Chapitre 3
Bases mathématiques de la progammation récursive

I Introduction

Jusqu'à présent, nous nous sommes contentés d'une définition informelle de la récursivité : est récursive toute fonction qui intervient dans sa définition. Bien entendu, ce qualificatif est à mettre en rapport avec le principe de récurrence, dont on peut rappeler l'énoncé :

Théorème 1 (principe de récurrence simple)

Soit $\mathcal{P}$ un prédicat défini sur $\mathbb{N}$, tel que $\mathcal{P}(0)$ est vrai, ainsi que l'implication $\forall n \in \mathbb{N}^*, \mathcal{P}(n-1) \implies \mathcal{P}(n)$. Alors pour tout entier $n \in \mathbb{N}, \mathcal{P}(n)$ est vrai.

Ce principe permet de démontrer que la donnée d'un élément $a \in E$ et d'une application $f : E \to E$ permet de définir sans ambiguïté une suite $(u_n)_{n\in\mathbb{N}}$ à l'aide des relations :

$$u_0 = a \quad \text{et} \quad \forall n \in \mathbb{N}^*, u_n = f(u_{n-1})$$

et que tout élément de cette suite peut être calculé à l'aide d'une définition qui suivra le schéma suivant :

```
let rec u n = match n with
  | 0 -> a
  | n -> f (u (n - 1)) ;;
```

Nous dirons dans ce cas que la fonction se termine, autrement dit que l'appel `u n` lorsque n est un entier naturel retournera une réponse en un temps fini.

Mais toutes les fonctions récursives ne suivent pas un modèle aussi simple. La fonction Q de Hofstadter, par exemple, est « définie » sur $\mathbb{N}^*$ par les relations :

```
let rec q = function
  | 1 -> 1
  | 2 -> 1
  | n -> q (n - q (n - 1)) + q (n - q (n - 2)) ;;
```

cependant la preuve de sa terminaison reste un problème ouvert. Autrement dit, il n'a pas encore été prouvé que pour un entier n quelconque, le calcul de $Q(n)$ retournera un résultat en un nombre fini d'étapes !

On va se limiter ici aux fonctions récursives. Mais nous étudierons également plus tard des structures récursives (par exemple les arbres), et nous introduirons en cours les preuves par induction structurelle sur le seul type récursif que nous connaissons, à savoir les listes.

Intermède culturel : le problème de l'arrêt Le problème de l'arrêt consiste à déterminer si le calcul d'une fonction donnée se termine ou pas. Turing a démontré qu'il s'agit d'un problème indécidable (au sens algorithmique du terme), autrement dit qu'il ne peut exister de moyen algorithmique de déterminer à l'avance si un calcul va se terminer ou pas. Sa preuve repose sur le principe de la diagonale de Cantor.

Il faut commencer par observer qu'un programme n'est rien d'autre qu'une suite finie de bits (son écriture machine en code binaire par exemple) ; autrement dit, si on se restreint par exemple à l'ensemble $\mathcal{F}$ des fonctions Caml de type `int -> int` , alors $\mathcal{F}$ est en bijection avec un sous-ensemble de l'ensemble des suites finies sur $\{0, 1\}$. Ce dernier ensemble étant dénombrable, il en est de même de $\mathcal{F}$. Il existe donc une bijection $\varphi : \mathbb{N} \to \mathcal{F}$. Ainsi on peut identifier n'importe quel programme par un entier (ci-dessous $\varphi(p)$ par exemple).

Nous allons maintenant supposer qu'il existe une fonction `termine` de type `int -> int -> bool` capable de déterminer si un calcul se termine ou pas. Autrement dit,

$$\texttt{termine p q} = \begin{cases} \texttt{true} & \text{si le calcul de } \varphi(p)(q) \text{ se termine} \\ \texttt{false} & \text{sinon} \end{cases} \quad (1)$$

Considérons alors la fonction :

```
let rec f p = match p with
  | p when termine p p -> f p
  | p -> 0 ;;
```

et notons $r \in \mathbb{N}$ tel que $f = \varphi(r)$ (ce qui est possible en raison de la bijection évoquée plus haut).

