Bijections

Marc Lorenzi

14 février 2018

Résumé

Soit $f_2 : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ définie par $f_2(x,y) = \frac{1}{2}(x+y)(x+y+1)+y$. La fonction f_2 est une bijection. On se propose de le montrer, ainsi que d'écrire un algorithme qui calcule la réciproque de f_2. On utilise ensuite f_2 pour créer des bijections de $\mathbb{N}^p$ sur $\mathbb{N}$, ceci pour tout entier $p \geq 1$. On termine par une bijection explicite de l'ensemble $\mathbb{L}$ des suites finies d'entiers naturels sur $\mathbb{N}$.

Dans tout cet article, les entiers sont des entiers naturels.

I Une bijection de $\mathbb{N}\times\mathbb{N}$ sur $\mathbb{N}$

I.1 La fonction φ

Soit $\varphi : \mathbb{N} \to \mathbb{N}$ définie par $\varphi(n) = \frac{1}{2}n(n+1)$.

Propriété I.1 .

1. *Pour tout entier n, $\varphi(n+1) - \varphi(n) = n+1$.*
2. *La fonction φ est strictement croissante.*

Démonstration : Immédiate

Propriété I.2 *Soit $t \in \mathbb{N}$. Il existe un unique entier n tel que $\varphi(n) \leq t < \varphi(n+1)$.*

Démonstration : Commençons par l'unicité. Soient $n, n' \in \mathbb{N}$ tels que $\varphi(n) \leq t < \varphi(n+1)$ et $\varphi(n') \leq t < \varphi(n'+1)$. On a donc $\varphi(n) < \varphi(n'+1)$ et $\varphi(n') < \varphi(n+1)$. Comme φ est strictement croissante, on en déduit $n < n'+1$ et $n' < n+1$. Mais n et n' sont entiers, donc $n \leq n'$ et $n' \leq n$, d'où $n = n'$.

Montrons maintenant l'existence. On a, pour tout entier n, $\varphi(n) - n = \frac{1}{2}n(n+1) - n = \frac{1}{2}n(n-1) \geq 0$. Ainsi $\varphi(n) \geq n$. Soit $E = \{n \in \mathbb{N}, \varphi(n) > t\}$. On a $\varphi(t+1) \geq t+1 > t$ donc $t+1 \in E$ et $E \neq \emptyset$. E possède donc un plus petit élément m. Comme $\varphi(m) > t \geq 0$, on a $m > 0$. Soit donc $n = m-1$, qui est un entier naturel. Par minimalité de m, on a $\varphi(n) \leq t$. Et de plus $\varphi(n+1) = \varphi(m) > t$. D'où l'existence.

I.2 La fonction f_2

Soit $f_2 : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ définie par $f_2(x,y) = \varphi(x+y) + y$.

Propriété I.3 *La fonction f_2 est injective.*

Démonstration : Soient (x,y) et (x',y') deux éléments de $\mathbb{N}^2$. Supposons $f_2(x,y) = f_2(x',y')$. Posons $n = x+y$ et $n' = x'+y'$. On a donc $\varphi(n)+y = \varphi(n')+y'$. Supposons pour fixer les idées que $n' \geq n$. On a $\varphi(n') = \varphi(n) + y - y' \leq \varphi(n) + y \leq \varphi(n) + n < \varphi(n) + n + 1 = \varphi(n+1)$. Comme φ croît strictement, on en déduit que $n' < n+1$, c'est à dire $n' \leq n$. Ainsi, $n = n'$.

Propriété I.4 *La fonction f_2 est surjective.*

Démonstration : Soit $t \in \mathbb{N}$. Soit n l'unique entier naturel tel que $\varphi(n) \leq t < \varphi(n+1)$. Posons $y = t - \varphi(n)$. On a donc $t = \varphi(n) + y$ et ainsi $0 \leq y < \varphi(n+1) - \varphi(n) = n + 1$, ou encore $0 \leq y \leq n$. Posons $x = n - y$. On a alors $n = x + y$ et $x \geq 0$. Pour conclure, x et y sont des entiers naturels et $t = \varphi(x+y) + y = f_2(x, y)$.

