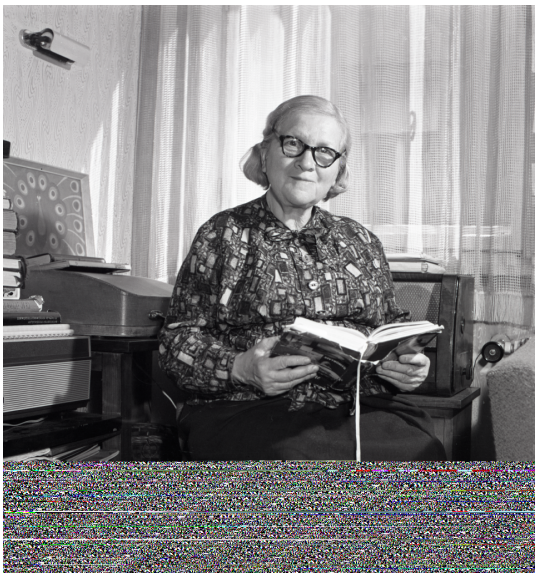
École Normale Supérieure de Rennes
Université de Rennes 1

Mathieu MARI

Fonctions primitives récursives et non primitives.

Encadrant : Yannick ZAKOWSKI
IRISA



Rózsa Péter (1905-1977), mathématicienne hongroise, figure éminante de la théorie des fonctions primitives récursives. Elle a notament introduit la version de la fonction d'Ackermann à deux variables.

Résumé.

Les fonctions récursives définissent un cadre formel à la notion intuitive de fonction calculable. A travers leur contruction en deux temps, nous verrons comment ce modèle rejoint les autres modèles de calcul, et comment tous ensembles, ils englobent la notion de calculabilité.

Année 2015-2016

3.3 Ensembles récursifs et récursivement énumérables

Définition. *Etant donné un ensemble $A \subset \mathbb{N}^k$, sa fonction caractéristique f_A est définie par*

$$f_A(\bar{m}) = \begin{cases} 1 & \text{si } \bar{m} \in A \\ 0 & \text{sinon} \end{cases}$$

Définition (Ensemble récursif)**.** *Un ensemble est* récursif *si sa fonction caractéristique est une fonction récursive totale.*

Exemples. Les ensembles suivants sont récursifs :
- Les ensembles finis.
- L'ensemble des multiples d'un entier k donné.
- L'ensemble des nombres premiers.

Définition. *Un ensemble est* récursivement énumérable *s'il est égal à l'ensemble de définition d'une fonction partielle récursive.*

Remarque. De façon équivalente, un ensemble peut être vu comme l'image d'une fonction partielle récursive.

Les ensembles récursifs (*resp.* récursivement énumérable correspondent aux langages décidés (*resp.* acceptés) par des machines de Turing.

Proposition. *Tout ensemble récursif est récursivement énumérable.*

Preuve. Soit $A \subset \mathbb{N}^k$ un ensemble récursif et soit f_A sa fonction caractéristique. On pose $g : \mathbb{N}^{k+1} \to \mathbb{N}$ tel que pour tout $\bar{m} \in \mathbb{N}^k$ et pour tout entier i,

$$g(\bar{m}, i) = \begin{cases} 0 & \text{si } \bar{m} \in A \\ 1 & \text{sinon} \end{cases}$$

On pose alors $h = \texttt{Min}(g)$. Comme g est récursive, h est une fonction partielle récursive donc l'ensemble de définition est exactement A. Donc A est récursivement énumérable. □

La réciproque de ce résultat est fausse mais le théorème suivant établit une forme de réciproque partielle.

Théorème 3. *Un ensemble $A \in \mathbb{N}^k$ est récursif si est seulement si A et $\mathbb{N}^k \setminus A$ sont récursivement énumérables.*

4 Equivalence avec les autres modèles de calcul.

On va voir dans cette partie d'autres modèles de calcul, et comparer leur pouvoir d'expression à celui des fonctions primitives.

