1 Hiérarchie de Grzegorczyk

Cette note présente la hiérarchie de Grzegorczyk, en commançant par sa définition, quelques propriétés et enfin un théorème qui établit une hiérarchie des fonctions primitives récursives.

1.1 Introduction

La hiérarchie de Grzegorczyk – du nom du logicien polonais Andrzej Grzegorczyk – est une hiérarchie de fonctions utilisée en théorie de la calculabilité. Toutes les fonctions de la hiérarchie de Grzegorczyk sont primitives récursives et toute fonction primitive récursive apparaît dans cette hiérarchie. Cette hiérarchie classe les fonctions selon leur croissance. Intuitivement, les fonctions d'un niveau croissent moins vite que toutes les fonctions des niveaux supérieurs.

1.2 Définition

Une famille de fonctions Tout d'abord on introduit un ensemble (infini) de fonctions notées E_i pour tout entier naturel $i \geqslant 0$. On pose $E_0 : (x, y) \mapsto x + y$ et $E_1 : x \mapsto x^2 + 2$. En d'autre terme, E_0 est la fonction d'addition et E_1 est une fonction unaire qui élève au carré son argument et ajoute 2.

Ensuite, pour tout entier $n \geqslant 2$, on définit récursivement

$$E_n : x \mapsto \begin{cases} 2 & \text{si } x = 0 \\ E_{n-1}(E_n(x-1)) & \text{sinon} \end{cases}$$

Quelques valeurs Voici des exemples de valeurs, calculées avec Python ou OCaml, pour vérifier que ces fonctions sont "fortement croissantes".

$n \setminus x$	1	2	3	4	5	6
1	3	6	11	18	27	38
2	6	38	1446	2090918	4371938082726	19113842599189892819591078
3	38	trop grand	trop grand	trop grand	trop grand	trop grand
4	trop grand	trop grand	trop grand	trop grand	trop grand	trop grand

TABLE 1 – Quelques valeurs de $E_n(x)$.

En OCaml, une exception `Stack overflow during evaluation (looping recursion?).` est déclenchée pour les valeurs trop grandes, et en Python le calcul prend juste trop de temps pour être raisonnable (plus d'une heure).

Notez qu'il y a une différence dans les résultats donnés par les deux implémentations présentées plus bas. Par exemple, $E_2(6) = 19113842599189892819591078$ calculé en

Python (qui utilise des entiers en précisions arbitraires), mais -926155691629764698 en OCaml (qui utilise des entiers sur 64 bits). Dès que le résultat est négatif, cela signifie que OCaml a calculé un entier plus grand que $2^{62} - 1$, et donc le calcul devient faux.

Code OCaml Voici un code OCaml qui implémente cette fonction (qui considère que $n > 0$) :

```
let rec base_grzegorczyk n x =
    if n = 1 then x*x + 2
    else begin
        if x = 0 then 2
        else base_grzegorczyk (n-1) (base_grzegorczyk n (x - 1))
    end
;;
(* val base_grzegorczyk : int -> int -> int = <fun> *)
```

Code Python Voici un code Python qui implémente cette fonction :

```
def base_grzegorczyk(n, x):
    if n == 1:
        return x*x + 2
    elif x == 0:
        return 2
    else:
        return base_grzegorczyk(n-1, base_grzegorczyk(n, x - 1))
```

Propriétés de croissance De même que lorsqu'on étudie la fonction d'Ackermann, on peut énoncer les propriétés de croissance suivantes [Gakwaya 1997].

Lemma 1. *Pour tout n et x, on a :*

1. $x + 1 \leqslant E_n(x)$,
2. $E_n(x) \leqslant E_{n+1}(x)$,
3. $E_n(x) \leqslant E_n(x + 1)$,
4. $\forall m \geqslant 0, (E_n(x))^m \leqslant E_{n+1}(x + m)$.