Il reste à considérer la valeur de `termine r r` pour aboutir à une absurdité : si `termine r r = true` alors `f r` bouclera indéfiniment ce qui est absurde, mais si `termine r r = false` alors `f r` retournera 0, ce qui est tout aussi contradictoire !

Sans prétendre à l'exhaustivité, nous allons maintenant définir un cadre formel plus général que le simple principe de récurrence pour prouver la terminaison et la correction des fonctions récursives que nous seront conduits à rencontrer dans la suite de ce cours.

II Ensembles bien fondés

II.1 Généralités et définitions

Commençons par rappeler la définition d'un ordre. Une relation binaire $\mathcal{R}$ sur un ensemble E est un ordre s'il elle est :

- Réflexive : $\forall x \in E, x\mathcal{R}x$
- Antisymétrique : $\forall x, y \in E^2, x\mathcal{R}y$ et $y\mathcal{R}x \implies x = y$
- Transitive : $\forall x, y, z \in E^3, x\mathcal{R}y$ et $y\mathcal{R}z \implies x\mathcal{R}z$

On note plutôt $\preccurlyeq$ les relations d'ordre. Un ensemble muni d'un ordre est un ensemble **ordonné**.

Enfin on note et définit l'ordre strict $\prec$ associé à $\preccurlyeq$ par $x \prec y <=> x \preccurlyeq y$ et $x \neq y$

Rappelons ensuite la différence entre élément minimal et plus petit élément. Si $(E, \preccurlyeq)$ est un ensemble ordonné, A une partie non vide de E et $a \in A$, on dit que :

- a est un élément minimal de A lorsque pour tout $x \in A, x \preccurlyeq a \implies x = a$;
- a est le plus petit élément de A lorsque pour tout $x \in A, a \preccurlyeq x$.

Bien entendu, le plus petit élément, s'il existe, est un élément minimal. En revanche, l'implication réciproque n'est pas toujours vraie dès lors que l'ordre n'est pas total. En outre, une partie peut posséder plusieurs éléments minimaux, mais au plus un plus petit élément

Définition

On dit qu'un ensemble ordonné $(E, \preccurlyeq)$ est bien fondé *(ou que $\preccurlyeq$ est un ordre bien fondé) lorsque toute partie non vide possède (au moins) un élément minimal*

Nous montrerons en exercice la proposition suivante :

Proposition 2 (caractérisation équivalente)

Un ensemble ordonné $(E, \preccurlyeq)$ est bien fondé si et seulement s'il n'existe pas de suite strictement décroissante dans E

Pour se faire une image cela veut également dire que dans un ensemble bien fondé, toute suite infinie décroissante est stationnaire.

On peut définir des ordres à partir d'autres ordres.

Ordre produit :

Définition

On définit l'ordre produit sur deux ensembles ordonnés $(E_1, \preccurlyeq_1)$ et $(E_2, \preccurlyeq_2)$ par $(x_1, x_2) \preccurlyeq (y_1, y_2)$ si et seulement si $x_1 \preccurlyeq_1 y_1$ et $x_2 \preccurlyeq_2 y_2$.

Proposition 3

L'ordre produit de deux ordres bien fondés est bien fondé.

Exemple Lorsqu'on munit $\mathbb{N}^2$ de l'ordre produit : $(a, b) \preccurlyeq (a', b') \iff a \leqslant a'$ et $b \leqslant b'$, on obtient un ensemble bien fondé.

Soit A une partie non vide de $\mathbb{N}^2$. Exhibons un élément minimal de A. On vérifie facilement que l'élément (a_0, b_0) défini par :

$a_0 = \min\{a \in \mathbb{N} \quad | \quad \exists b \in \mathbb{N} \text{ tq } (a, b) \in A\}$ et $b_0 = \min\{b \in \mathbb{N} | (a_0, b) \in A\}$ en est **un** élément minimal.