4.1 Machines de Turing

4.1.1 Rappel du modèle.

On rappelle (très) rapidement les notions importantes et les notations utilisées concernant les machines de Turing. On pourra trouver un descriptif plus précis et plus complet dans par exemple [2]

Une machine de Turing M est un septuplet $(Q, \Sigma, \Gamma, E, q_0, F, \#)$ où
- Q est un ensemble fini d'*états*.
- Σ et Γ sont des ensembles finis, respectivement *alphabet d'entrée* et *alphabet de bande*.
- E est un ensemble fini de *transitions* de la forme $p, a \to q, b, x$ où p et q sont des états, a et b des symboles de bande, et x un élément de $\{\triangleleft, \triangleright\}$.
- $q_0 \in Q$ est l'*état initial*.
- $F \subset Q$ est l'ensembles des *états finaux* ou *états acceptants* .

— # est le symbole blanc.

Une *configuration* représente l'état de la machine à un instant donné. Les informations que contient une configuration sont :

— Le mot inscrit sur la bande.

— L'état courant.

— La position du pointeur.

Un *calcul* est une suite finie de configurations successives, où chaque configuration est obtenue à partir de la précédente en utilisant une transition de E.

Un calcul est *acceptant* si l'état dans lequel se trouve la machine à la fin du calcul est acceptant.

Un mot de Σ^* est *accepté* par M si il existe au moins un calcul acceptant avec w initialement inscrit sur le ruban.

Le *langage accepté* par M est l'ensemble des mots acceptés par M. On le note $L(M)$.

Un langage L est *décidé* par M si $L = L(M)$ et s'il n'existe pas de calcul infini sur M.

Une fonction $f : \Sigma^* \to \Gamma^*$ est *calculable* par machine de Turing (ou MT-calculable) si il existe une machine de Turing déterministe telle pour tout mot w initialement inscrit sur le ruban, il existe un calcul acceptant s'arretant sur $f(w)$.

Remarque. On peut également définir la notion de calculabilité sur des machine de Turing non-déterministes mais dans ce cas il faut garantir que le mot inscrit sur le ruban à l'issue de tout calcul acceptant soit le même. Comme toute machine de Turing est équivalente à une machine de Turing déterministe, il n'est pas restrictif de supposer que la machine de Turing utilisée pour calculer une fonction est déterministe.

4.1.2 Développement 2 : Equivalence entre les deux modèles.

Ce développement est tiré du Carton[2].

Théorème 4. *Une fonction est récursive si et seulement si elle est calculable par machine de Turing.*

Démonstration. Dans ce développement, on montre seulement un sens : une fonction calculable par une machine de Turing est récursive. On va cependant donner les idées de la démonstration du sens réciproque. Pour plus de détails voir [3].

Il est facile de concevoir des machines de Turing reconnaissant la fonctions nulle, la fonction successeur ainsi que les fonctions projections et identités.

Si une fonction est définie par composition de deux fonctions récursives g et f, on calcule avec une première machine $f(w)$, puis on recopie le résultat sur le ruban de la machine calculant g.

Pour une fonction h définie par récurrence de base f et de pas g, on peut construire une machine de Turing à trois rubans calculant h.

Si maintenant f est définie par minimisation de g, on peut également, à partir de celle de g, concevoir une machine à trois rubans calculant f.

On démontre maintenant que toute fonction calculable par une machine de Turing M est récursive. Comme toute machine de Turing est équivalente à une machine de Turing déterministe, à un seul état acceptant, qui ne bloque jamais, on peut supposer que M possède :

- Un alphabet dont les lettres sont les entiers de 0 à $m-1$ et que 0 représente le symbole blanc.
- Des états indéxés par les entiers de 0 à $n-1$, un état initial q_i et un unique état acceptant q_+.
- Une fonction de transition totale $\delta : \{0, \dots, m-1\} \times \{0, \dots, n-1\} \times \{\triangleleft, \triangleright\}$, où $\triangleleft$ (*resp.* $\triangleright$) symbolise le déplacement de la tête de lecture vers la gauche (*resp.* vers la droite).