Hiérarchie On peut alors définit la hiérarchie de Grzegorczyk : $\mathcal{E}^n$ la n-ième classe (ou niveau) de la hiérarchie de Grzegorczyk est le plus petit ensemble qui contient :

— les fonctions E_k pour $k < n$,

— **la fonction nulle** $(x \mapsto 0)$,

— **la fonction successeur** $S : x \mapsto x + 1$,

— **les projections** $\pi_i^m(t_1, t_2, \ldots, t_m) = t_i$,

et qui est stable par les deux opérations suivantes :

— **composition de fonction**, si f, g_1, g_2, …, g_m sont des fonctions de $\mathcal{E}^n$, alors $f : \bar{u} \mapsto h(g_1(\bar{u}), g_2(\bar{u}), \ldots, g_m(\bar{u}))$ l'est aussi,

— **récursion bornée**, si g, h et j sont des fonctions de $\mathcal{E}^n$ et que f est telle que $\forall t, \forall \bar{u}, f(t, \bar{u}) \leqslant j(t, \bar{u})$, $f(0, \bar{u}) = g(\bar{u})$ et $f(t+1, \bar{u}) = h(t, \bar{u}, f(t, \bar{u}))$, alors f est aussi une fonction de $\mathcal{E}^n$.

Démonstration. $\mathcal{E}^n$ existe et est bien défini, comme d'habitude lorsque qu'un ensemble de fonctions est construits comme le plus petit ensemble contenant un nombre fini (ou dénombrable) de fonctions de bases et stables par des constructions récursives. On a l'habitude de ce genre de construction. □

1.3 Propriétés

On commence par donner des exemples, puis un résultat contrôlant la croissance des fonctions dans les classes $\mathcal{E}_n$ successives. Ce résultat permet de justifier que les classes sont strictement inclues les unes dans les suivantes.

Exemples

— $\mathcal{E}^0$ contient des fonctions comme $x \mapsto x$, $x \mapsto x+1$, $x \mapsto x+2$ etc,

— $\mathcal{E}^1$ contient toutes les fonctions d'addition telles que $x \mapsto 4x$, $(x, y) \mapsto x + y$,

— $\mathcal{E}^2$ contient les fonctions de multiplication, telles que $(x, y) \mapsto xy$, $x \mapsto x^4$,

— $\mathcal{E}^3$ contient les fonctions exponentiation, comme $(x, y) \mapsto x^y$, $x \mapsto 2^{2^{2^x}}$. Cet ensemble est égal à celui des fonctions élémentaires [1],

— $\mathcal{E}^4$ contient la *tétration*. C'est l'opération ${}^b a = \underbrace{a^{a^{\cdot^{\cdot^a}}}}_{b \text{ copies de } a}$ Chaque opération est définie par itération à partir de l'opération a précédente. L'addition $(a+b)$ peut être définie comme b itérations de l'opération "ajouter 1" appliquée à a ; la multiplication $(a.b)$, comme b itérations de l'opération "ajouter a" appliquée à 0, et l'exponentiation (a^b) comme b itérations de l'opération "multiplier par a" appliquée à 1. De manière analogue, la tétration $({}^b a)$ peut être considérée comme b itérations de l'opération d'exponentiation de base a ("élever a à la puissance") appliquée à 1.

Inclusions successives La seule différence entre les définitions des classes $\mathcal{E}^n$ tient dans les fonctions E_k. Et donc il est immédiat de vérifier que l'on a $\mathcal{E}^0 \subseteq \mathcal{E}^1 \subseteq \mathcal{E}^2 \subseteq \cdots$, puisque $\{E_k \mid k < 0\} \subseteq \{E_k \mid k < 1\} \subseteq \{E_k \mid k < 2\} \subseteq \cdots$.

1. Cf. `https://complexityzoo.uwaterloo.ca/Complexity_Zoo:E#elementary`.

Croissances contrôlées [Gakwaya 1997] donne aussi le lemme de croissances contrôlées suivant.

Lemma 2. — *Les fonctions dans $\mathcal{E}^0$ sont bornées par les fonctions à faibles croissances :*

$$\forall f \in \mathcal{E}^0, \exists i, c_f \in \mathbb{N}, \forall \overline{x}, f(\overline{x}) \leqslant x_i + c_f,$$

— *Les fonctions dans $\mathcal{E}^1$ sont bornées par les fonctions linéaires :*

$$\forall f \in \mathcal{E}^0, \exists c_0, \dots, c_k \in \mathbb{N}, \forall \overline{x}, f(\overline{x}) \leqslant c_0 + c_1 x_1 + \cdots + c_k x_k,$$

— *Pour $n \geqslant 2$, les fonctions dans $\mathcal{E}^n$ sont bornées par les itérations de la fonction de contrôle E_{n-1} :*

$$\forall n \geqslant 2, \forall f \in \mathcal{E}^0, \exists m_f \in \mathbb{N}, \forall \overline{x}, f(\overline{x}) \leqslant (E_{n-1}(\max \overline{x}))^{m_f}.$$