Ordre lexicographique :

Définition

On définit l'ordre lexicographique sur deux ensembles ordonnés $(E_1, \preccurlyeq_1)$ et $(E_2, \preccurlyeq_2)$ par $(x_1, x_2) \preccurlyeq (y_1, y_2)$ si et seulement si $x_1 \prec_1 y_1$ ou $(x_1 = y_1$ et $x_2 \preccurlyeq_2 y_2)$.

Proposition 4

L'ordre lexicographique de deux ordres bien fondés est bien fondé.

Exemple Muni de l'ordre lexicographique : $(a, b) \preccurlyeq (a', b') \iff a < a'$ ou $(a = a'$ et $b \leqslant b')$, $(\mathbb{N}^2, \preccurlyeq)$ est bien fondé.

Remarque Plus généralement, on peut définir l'ordre lexicographique sur l'ensemble des suites finies d'éléments d'un ensemble bien ordonné. C'est bien entendu l'ordre utilisé dans les dictionnaires.

II.2 Principe d'induction

Le principe de récurrence se généralise alors de la manière suivante :

Théorème 5 (principe d'induction)

Soit $(E, \preccurlyeq)$ un ensemble bien fondé et une propriété $\mathcal{P}$ sur E
Si :

- *(cas de base) pour tout a minimal de E, $\mathcal{P}(a)$ est vrai ;*
- *(induction) le fait que pour tout élément x non minimal de E, $\mathcal{P}(z)$ soit vérifiée pour tout élément $z \prec x$ implique $\mathcal{P}(x)$;*

alors $\mathcal{P}(x)$ est vrai pour tout élément x de E.

Preuve. Soit X l'ensemble des éléments x de E tel que $\mathcal{P}(x)$ soit faux, et supposons X non vide. X possède donc un élément minimal x_0 (dans X). Nécessairement $\mathcal{P}(x_0)$ est faux et donc x_0 ne peut être un élément minimal de E (cas de base). Considérons l'ensemble Y des éléménts strictement inférieurs à x_0. Il est non vide (sinon x_0 serait un élément minimal de E). On peut trouver $y_0 \in Y$ tel que $\mathcal{P}(y_0)$ soit faux, sinon par le cas d'induction $\mathcal{P}(x_0)$ serait vrai. On a donc trouvé un élément y_0 strictement plus petit que x_0 tel que $\mathcal{P}(y_0)$ soit faux qui est donc dans X alors que x_0 est un élément de X ! Il y a donc contradiction. Donc X est l'ensemble vide.

II.3 Fonctions inductives

Le principe d'induction permet de justifier qu'on définit sans ambiguïté une fonction $f : E \to F$ en procédant de la manière suivante. $(E, \preccurlyeq)$ étant un ensemble bien fondé, soit M l'ensemble des éléments minimaux de E, $g : M \to F$ et $h : E\backslash M \times F^n \to F$ deux fonctions quelconques (non inductives, et dont le calcul des valeurs se fait en un nombre fini d'étapes).

On pose

$$\begin{aligned} \forall a \in M, \quad & f(a) = g(a) \\ \forall x \in E\backslash M, \quad & f(x) = h(x, f(z_1), \cdots, f(z_n)) \text{ où } \forall i, 1 \leqslant i \leqslant n, z_i \prec x \end{aligned}$$

Une fonction ainsi définie est dite **inductive**.

On peut élargir cette définition en remplaçant M par une partie A incluant M. Dans les deux cas cet ensemble constitue l'ensemble des **cas de base**.

Pour les autres cas on voit que le calcul de $f(x)$ se fait en utilisant un nombre fini d'autres calculs de f avec des arguments strictement inférieurs à x.

Pour prouver la terminaison du calcul de $f(x)$, on utilise la proposition $\mathcal{P}(x) =$ « le calcul de `f(x)` se termine en un nombre fini d'étapes ».

Pour les éléments a minimaux (cas de base) $f(a)$ se calcule par $g(a)$ et donc termine.