Pour passer du concept de fonction MT-calculable à celui de fonction récursive, il faut d'abord pouvoir représenter un mot du ruban par un entier. On se munit donc de deux codages, correspondant à l'écriture en base m avec les unités à droite pour α et à gauche pour β.

$$\alpha(w) = \sum_{i=0}^{k} a_i m^{k-i} \qquad \text{et} \qquad \beta(w) = \sum_{i=0}^{k} a_i m^i$$

Soit w une entrée sur M et $C_0 \to \cdots \to C_n$ le calcul de M sur w. On suppose que chaque configuration C_k est égale à (u_k, q_k, v_k) où u_k est le mot sur le ruban strictement à gauche du pointeur, v_k le mot à droite du pointeur et q_k l'état courant.

On va montrer que le fonction c est primitive récursive où $c : \mathbb{N}^2 \to \mathbb{N}^3$ est définie par

$$c(\beta(w), k) = (\alpha(u_k), q_k, \beta(v_k))$$

Pour cela, on complète la fonction δ sur $\mathbb{N}^2$ en posant $\delta(i, j) = (0, 0, 0)$ si $i \geq m$ ou $j \geq n$. δ est bien sur une fonction primitive récursive puisqu'elle est « à support fini ». On définit alors la fonction c par récurrence. Dans la définition suivante, les fonctions `div` et `mod` désignent respectivement la division euclidienne et le reste de la division euclidienne qui sont toute deux primitives récursives.

- $c(\beta(w), 0) = (0, q_i, \beta(w))$
- On pose $a_k = mod(\beta(v_k), m)$ la lettre écrite dans la case pointée. On calcule la transition à effectuer :
$$(a_{k+1}, q_{k+1}, d_{k+1}) := \delta(a_k, q_k)$$
Deux situations se présentent alors,
— Si $d_{k+1} = \triangleright$, on pose
$$\alpha(u_{k+1}) = m\alpha(u_k) + a_{k+1}$$
$$\beta(v_{k+1}) = \texttt{div}(\beta(v_k), m)$$
— Si $d_{k+1} = \triangleleft$, on pose
$$\alpha(u_{k+1}) = \texttt{div}(\alpha(u_k), m)$$
$$\beta(v_{k+1}) = m(\beta(v_k) - a_k + a_{k+1}) + \texttt{mod}(u_k, m)$$

On définit bien là une fonction primitive récursive. Il suffit alors d'utiliser le schéma de minimisation afin de trouver le plus petit entier n tel que $q_n = q_+$. On renvoit alors $\beta(u_n v_n)$. Pour ce faire, on « retourne » u_n à partir de $\alpha(u_n)$ pour obtenir $\beta(u_n)$. On peut effectuer cette opération en extrayant par récurrence un à un chaque chiffre grâce à la fonction `mod`. On concatène ensuite u_k et v_k en effectuant l'opération :

$$\beta(u_k v_k) = \beta(u_k) + m^r \beta(v_k)$$

où r désigne le nombre de lettres du mot u_k.

□

Remarques. Cette démonstration nous conduit à remarquer plusieurs faits.

- Toute fonction récursive est égale à une fonction récursive n'utilisant qu'un seul schéma de minimisation.
- Si on connait une borne sur le nombre de calcul à effectuer pour arriver sur un état acceptant, il suffit d'utiliser la minimisation bornée (primitive récursive) pour minimiser la fonction c. Cette démonstration prouve donc également que les fonctions calculées par des machines de Turing linéairement bornée sont exactement les fonctions primitives récursives (il faudrait tout de même un peu préciser pourquoi une fonction primitive récursive peut être calculée par une machine de Turing linérairement bornée, mais cela n'est pas difficile).
Mais ce n'est pas tout. Si on sait borner le temps de calcul de M sur une entrée par une fonction primitive récursive prenant comme paramètre la taille de l'entrée, on peut encore utiliser la minimisation bornée, et alors la fonction calculée par M est primitive récursive. En particulier on peut énoncer deux corolaires intéressants.
  - Toute fonction totale de la classe EXPSPACE est primitive récursive (mais donc également toute fonction de la classe NEXPSPACE en vertu du théorème de Savitch).
  - Toute fonction totale de la classe EXPTIME est primitive récursive (c'est une conséquence du point précédent).

