal{I}_{1 \times 2}$, soit il existe i_1 dans \mathcal{I}_1 tel que $i \in i_1^<$, soit il existe i_2 dans \mathcal{I}_2 tel que $i \in i_2^>$. Les deux cas sont presque identiques, nous nous contenterons donc de traiter le cas $i \in i_1^<$.

Par définition de $i_1^<$, i s'écrit i_1^k pour un certain k et z s'écrit donc $(i_1(x_1, x_2, \dots, x_{a_1}), y_k)$. Or, par définition de E_1 , comme tous les x_i sont dans E_1 et comme i_1 est dans \mathcal{I}_1 , $i_1(x_1, x_2, \dots, x_{a_1})$ est dans E_1 . Comme, de plus, nous avons déjà prouvé que y_k est dans E_2 , nous pouvons en conclure que $(i_1(x_1, x_2, \dots, x_{a_1}), y_k) \in E_1 \times E_2$, c'est-à-dire que $z \in E_1 \times E_2$. Ce qui prouve l'héritage et achève la preuve par induction.

Deuxième inclusion. Prouvons maintenant que $E_1 \times E_2 \subseteq E$. Pour ce faire, choisissons (x, y) dans $E_1 \times E_2$ et trouvons comment construire cet élément (x, y) par induction à partir de $\mathcal{B}_1 \times \mathcal{B}_2$ et des opérateurs de $\mathcal{I}_{1 \times 2}$.

Commençons par prouver l'hypothèse d'induction structurelle $\mathcal{H}(x)$ "Pour tout y dans E_2 , (x, y) est un élément de E ".

Initialisation. Si x s'écrit b_1 avec b_1 dans \mathcal{B}_1 , prouvons par induction l'hypothèse que $\mathcal{H}'(y)$ " (x, y) est un élément de E " est vraie pour tout y dans E_2 .

Initialisation. Si y s'écrit b_2 avec b_2 dans \mathcal{B}_2 , nous avons $(x, y) = (b_1, b_2)$ un élément de $\mathcal{B}_1 \times \mathcal{B}_2$. Comme $\mathcal{B}_1 \times \mathcal{B}_2 \subseteq E$, nous avons prouvé l'initialisation.

Héritage. Si y s'écrit $i_2(y_1, \dots, y_{a_2})$ pour un certain opérateur i_2 de \mathcal{I}_2 d'arité a_2 et avec y_1, y_2, \dots, y_{a_2} tels que $\mathcal{H}'(y_1), \dots, \mathcal{H}'(y_{a_2})$ soient vraies. D'après \mathcal{H}' , nous savons alors que (x, y_1) est un élément de E , (x, y_2) est un élément de E , ... et (x, y_{a_2}) est un élément de E .

Or, si k est un entier de $\llbracket 1, a_2 \rrbracket$, si nous appliquons la fonction i_2^k à a_2 éléments de E , le résultat est donc bien un élément de E . En particulier, $i_2^k((x, y_1), \dots, (x, y_{a_2}))$ est bien un élément de E , c'est-à-dire (x, y) est bien un élément de E . Ce qui prouve l'héritage et achève la preuve par induction.

Héritage. Ce cas est essentiellement identique à l'héritage précédent. Maintenant, si x s'écrit $i_1(x_1, \dots, x_{a_1})$ pour un certain opérateur i_1 de \mathcal{I}_1 d'arité a_1 et certains x_1, x_2, \dots, x_{a_1} tels que $\mathcal{H}(x_1), \dots, \mathcal{H}(x_{a_1})$ soient vraies. Soit alors y dans E_2 . Par hypothèse d'induction, $(x_1, y) \in E, \dots, (x_{a_1}, y) \in E$. Prouvons que $(x, y) \in E$.

Or, si k est un entier de $\llbracket 1, a_1 \rrbracket$, si nous appliquons la fonction i_1^k à a_1 éléments de E , le résultat est donc bien un élément de E . En particulier, $i_1^k((x_1, y), \dots, (x_{a_1}, y))$ est bien un élément de E , c'est-à-dire (x, y) est bien un élément de E . Ce qui prouve l'héritage et achève notre grande preuve par induction.

Nous avons donc prouvé que $E_1 \times E_2 \subseteq E$ et $E \subseteq E_1 \times E_2$, c'est-à-dire $E = E_1 \times E_2$. En d'autres termes, l'ensemble $E_1 \times E_2$ peut se définir par induction à partir des bases $\mathcal{B}_1 \times \mathcal{B}_2$ et des opérateurs d'induction $\mathcal{I}_{1 \times 2}$. \square

Outils pour la complexité

Exercice 2. (environ 8 points) Considérons deux suites u et v à valeurs dans \mathbf{N} . Est-il vrai que si $\lim_{n \rightarrow +\infty} \frac{u_n}{v_n}$ existe et n'est pas $+\infty$, alors $u = \mathcal{O}(v)$? Prouvez-le.

Démonstration. Prouvons que tel est bien le cas. Pour ce faire, supposons l'existence de deux suites u et v à valeurs dans \mathbf{N} telles que $\lim_{n \rightarrow +\infty} \frac{u_n}{v_n}$ existe et n'est pas $+\infty$. Appelons l cette limite et notons, par définition de la limite,

$$\forall \epsilon > 0, \exists N \in \mathbf{N}, \forall n \geq N, \left| \frac{u_n}{v_n} - l \right| < \epsilon \quad (1)$$

Rappelons que nous souhaitons prouver que $u = \mathcal{O}(v)$, c'est-à-dire, par définition

$$\exists K \in \mathbf{R}^+, \exists N \in \mathbf{N}, \forall n \geq N, \left| \frac{u_n}{v_n} \right| \leq K \quad (2)$$

Pour prouver que (2) est vrai, il nous revient de trouver un bon K . Pour ce faire, fixons arbitrairement $\epsilon = 0,5$. Choisissons un N tel que décrit précédemment et $n \geq N$.