Soit un élément x non minimal quelconque. On suppose que le calcul de $f(z)$ se termine pour tous les $z \prec x$. Pour calculer $f(x)$ on a besoin de calculer un nombre fini de valeurs $y_i = f(z_i)$ avec $z_i \prec x$, qui est un calcul qui par hypothèse prend un temps fini. Puis on calcule $h(x, y_1, \cdots, y_n)$ qui se fait aussi en un temps fini. Donc $\mathcal{P}(x)$ est vraie.

Par le principe d'induction bien fondée le calcul de $f(x)$ se termine pour toute valeur de $x \in E$.

Exemple D'après l'algorithme d'Euclide, la fonction pgcd peut-être définie inductivement par les relations :

$$\forall q \in \mathbb{N}, \text{pgcd}(0, q) = q \quad \text{et} \quad \forall (p, q) \in \mathbb{N}^* \times \mathbb{N}, \text{pgcd}(p, q) = \text{pgcd}(q \bmod p, p).$$

Nous avons ici $E = \mathbb{N}^2$, L'ensemble des cas de bases est $B = \{(0, q) | q \in \mathbb{N}\}$, $g : (x, y) \mapsto y$ et $h : (x, y) \mapsto y$. Pour les cas qui ne sont pas des cas de bases le calcul de $f(x)$ se fait par $h(x, f(z_1))$ avec $z_1 = (q \bmod p, p)$ si $x = (p, q)$

Lorsqu'on munit $\mathbb{N}^2$ de l'ordre lexicographique, on a bien : $\forall (p, q) \in E\backslash B, (q \bmod p, p) \prec (p, q)$. Ceci nous assure que la fonction définie ci-dessous se termine :

```
let rec pgcd p q = match p, q with
  | 0, q -> q
  | p, q -> pgcd (q mod p) p ;;
```

Par ailleurs le principe d'Euclide nous assure en plus que le résultat calculé est bien le pgcd des entiers p et q.

Remarque Toute fonction définie inductivement va être calculable à l'aide d'une définition suivant peu ou prou le schéma suivant :

```
let rec f = function
  | x when Cas_De_Bases x -> g x
  | x -> h x f(z1) ... f(zn) ;;
```

à condition d'avoir au préalable défini les fonctions `dans_A : 'a -> bool` , `phi : 'a -> 'a` , `g : 'a -> 'b` et `h : 'a -> 'b -> 'b` .

II.4 Démarche générale pour l'algorithmique récursive

Pour résoudre récursivement un problème algorithmique on peut procéder ainsi :

On commence par concevoir l'algorithme récursif (ce qui définit une fonction inductive)

- on choisit l'ensemble ordonnée $(E, \preccurlyeq)$ bien fondé adapté au problème ;
- on détermine les cas de base (qui doit inclure les éléments minimaux de E), pour lesquels le résultat est immédiat ;

— pour les autres valeurs de x, en supposant qu'on est capable de calculer $f(y)$ pour y tel que $y \prec x$, on explicite la façon de calculer $f(x)$ en fonction d'un certain nombre de telles valeurs $f(y)$.

Alors les résultats précédents assurent que l'algorithme termine. Autrement dit, l'algorithme termine car les paramètres des appels récursifs décroissent strictement dans un ensemble bien fondé, donc forment nécessairement des suites finies. La preuve est la même que celle donnée plus haut ($\mathcal{P}(x)$ = « le calcul de `f(x)` se termine en un nombre fini d'étapes ».)

Pour prouver la correction de l'algorithme, on utilise une proposition $\mathcal{P}(x)$, dite **invariant d'appel récursif**, du type « le calcul de `f(x)` fournit une solution au problème », qui doit vérifier les propriétés :

— pour tout élément x faisant partie des cas de base, $\mathcal{P}(x)$ est vraie ;

— pour tout les autres éléments, si on suppose que $\mathcal{P}(y)$ est vraie pour tout y tel que $y \prec x$, alors $\mathcal{P}(x)$ est vraie.

Alors d'après le principe d'induction, la proposition est toujours vraie, donc l'algorithme calcule dans tous les cas ce qu'on attend de lui.

En pratique, l'ensemble bien fondé est souvent $\mathbb{N}$ muni de l'ordre usuel.