Machine de Turing $\Longrightarrow$ μ-recursif

Leçons : 912 ; 913

Références : WOLPER, *Introduction à la calculabilité* (p.135) [?] et CARTON, *Langages formels, calculabilité et complexité* (p.185) [?] et LESESVRE–MONTAGNON–LE BARBENCHON–PIERRON, *131 développements pour l'oral* (p. 687) [?]

Prérequis :

- Savoir coder les fonctions de base en μ-récursif (voir appendice sur les fonctions μ-récursives)

Introduction :

On va montrer que toute fonction calculable par une machine de Turing est une fonction μ-récursive, c'est un étape pour valider la thèse de Church qui nous dit que toute fonction calculable par une procédure effective est en fait une fonction calculable par une machine de Turing. En effet, on va pouvoir remplacer machine de Turing par λ-calcul ou fonctions μ-récursives quand on aura prouver que ces trois modèles de calcul sont équivalents. On prouve ici une implication de cette équivalence.

Théorème 1. Tout fonction (totale) calculable par une machine de Turing est calculable par une fonction (totale) μ-récursive.

Démonstration. Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, F, \#)$ une machine de Turing déterministe à ruban bi-infini qui boucle sur ces états finals [1] telle que si n est écrit en unaire sur son ruban, à la fin de l'exécution, il y a écrit $f(n)$ en unaire.

On représente :

— l'ensemble des états Q par des entiers $\{0, \dots, m\}$ où l'état initial $q_0 = 0$

— l'alphabet de lecture $\Sigma = \{1\}$ [2]

— l'alphabet d'écriture $\Gamma = \{0, \dots, k-1\}$ où $k = |\Gamma|$ et le symbole blanc $\#$ est représenté par 0.

On représente la configuration

							$\downarrow$						
$\dots$	$\#$	$\#$	g_j	$\dots$	g_1	g_0	d_0	d_1	$\dots$	d_i	$\#$	$\#$	$\dots$

par le triplet d'entiers (g, q, d) où $q \in \{0, \dots, m\}$ et g, d sont les entiers en base k des mots avant et après la tête de lecture de M. [3]

$$g = \overline{g_0 \dots g_j}^k = \sum_{p=0}^{j} g_p k^p \quad \text{et} \quad d = \overline{d_0 \dots d_i}^k = \sum_{p=0}^{i} d_p k^p$$

La fonction $\mathbb{1}_F$ est primitive récursive car c'est une somme finie d'indicatrice.

La fonction δ est primitive récursive car elle est aussi à support fini.

1. pour permettre à la fonction config_suivante d'être totale, car toutes les fonctions primitives récursives sont totales
2. car n est écrit en unaire
3. avec bit de poids faible au niveau de la tête de lecture

On va maintenant décrire le fonctionnement de la machine de Turing par des fonctions primitives récursives et μ-récursives et donc écrire la fonction f sous la forme d'une fonction μ-récursive.

$$\mathsf{init} : \begin{cases} \mathbb{N} & \to & \mathbb{N}^3 \\ n & \mapsto & \left(0, 0, \sum_{p=0}^{n-1} k^p\right) \end{cases}$$

La fonction init est primitive récursive car la somme et l'exponentiation sont primitives récursives. [4]

$$\mathsf{config\_suivante} : \begin{cases} \mathbb{N}^3 & \to & \mathbb{N}^3 \\ (g, q, d) & \mapsto & \begin{cases} (g * k + \alpha, q', d//k) & \text{si } dir = \to \\ (g//k, q', g_0 + k * (\alpha + (d//k) * k)) & \text{si } dir = \leftarrow \end{cases} \end{cases}$$

pour $\delta(q, d_0) = (q', \alpha, dir)$. [5]

La fonction config_suivante est primitive récursive car on utilise que des fonctions primitives récursives (mod, //, +,∗).