Démonstration. Cela se fait presque comme la preuve des résultats de croissances sur les fonctions primitives récursives quand on prouve que la fonction d'Ackermann n'est pas primitive récursive. Les deux premiers points sont faciles, et le dernier vient de la définition de la classe $\mathcal{E}^n$. □

Inclusions strictes En fait, l'inclusion est stricte [Rose 1984 ; Gakwaya 1997].

$$\mathcal{E}^0 \subsetneq \mathcal{E}^1 \subsetneq \mathcal{E}^2 \subsetneq \cdots$$

Démonstration. Une conséquence du lemme de croissances contrôlées est que chaque classe $\mathcal{E}^n$ contient des fonctions dont la rapidité de croissance augmente avec l'indice n. Avec un argument diagonal, on montre que la classe $\mathcal{E}^n$ est différente de $\mathcal{E}^{n+1}$.

En effet, si elle sont égales, alors comme $f = E_n$ est dans $\mathcal{E}^{n+1}$, on obtient qu'il existe un $m_f \in \mathbb{N}$ constant (avec le troisième point du lemme 2) tel que $E_n(x) \leqslant (E_{n-1}(x))^{m_f}$. Mais par définition de E_n, on a aussi que $E_n(x) = E_{n-1}(E_n(x-1))$, donc cela donne $E_{n-1}(E_n(x-1)) \leqslant (E_{n-1}(x))^{m_f}$. Or le dernier point du lemme 1 donne que $(E_{n-1}(x))^{m_f} \leqslant E_n(x + m_f)$. Donc cela donne que $E_{n-1}(E_n(x-1)) \leqslant E_n(x + m_f)$.

Il faut finir, essayez en exercice ! □

Une autre explication vient du fait que que l'hyperopération H_n appartient à $\mathcal{E}^n$ mais pas à $\mathcal{E}^{n-1}$. L'hyperopération de degré m est notée $a \uparrow^m b$, pour $m \geqslant 0$, et est définie

$$a \uparrow^m b = \underbrace{a \uparrow^{m-1} \left(a \uparrow^{m-1} \left[a \uparrow^{m-1} \left(\dots \left[a \uparrow^{m-1} \left(a \uparrow^{m-1} a\right)\right] \dots\right)\right]\right)}_{b \text{ copies de } a}.$$

1.4 Hiérarchie des fonctions primitives récursives

La définition de $\mathcal{E}^n$ est la même que celle des fonctions primitives récursives, PR, sauf que la construction récursive est bornée $f(t, \bar{u}) \leqslant j(t, \bar{u})$ pour une certaine fonction $j \in \mathcal{E}^n$ et les fonctions $(E_k)_{k<n}$ sont clairement comprises dans $\mathcal{E}^n$. Par conséquent, la hiérarchie de Grzegorczyk peut être vue comme une façon de limiter la puissance de la récursion primitive.

Il est clair que les fonctions de chaque niveau sont primitives récursives (i.e., $\mathcal{E}^n \subseteq PR$) et par conséquent

$$\bigcup_n \mathcal{E}^n \subseteq PR.$$

On peut aussi montrer que toute fonction primitive récursive est présente dans la hiérarchie de Grzegorczyk [Rose 1984 ; Gakwaya 1997] soit

$$\bigcup_n \mathcal{E}^n = PR.$$

et les ensembles $\mathcal{E}^0, \mathcal{E}^1 \backslash \mathcal{E}^0, \mathcal{E}^2 \backslash \mathcal{E}^1, \ldots, \mathcal{E}^n \backslash \mathcal{E}^{n-1}, \ldots$ forment une partition de l'ensemble des fonctions primitives récursives PR.

1.5 Références

— `https://fr.wikipedia.org/wiki/Hiérarchie_de_Grzegorczyk`

— Grzegorczyk, A (1953). *Some classes of recursive functions*. Rozprawy matematyczne. 4 : 1–45.
L'article original est assez dur à suivre.

— Brainerd, W.S., Landweber, L.H. (1974), *Theory of Computation*, Wiley.

— Rose, H.E., *Subrecursion : Functions and hierarchies*, Oxford University Press, New York, USA, 1984.

— Gakwaya, J.–S. (1997), *A survey on the Grzegorczyk Hierarchy and its Extension through the BSS Model of Computability*, DOI 10.1.1.69.4621
Cet article plus récent est plus facilement compréhensible.
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.4621&rep=rep1&type=pdf`