4.2 Opérateurs des langages de programmation

On peut imaginer les langages de programmation et les algorithmes en général comme des méthodes de calcul effectif au même titre que les fonctions primitives récursives ou les machines de Turing.

Certains langages de programmations ne sont pas aussi expressifs que d'autres, c'est pourquoi nous ne considérons que les langages Turing-complets comme C, Java, OCaml, *etc.* Il existe des langages de programmation qui ne sont pas Turing-complet comme par exemple Coq. Un avantage est d'avoir une garantie que le programme va terminer. On peut identifier cette remarque avec le fait que les fonctions primitives récursives sont totales. On peut donc se demander quelles fonctions sont calculables par un langage de programmation donné, et dans ce but on va expliciter le lien avec les fonctions récursives.

Pour simplifier, disons que tout programme ou algorithme s'articule autour de plusieurs opérateurs, *If-Then-Esle*, la boucle *For*, la boucle *While*, et la récursion. Nous allons montrer comment écrire ces opérateurs dans le langage des fonctions récursives, et nous verrons que les deux premiers s'expriment simplement dans le langages des fonctions primitives récursives.

4.2.1 If-Then-Else et boucle For

Soient un prédicat $P : \mathbb{N}^k \to \mathbb{N}$, et $f, g : \mathbb{N}^k \to \mathbb{N}^r$ deux fonctions. On note $\texttt{IfThenElse}[P, f, g]$ la fonction telle que pour tout $\bar{m} \in \mathbb{N}^k$,

$$\texttt{IfThenElse}[P, f, g](\bar{m}) = \begin{cases} f(\bar{m}) & \text{si } P(\bar{m}) = 1 \\ g(\bar{m}) & \text{sinon} \end{cases}$$

Cet opérateur évalué en $\bar{m}$ est égal à $P(\bar{m}) \cdot f(\bar{m}) + (1 - P(\bar{m})) \cdot g(\bar{m})$. On peut donc l'exprimer dans le langage des fonctions primitives récursives, à condition que P soit bien un prédicat primitif récursif. Or, les tests effectués s'expriment généralement à l'aide de $\leq$, $=$ et $\in$, qui sont primitifs récursifs. Par exemple, pour tester l'égalité entre deux tableaux, on pourra comparer successivement chaque élément de la liste.

L'opérateur boucle *For* des langages de programmation correspond en fait exactement au schéma de récurrence. Pour executer une boucle *For*, on se donne simplement une fonction f à executer dans le corps de boucle ainsi qu'un entier n, et on effectue consécutivement $f(i)$ pour i variant de 0 à n. La boucle *for* peut donc être simplement implémentée par un schéma de récurrence.

On peut donc donner le théorème informel suivant.

Théorème informel. *Les algorithmes ou programmes informatiques n'utilisant que des boucles* For *et des tests* If-Then-Else *calculent des fonctions qui sont primitives récursives.*

4.2.2 La boucle While et la récursion.

C'est avec l'utilisation à juste titre d'une boucle *While* que nous quittons le monde des fonctions primitives récursives. En effet, cette boucle s'identifie exactement au schéma de minimisation. Dans les deux cas nous arrêtons la boucle dès que nous rencontrons une instance qui satisfait le prédicat. Dans les deux cas, on ne peut pas savoir à l'avance s'il va en exister une, et c'est pourquoi il peut y avoir des entrées sur lesquelles un programme ne termine pas. Cependant, dans un programme informatique, on essaie toujours de garantir la *terminaison* du programme, pour des raisons clairement pratiques.