On a donc montré le

Théorème I.1 *La fonction f_2 est une bijection.*

II Calcul de f_2^{-1}

II.1 Encadrements par φ

Soit $t \in \mathbb{N}$. Soit n l'unique entier naturel tel que $\varphi(n) \leq t < \varphi(n+1)$. On a vu que $0 \leq n < t + 1$. La recherche de n peut alors se faire par dichotomie et la complexité de la fonction est alors $O(\log t)$ en nombre de comparaisons et d'opérations sur des entiers. Nous allons montrer tout cela.

```
def phi(n):
  return n * (n + 1) // 2
```

```
1 def max_phi(t):
2   a = 0
3   b = t + 1
4   while b - a > 1:
5     c = (a + b) // 2
6     if phi(c) <= t: a = c
7     else: b = c
8   return a
```

Dans la suite nous allons discuter sur les valeurs de a et b à la fin de la $k$ième itération, où k est un entier naturel. Pour être parfaitement clairs :

- Les itérations commencent à l'itération numéro 1.
- Nous convenons que les valeurs de a et b à la fin de la zéroième itération sont les valeurs *initiales* de a et b, à savoir 0 et $t + 1$.
- L'entier k est inférieur ou égal au nombre total d'itérations, en supposant que la boucle `while` termine.

II.2 Étude de l'algorithme

Soit $t \in \mathbb{N}$. Soit n l'unique entier naturel tel que $\varphi(n) \leq t < \varphi(n+1)$.

Propriété II.1 *Après chaque itération on a $a \leq n < b$*

Démonstration : On fait une récurrence sur k, où k est le numéro de l'itération. Pour $k = 0$ c'est bien le cas : on a déjà vu que $0 \leq n < t + 1$. Soit maintenant k un entier. Supposons qu'à la fin de la $k$ième itération on a $a \leq n < b$. Soit $c = \lfloor \frac{a+b}{2} \rfloor$ (ou n'importe quoi d'autre, d'ailleurs). Si $\varphi(c) \leq t$, alors on pose à la ligne 6 $a = c$. On a donc $\varphi(a) \leq t < \varphi(n+1)$ donc, par croissance stricte de φ, $a < n + 1$ d'où $a \leq n$ (entiers). Et bien sûr, puisque b n'a pas changé, $n < b$. Si, au contraire, $\varphi(c) > t$, on pose à la ligne 7 $b = c$. Donc $\varphi(b) > t \leq \varphi(n)$ d'où $b > n$ par les mêmes arguments que plus haut.

Il peut paraître curieux que la valeur précise de c n'intervienne pas dans la démonstration de l'invariant de boucle ci-dessus. Que l'entier n soit encadré par a et b est une chose, mais encore faut-il que les encadrements s'améliorent au fil des boucles ... c'est là que la valeur de c joue un rôle.

Propriété II.2 *Soient a', b' les valeurs de a et b à la fin d'une itération. Soient a'', b'' leurs valeurs à la fin de l'itération suivante. On a $b'' - a'' \leq \lceil \frac{1}{2}(b' - a') \rceil$.*

Démonstration : Soit $c = \lfloor \frac{a'+b'}{2} \rfloor$. Il y a deux cas à considérer.

- Supposons $\varphi(c) \leq a'$. On a alors $a'' = c$ et $b'' = b'$, d'où $b'' - a'' = b' - \lfloor \frac{a'+b'}{2} \rfloor = \lceil \frac{1}{2}(b' - a') \rceil$.
- Supposons $\varphi(c) > a'$. On a alors $a'' = a'$ et $b'' = c$, d'où $b'' - a'' = \lfloor \frac{a'+b'}{2} \rfloor - a' = \lfloor \frac{1}{2}(b' - a') \rfloor \leq \lceil \frac{1}{2}(b' - a') \rceil$.