$$\mathsf{config} : \begin{cases} \mathbb{N}^3 \times \mathbb{N} & \to & \mathbb{N}^3 \\ (\vec{x}, k) & \mapsto & \begin{cases} \vec{x} & \text{si } k = 0 \\ \mathsf{config\_suivante}(\mathsf{config}(\vec{x}, k-1)) & \text{sinon} \end{cases} \end{cases}$$

La fonction config est primitive récursive en utilisant le principe de récursion.

$$\mathsf{temps\_arret} : \begin{cases} \mathbb{N}^3 & \to & \mathbb{N} \\ \vec{x} & \mapsto & \mu_i(\mathbb{1}_F(\pi_2^3(\mathsf{config}(\vec{x}, i)))) \end{cases}$$

La fonction temps_arret est μ-récursive car utilise une minimisation non bornée.

$$\mathsf{somme} : \begin{cases} \mathbb{N} & \to & \mathbb{N} \\ p & \mapsto & \mu(i \leqslant p)(k^i > p) \end{cases}$$

La fonction somme est primitive récursive, elle permet de retrouver l'entier en unaire qui correspond au nombre en base k.

Ainsi on pose la fonction

$$\begin{aligned} f(n) =& \mathsf{somme}(\pi_1^3(\mathsf{config}(\mathsf{init}(n), \mathsf{temps\_arret}(\mathsf{init}(n))))) \\ &+ \mathsf{somme}(\pi_3^3(\mathsf{config}(\mathsf{init}(n), \mathsf{temps\_arret}(\mathsf{init}(n)))))\end{aligned}$$

La fonction $\mathsf{config}(\mathsf{init}(n), \mathsf{temps\_arret}(\mathsf{init}(n)))$ renvoie une configuration, il faut ensuite récupérer le g et d correspondant d'où l'utilisation de projections.

On a donc réussi à simuler la machine de Turing avec une fonction μ-récursive.

□

4. Elle représente le ruban au début de l'exécution de la machine car la tête de lecture est sur la première case du mot (donc $g = 0$), dans l'état initial $q_0 = 0$ et il y a n fois le symbole 1 sur le ruban donc représenté par $\sum_{p=0}^{n-1} k^p$ en base k.

5. où $g_0 = g \bmod k$ et $d_0 = d \bmod k$

Remarques :

C'est en fait une équivalence de modèle de calcul et on prouve l'autre sens par induction, en simulant les fonctions de base des fonctions μ-récursives par une machine de Turing.

Le symbole # est représenté par 0 ce qui coïncide bien avec le fait que $\underbrace{\#\#\#\dots\#}_{r} g_j \dots g_0$ s'écrit $g = \sum_{l=1}^{j} g_l k^l + \sum_{l=1}^{r} 0 k^{i+l}$. Autrement dit, cela ne change rien d'ajouter des 0 à gauche (ou à droite si on pense à d) des nombres que l'on écrit en base k.

Astuces de l'agrégatif :

Cette preuve existe aussi avec des codages de Gödel mais elle est plus compliquée pour les notations, ici ce qui facilite c'est que l'on code tout en unaire pour l'entrée et sortie de la machine de Turing.

Questions possibles :

- Justifier que l'on peut considérer une machine de Turing qui vérifie les hypothèses de l'énoncé (déterminisme, ruban bi-infini, unaire, unique état initial et final, boucle sur l'état final).
- Montrer que toute fonction μ-récursive est calculable par une machine de Turing en explicitant les machines de Turing.
- Montrer que les fonctions arithmétiques utilisées dans ce développement sont primitives récursives.
- Montrer que la comparaison est un prédicat primitif récursif.
- Montrer que la conditionnelle est une fonction primitif récursif, *i.e.* si P est un prédicat primitif récursif et f, g deux fonctions primitives récursives, alors la fonction qui a n associe $\mathsf{if}\ P(n)\ \mathsf{then}\ f(n)\ \mathsf{else}\ g(n)$ est primitive récursive.
- Montrer que la minimisation bornée est une fonction primitive récursive.