Les programmes définis inductivement, peuvent également s'écrire dans les langages des fonctions récursives. En effet, on peut utiliser le schéma de minimisation pour savoir combien d'appels récursifs sont à effectuer, puis une fois cette borne connue, on peut peut calculer les résultats des différents appels en utilisant une boucle *For* en partant des cas de base de la récursion.

On peut finalement énoncer le théorème informel suivant.

Théorème informel. *Tout algorithme ou programme qui termine calcule une fonction qui est récusive totale.*

Ce « théorème » nous enseigne donc qu'a tout algorithme donné calculant une fonction correspond une fonction récursive et vice et versa. Ces deux modèles de calcul sont donc équivalent au sens où toute fonction calculable dans un des deux modèles l'est également dans l'autre.

4.3 Frontière de la calculabilité

4.3.1 Thèse de Church-Turing

Dans les parties précédentes, on a vu que le modèle des fonctions récursives était équivalent à celui des machines de Turing ainsi que celui des langages Turing-complets. Il est également équivalement à d'autres modèles de calcul comme celui du λ-calcul. Tous ces modèles sont équivalents dans leur pouvoir d'expression bien que les règles de construction qui les définissent soient parfois bien différentes. Tout cela nous conduit à penser que nous avons atteind la « borne supérieure » du calculable. Tous ces modèles formalisent correctement et complétement la notion intuitive de fonction calculable par une procédure effective. Ce résultat encore une fois informel est connu sous le nom de thèse de Church-Turing.

Thèse de Church-Turing. *Les règles formelles de calcul des fonctions récursives formalisent correctement la notion de méthode effective de calcul.*

Cette thèse a été initialement formulée pour le λ-calcul, mais s'applique aussi bien aux fonctions récursives qu'aux machines de Turing.

4.3.2 Fonctions non récursives.

L'ensemble des fonctions récursives étant dénombrable, il existe des fonctions de $\mathbb{N}$ dans $\mathbb{N}$ qui ne sont pas récursives. Il n'est donc pas possible de donner de procédure mécanique pour les calculer et leur définition même ne s'exprime pas simplement. Nous allons donner l'exemple du *castor affairé*, une fonction non calculable qui traduit des propriétés d'une certaine classe de machines de Turing.

4.3.3 Le castor affairé

Pour un entier n donné, on regarde l'ensemble des machines de Turing à $n+1$ états, avec un seul état final. Chacune de ces machines a un ruban infini à gauche et à droite et son alphabet est $\{0,1\}$ où 0 est le symbole blanc. Initialement, aucun 1 n'est inscrit sur le ruban. On lance un calcul sur une telle machine et si la machine s'arrête en temps fini, on note le nombre de 1 inscrits sur le ruban. Le *score* d'une telle machine est nombre maximum de 1 que l'on peut obtenir à l'issue d'un calcul acceptant.

Définition. *La fonction du* castor affairé $\Sigma : \mathbb{N} \to \mathbb{N}$ *est telle que* $\Sigma(n)$ *est le score maximal que peut obtenir une machine de Turing à* $n+1$ *états répondant aux spécifications précédentes.*

Proposition. *La fonction du castor affairé n'est pas récursive.*

La démonstration de son incalculabilité repose sur le fait que pour toute fonction récursive $f : \mathbb{N} \to \mathbb{N}$, $\Sigma(n) > f(n)$ pour n suffisament grand.

Références

[1] J.-M. Autebert. *Calculabilité et Décidabilité.* Masson, 1992.

[2] O. Carton. *Langages formels, calculabilité et compléxité.* Vuibert, 2008.

[3] P. Dehornoy. *Mathématiques de l'informatique.* Dunob, 2000.

[4] P. Wolper. *Introduction à la calculabilité.* Dunob, 2006.