Les égalités entre plafonds et planchers sont laissées en exercice.

Propriété II.3 *Soit $(\delta_k)_{k \geq 0}$ la suite d'entiers naturels définie par $\delta_0 \in \mathbb{N}^*$ et pour tout $k \in \mathbb{N}$, $\delta_{k+1} = \lceil \frac{1}{2}\delta_k \rceil$. On a pour tout entier k, $0 \leq \delta_k - 1 \leq \frac{1}{2^k}(\delta_0 - 1)$.*

Démonstration : On fait une récurrence sur k. Pour $k = 0$ c'est évident. Soit maintenant $k \in \mathbb{N}$. Supposons la double inégalité vraie pour k. On a $1 \leq \delta_{k+1}$ car $\frac{1}{2}\delta_k > 0$. De plus, $\delta_{k+1} = \lceil \frac{1}{2}\delta_k \rceil \leq \frac{1}{2}\delta_k + \frac{1}{2}$ car δ_k est un entier non nul. Soustrayant 1, il vient $\delta_{k+1} - 1 \leq \frac{1}{2}(\delta_k - 1)$ et on termine avec l'hypothèse de récurrence.

Corollaire II.1 *Si $\delta_0 = 1$, la suite (δ_k) est constante, égale à 1. Sinon, on a $\delta_k = 1$ pour tout entier $k > \lg(\delta_0 - 1)$ où $\lg$ désigne le logarithme en base 2.*

Démonstration : Pour un tel entier k, on a $0 \leq \delta_k - 1 < 1$ donc $\delta_k - 1 = 0$ puisque ce nombre est un entier.

Corollaire II.2 *L'appel* `max_phi(t)` *termine et renvoie $\varphi(t)$ en $O(\log t)$ opérations.*

Démonstration : Appelons, pour tout entier k, a_k et b_k les valeurs de a et b après la $k$ième itération. Posant $\delta_k = b_k - a_k$ on a donc $\delta_k = 1$ pour un entier $k \leq \lg(\delta_0 - 1) = \lg t$. On a alors $b_k - a_k = 1$. La boucle se termine. Enfin, comme $a_k \leq n < b_k = a_k + 1$, on en déduit que $a_k = n$, ce qui est la valeur renvoyée par la fonction. À chaque itération, on effectue $O(1)$ opérations. Comme il y a $O(\lg t)$ itérations, la fonction effectue bien $O(\lg t)$ opérations.

II.3 La fonction f_2 et sa réciproque

Il est maintenant immédiat de coder la réciproque de f_2, qui termine en temps logarithmique.

```
def f2(x, y):
  n = x + y
  return phi(n) + y

def recip_f2(t):
  n = max_phi(t)
  y = t - phi(n)
  x = n - y
  return (x, y)
```

III $\mathbb{N}^3$, $\mathbb{N}^4$, . . . , $\mathbb{L}$

III.1 Une bijection de $\mathbb{N}^3$ sur $\mathbb{N}$

Propriété III.1 *Soit $f_3 : \mathbb{N}^3 \to \mathbb{N}$ définie par $f_3(x, y, z) = f_2(x, f_2(y, z))$. La fonction f_3 est bijective.*

Démonstration : Soient (x, y, z) et (x', y', z') deux éléments de $\mathbb{N}^3$. Supposons $f_3(x, y, z) = f_3(x', y', z')$. On a donc $f_2(x, f_2(y, z)) = f_2(x', f_2(y', z'))$. Par l'injectivité de f_2, il vient $x = x'$ et $f_2(y, z) = f_2(y', z')$. Toujours grâce à l'injectivité de f_2, on obtient $y = y'$ et $z = z'$.

