CHAPITRE 4 : Bijection et Ensembles énumérables

Michaël PÉRIM

March 7, 2018

Contents

1 Rappel des définitions

1.0.1 une relation R : AxB est une application si tout élément a:A possède *au moins une* image par R

au moins une image : QQ a:A, EX b:B, a R b

1.0.2 une relation R : AxB est une fonction si chaque élément a:A possède *au plus une* image par R

au plus une image : QQ a:A QQ b1,b2:B . a R b1 /\ a R b2 ==> b1 = b2

C'est la technique (surprenante mais très puissante) des mathématiciens pour dire qu'il n'y a qu'une seule image : « s'il y en avait deux b1 et b2 ce serait la même ! »

Notation puisque chaque a est lié à *au plus un* b on s'autorise à écrire R : A −> B au lieu de R : AxB et R(a) = b au lieu de a R b

1.0.3 une relation R:AxB est injective si deux éléments de A ne sont jamais en relation avec un même élément de B

QQ a1,a2:A. QQ b:B. a1 R b /\ a2 R b ===> a1=a2

Remarque: Cette propriété est utile pour coder et décoder.

Imaginez qu'on veuille coder les lettres a,b,c,d,... par des nombres et qu'on choissine le codage suivant a —> 0 , b —> 0, c —> 1, d —> 2, ... Dans ce condition le mot "baba" se code 0000.

Ce codage n'est pas injectif puisqu'on envoie "a" et "b" sur le même nombre. ce qui pose un problème lorsqu'on veut décoder 0000. On ne sait pas à quel mot cela correspondait car il y a plusieurs possibilités (16 en fait) : aaaa, aaab, aabb, abbb, bbbb, baba, abba, baab, ...

1.0.4 une relation R:AxB est surjective si tout élément de B a *au moins un* antécédent par R dans A

QQ b:B, EX a:A, a R b

1.0.5 une relation R:AxB est une bijection si c'est une application et une fonction, injective et surjective

2 Utilités des bijections

2.1 Représentation de la mémoire :

La mémoire est un tableau M à une dimension de cases pouvant contenir un entier 64 bits Les cases sont numérotées de 0 à N où N est la taille capacité mémoire de votre machine. (sur les machines récentes N = 1 Go = 10^9 cases)

M[1],. . . ., M[N]

2.2 Représentation d'un tableau T[L][C] en mémoire :

Lorsque dans un programme vous utilisez un tableau T[L][C] celui-ci doit être enregistré dans la mémoire M

il faut donc associer à la case de coordonnées (l,c) une case mémoire m afin que M[m] = T[l][c]

On cherche donc une bijection entre les couples (l,c) et les indices m de la mémoire

L x C <˜ ˜> [0..L*C-1] (l , c) —> m = l*C+c avec c < C (m div C, m mod C) <— m

Remarques :

1. c'est le codage utilisé par le compilateur gcc pour représenter en mémoire les tableaux à plusieurs dimensions.

2. on constate que la bijection a besion de connaître le nombre de colonnes. Voilà pourquoi dans le langage C il faut donner la taille des tableaux.

2.3 Généralisation à des tableaux 2D de taille non connue à l'avance

Certains langages de programmation modernes ne demandent pas à l'avance la taille des tableaux. Ils ne peuvent donc pas utiliser la bijection précédente.

Puiqu'on ne connaît pas le nombre C de colonnes (elle peut être quelconque dans Nat) on doit trouver une bijection qui ne dépendent pas de la taille L x C du tableau.

On cherche donc **une bijection Nat x Nat -> Nat**

[1] DEFINITION NOT FOUND.

Avant de chercher à la constuire demandons-nous si une telle bijection existe-elle ? (sinon on va chercher longtemps et pour rien !)

2.3.1 Le théorème de Cantor-Bernstein permet de répondre à la question de l'existence.

Théorème de Cantor-Bernstein

Soient E et F deux ensembles si il existe deux fonctions injectives g: E -> F h: F -> E alors il existe une bijection entre E et F.

On note alors E <~ > F et on dit que les ensembles ont la même taille.

2.3.2 Applications au cas Nat x Nat <~?~> Nat

g: Nat x Nat –> Nat (l , c) –> n = $2^{l}*3^{c}$ est une injection puisque la décomposition d'un nombre n en facteur premier est unique

h: Nat –> Nat x Nat n –> (n , n) est une injection (trivial)

Donc le théorème de Cantor-Bernstein nous assure qu'il existe une bijection Nat x Nat <~ > Nat mais il ne dit pas laquelle...

2.3.3 Bijection de Cantor Nat x Nat <~ > Nat

on définit deux fonctions f1 : Nat x Nat –> Nat et f2 : Nat –> Nat x Nat qui vérifient les propriétés de réciprocité

P1. QQ n:Nat, f1 (f2 n) = n et

P2. QQ (l,c):Nat x Nat, f2 (f1 (l,c)) = (l,c)

on dit alors que les fonctions f1 et f2 sont des bijections réciproques.

2.3.4 Implantation des fonctions de Cantor en caml

let rec (f1: nat * nat -> nat) = function

(0,0) -> 0
(0,c) -> 1 + f1(c-1,0)
(l,c) -> 1 + f1(l-1,c+1)

let rec (f2: nat -> nat * nat) = function

0 -> (0,0)
n -> let (l,c) = f2 (n-1)

in ... à finir en TD ...

On peut démontrer dans un prouveur que ces fonctions vérifient bien les propriétés P1 et P2

3 Ensembles énumérables (= dénombrables)

On vient de montrer que

Nat <~ > Nat x Nat

Donc les couples d'entiers sont énumérables.

Si on note par une fraction "l sur c" les coordonnées de la case (l,c) on voit que les coordonnées des cases du tableau Nat* x Nat* correspondent aux rationnels Q*

On a donc Q* <~ > Nat* x Nat* et Q <~ > { 0 } U Nat* x Nat*

Donc les rationnels sont énumérables.

De même les triplets Nat x Nat x Nat sont dénombrables :

Preuve :

Nat x Nat x Nat <~ ~ ~ ~ ~ > Nat (x , y , z) –codeage–> n = f1(f1(x,y),z) let (a,z) = f2(n) <-decodeage- n in let (x,y) = f2(a) in (x,y,z)

De même en généralisant le codage précédent on démontre que

Nat ^n = Nat x Nat x ... x Nat (n fois) = vecteurs d'entiers de taille n sont dénombrables

En plaçant dans un tableau infini à la ligne k les vecteurs de taille k et en utilisant le parcours de Cantor pour numéroter les vecteurs, on démontre que

L'ensembles de tous les vecteurs de tailles finies (toutes les tailles de vecteurs mais pas de vecteur infini) est énumérable

Autrement dit, $\mathrm{U}_{\mathrm{n:Nat}}$ $\mathrm{Nat}^{\mathrm{n}}$ <~ ~> Nat

Exercice : Montrez que

- Z est énumérable
- Les vecteurs faits des 26 lettres de l'alphabet sont énumérables
- Les textes sont énumérables
- Les programmes sont énumérables

============ la prochaine fois =======

Ensembles non-dénombrable et limites de l'informatique :

L'ensemble R des réels n'est pas dénombrable L'ensemble des fonctions Nat –> Nat n'est pas dénombrable

R <~ > [0,1] <~ > (Nat —> {0..9})

Principe de preuve de non-énumérabilité : diagonalisation de Cantor

Remarques troublantes sur les ensembles infinis :

1. Q <~ > N et pourtant entre deux entiers il existe une infinité de rationnels

2. Q énumérable, R continu et pourtant Q dense dans R.