Nous avons donc $\left| \frac{u_n}{v_n} - l \right| < \epsilon$, ce dont nous pouvons déduire par deuxième inégalité triangulaire que $\left| \frac{u_n}{v_n} \right| - |l| < \epsilon$ et donc $\left| \frac{u_n}{v_n} \right| < |l| + \epsilon$. Posons alors $K = |l| + \epsilon$. Nous venons de prouver $\exists N \in \mathbf{N}, \forall n \geq N, \left| \frac{u_n}{v_n} \right| \leq K$.

Nous pouvons donc conclure que oui, $u = \mathcal{O}(v)$. \square

Exercice 3. (environ 8 points) Nous nous plaçons sur \mathbf{Z} . Trouver une suite u croissante et une suite v décroissante telles que $u + v$ ne soit pas monotone.

Démonstration. Rappelons qu'une suite est monotone si elle est soit toujours croissante, soit toujours décroissante, soit constante. À nous donc de trouver u et v telles que $u + v$ soit parfois croissante, parfois décroissante.

Soit u la suite définie par $\forall n \in \mathbf{N}, u_n = 2^{n+100} + (-1)^n$. Prouvons que u est bien croissante. Pour ce faire, posons $n \geq 0$ et calculons $w_n = u_{n+1} - u_n$. Nous avons $w_n = 2^{n+100} - 2 \times (-1)^n$. Trivialement, $|2 \times (-1)^n| \leq 2$ et $2^{n+100} > 4$ donc $w_n > 0$. Nous en déduisons que $\forall n, u_{n+1} > u_n$, c'est-à-dire que u est (strictement) croissante.

Soit v la suite définie par $\forall n \in \mathbf{N}, v_n = -2^{n+100}$. Sans difficultés, v est (strictement) décroissante.

Or, $u + v$ est la suite $((-1)^n)_{n \in \mathbf{N}}$ qui n'est franchement pas monotone. □

Démonstration. Nous pouvons prouver un résultat plus fort, à savoir que ceci est vrai même dans \mathbf{N} . Pour ce faire, choisissons la suite u définie par

$$\begin{cases} u_0 = 0 \\ u_1 = 1 \\ \forall n \geq 2, u_n = 100 + n \end{cases}$$

et la suite v définie par

$$\begin{cases} v_0 = 100 \\ n \geq 1, v_n = 0 \end{cases}$$

La suite u est croissante, trivialement. La suite v est décroissante, tout aussi trivialement. Quant à la suite $u + v$, ses valeurs successives sont 100, 1, 102, 103, ... Elle n'est donc pas monotone. □

Note Notons cependant que, si nous nous plaçons dans \mathbf{N} , au delà d'un certain N , la suite v va forcément être constante. Par conséquent, au-delà de ce N , la suite $u + v$ sera croissante.

Complexité

Exercice 4. (environ 8 points) Les trois extraits suivants définissent la même fonction, en Java, en OCaml et en pseudo-langage algorithmique. Quelle est la complexité de cette fonction ? La réponse est la même pour les trois extraits, utilisez celui qui vous convient le mieux.

Version Java

```
public static List<Integer> garderPairs(List<Integer> nombres)
{
    List<Integer> resultat = new LinkedList<Integer>;
    for(Integer i : nombres)
        if(i % 2 == 0)
            resultat.addFirst(i);
    return resultat;
}
```

Version OCaml

```
value garder_pairs nombres =
    List.filter (fun i -> i mod 2 = 0) nombres;
```

Version algorithmique

Fonction GarderPairs

Argument : *Nombres* (une liste de nombres)
 Créer une nouvelle liste de nombres appelée *Résultats*, initialement vide
 Pour chaque élément i de *Nombres*
 Si i est pair
 Ajouter i comme premier élément à *Résultats*

```
Sinon
  Ne rien faire
Fin Si
Fin Pour
Renvoyer Résultats.
Fin Fonction
```

Démonstration. Le raisonnement est à peu près le même pour les trois extraits. Notons

$$\left\{ \begin{array}{l} \alpha \text{ le temps nécessaire pour créer la liste vide des résultats} \\ \beta \text{ le temps nécessaire pour ajouter un élément en tête de liste} \\ \gamma \text{ le temps nécessaire pour vérifier si un nombre est pair} \\ n \text{ la longueur de la liste } \mathit{nombre}s \end{array} \right.$$

Dans chacune des versions, la boucle est invoquée sur sur la liste *nombre*s. Le contenu de cette boucle sera donc exécuté au plus n fois. À chaque passage dans la boucle, le programme exécute une vérification de parité, en temps γ , et peut-être un ajout en tête de liste, en temps β . En plus de cela, avant de commencer la boucle, le programme peut créer une liste vide, en temps α (notons que cette création n'a pas lieu en OCaml, puisqu'il n'est jamais nécessaire de créer une nouvelle liste vide, mais cela ne change rien de fondamental à notre résultat).

Nous avons donc un temps total majoré par $\alpha + n \times (\beta + \gamma)$.

En d'autres termes, cette complexité est un $\mathcal{O}(n)$. □