Soit maintenant $t \in \mathbb{N}$. Soit $(x,u) = f_2^{-1}(t)$. Soit $(y,z) = f_2^{-1}(u)$. On a $f_3(x,y,z) = f_2(x, f_2(y,z)) = f_2(x,u) = t$. La fonction f_3 est donc surjective, et on a un algorithme effectif pour le calcul de sa réciproque.

```
def f3(x, y, z):
  return f2(x, f2(y, z))

def recip_f3(t):
  x, u = recip_f2(t)
  y, z = recip_f2(u)
  return (x, y, z)
```

III.2 Diviser pour régner - Une bijection de $\mathbb{N}^p$ sur $\mathbb{N}$

Soit p un entier non nul. Nous allons construire une bijection de $\mathbb{N}^p$ sur $\mathbb{N}$. Une approche naïve consiste à poser $f_{p+1}(x_1, x_2, \ldots, x_{p+1}) = f_2(x_1, f_p(x_2, \ldots, x_{p+1}))$. On montre alors par récurrence sur p que l'on obtient une famille de bijections $f_p : \mathbb{N}^p \to \mathbb{N}$. Le calcul de f_p nécessite $O(p)$ opérations, celui de sa réciproque $O(p \log t)$ opérations.

Une approche du type « diviser pour régner » se révèle plus efficace. Pour $p = 1$, la bijection f_1 est évidente. Soient $p, q \geq 1$. Supposons trouvées une bijection $f_p : \mathbb{N}^p \to \mathbb{N}$ et une bijection $f_q : \mathbb{N}^q \to \mathbb{N}$. La fonction $\psi : \mathbb{N}^{p+q} \to \mathbb{N}^p \times \mathbb{N}^q$ définie par $\psi(x_1, \ldots, x_{p+q}) = ((x_1, \ldots, x_p), (x_{p+1}, \ldots, x_{p+q}))$ est clairement bijective. Considérons la fonction $f_{p+q} : \mathbb{N}^{p+q} \to \mathbb{N}$ définie comme suit : Pour $X \in \mathbb{N}^{p+q}$, soit $(Y,Z) = \psi(X)$. On pose alors $f_{p+q}(X) = f_2(f_p(Y), f_q(Z))$. L'application f_{p+q} est une bijection.

L'approche naïve décrite plus haut consiste à prendre $q = 1$. L'approche plus fine, comme nous allons le voir, consiste à prendre $q = p$ ou $p+1$. Ci-dessous le code de la fonction f_p, appelée g : la valeur de p peut être retrouvée à partir de l'objet s passé en paramètre, que nous choisissons comme étant une liste. On coupe la liste en deux moitiés à peu près égales, on applique g récursivement sur les deux « moitiés », puis on applique f_2 aux deux valeurs obtenues.

```
def g(s):
  p = len(s)
  if p == 1: return s[0]
  else:
    q = p // 2
    s1 = s[0:q]
    s2 = s[q:]
    return f2(g(s1), g(s2))
```

Propriété III.2 *L'appel* `g(s)` *termine en* $O(\log p)$ *opérations, où* p *est la longueur de la liste* s*. Pour un entier* $p \geq 1$ *fixé, la fonction* g *est une bijection de l'ensemble des listes d'entiers naturels de longueur* p *sur l'ensemble* $\mathbb{N}$*.*

Démonstration : Une récurrence forte sur p s'impose. Pour $p = 1$ c'est évident. Soit maintenant $p \geq 2$. Supposons la propriété vraie pour tous les entiers non nuls strictement inférieurs à p. Soit s une liste de longueur p. Avec les notations de l'algorithme, la longueur de s_1 est $q = \lfloor \frac{p}{2} \rfloor$ et celle de s_2 est $q' = p - \lfloor \frac{p}{2} \rfloor = \lceil \frac{p}{2} \rceil$. Comme $p \geq 2$, ces deux entiers sont strictement compris entre 0 et p. On peut donc appliquer l'hypothèse de récurrence : les deux appels récursifs terminent et renvoient $f_q(s_1)$ et $f'_q(s_2)$ où f_q et f'_q sont des bijections de $\mathbb{N}^q$ sur $\mathbb{N}$ et $\mathbb{N}^{q'}$ sur $\mathbb{N}$ respectivement. L'appel à f_2 se fait ensuite en $O(1)$.

Notons $C(p)$ le nombre d'opérations nécessaires au calcul de $g(s)$ lorsque s est de longueur p. On a clairement $C(1) = O(1)$ et, pour tout $p \geq 2$, $C(p) = C(\lfloor \frac{p}{2} \rfloor) + C(\lceil \frac{p}{2} \rceil) + O(1)$. On montre dans tous les bons cours d'algorithmique que cette récurrence se résout en $C(p) = O(\log p)$.

Le calcul de la réciproque de f_p est facile. Soit $t \in \mathbb{N}$. Si $p = 1$, c'est immédiat, il suffit de renvoyer la liste de longueur 1 $[t]$. Sinon, soient $q = \lfloor \frac{p}{2} \rfloor$ et $q' = \lceil \frac{p}{2} \rceil$. Soit $(U,V) = f_2^{-1}(t)$. Soient $(x_1, \ldots, x_q) = f_q^{-1}(U)$ et $(x_{q+1}, \ldots, x_p) = f_{q'}^{-1}(V)$. On a alors $f_p^{-1}(t) = (x_1, \ldots, x_p)$.

```
def recip_g(p, t):
  if p == 1: return t
  else:
    U, V = recip_f2(t)
    p1 = p // 2
    p2 = p - p1
    Y = recip_g(p1, U)
    Z = recip_g(p2, V)
    return Y + Z
```

Une estimation grossière de la complexité de cette fonction est $O(\log p \log t)$. Nous ne creuserons pas plus avant.

III.3 Une bijection de $\mathbb{L}^*$ sur $\mathbb{N}$

On peut pour terminer (enfin, presque) se débarrasser de la dépendance en p. Nous allons écrire une bijection de $\mathbb{L}^* = \cup_{p=1}^{\infty} \mathbb{N}^p$, ensemble des listes non vides d'entiers, sur l'ensemble $\mathbb{N}$.

Propriété III.3 *Soit $f : \mathbb{L}^* \to \mathbb{N}$ définie comme suit. Pour tout $x \in \mathbb{L}^*$, $f(x) = f_2(f_p(x), p-1)$ où p est l'unique entier non nul tel que $x \in \mathbb{N}^p$. L'application f est une bijection.*

Démonstration : Soient $x, x' \in \mathbb{L}^*$. Supposons que $f(x) = f(x')$. Avec des notations évidentes, $f_2(f_p(x), p-1) = f_2(f_{p'}(x'), p'-1)$. Comme f_2 est injective, il vient $p = p'$ et $f_p(x) = f_{p'}(x') = f_p(x')$. Par injectivité de f_p on en déduit $x = x'$. L'application f est donc injective.

Soit maintenant $t \in \mathbb{N}$. Soit $(y, p-1) = f_2^{-1}(t)$. Soit $x = f_p^{-1}(y)$. On a $f(x) = f_2(f_p(x), p-1) = f_2(y, p-1) = t$. La fonction f est donc surjective et calculer sa réciproque est facile avec tout ce qui précède.

```
def f(s):
  p = len(s)
  return f2(g(s), p - 1)

def recip_f(t):
  y, p = recip_f2(t)
  return recip_g(p + 1, y)
```

La complexité de f est en $O(\log p)$ où p est la longueur de la liste s. Une estimation grossière de la complexité de sa réciproque est $O(\log^2 t)$.

III.4 Une bijection de $\mathbb{L}$ sur $\mathbb{N}$

Nous y sommes. La liste vide n'est pas prise en compte par la bijection f mais une modification mineure permet de créer une bijection $F : \mathbb{L} \to \mathbb{N}$. Ce qui était, si vous ne l'avez pas deviné, le but de tout cela.

```
def F(s):
  if len(s) == 0: return 0
  else: return f(s) + 1

def recip_F(t):
  if t == 0: return []
  else: return recip_f(t - 1)
```