Chapitre 2

Deux modèles calculatoires simples

Motivation

Tout problème peut-il être résolu par un programme qui a suffisamment de ressources ?

Avant de répondre à cette question, nous allons étudier, dans ce chapitre-ci et dans le chapitre suivant, la notion de calculabilité à travers cinq modèles de calcul :

- les programmes `RÉPÉTER`,
- les programmes `TANTQUE`,
- les machines de Turing,
- le λ-calcul
- les circuits booléens.

Les programmes RÉPÉTER

- Un nombre arbitrairement grand de registres est disponible : r_0, r_1 ... ;
- chaque registre contient un entier positif ou nul ;
- les registres sont implicitement initialisés à 0 avant utilisation ;
- Trois types d'instructions :
  - $r_i \leftarrow r_j$: remplace le contenu du registre r_i par celui de r_j ;
  - `inc`(r_i) : incrémente de 1 le registre r_i ;
  - **`répéter`** r_i **`fois`** $[\langle \text{Bloc} \rangle]$: répète l'éxécution d'un bloc d'instructions r_i fois ;

- le nombre d'éxécution de $\langle \textsc{Bloc} \rangle$ est fixé une fois pour toutes avant l'entrée dans la boucle, que r_i y soit modifié ou non.
- Un programme `RÉPÉTER` implante une fonction

$$\begin{array}{rcccl} f & : & \mathbb{N} \times \mathbb{N} \times \cdots \times \mathbb{N} & \to & \mathbb{N} \\ & & (r_1,\, r_2,\, \ldots,\, r_k) & \mapsto & r_0. \end{array}$$

  Au début de l'exécution, les registres r_1 à r_k contiennent les arguments de f, et à la fin, r_0 contient $f(r_1, \ldots, r_k)$.

Structure récursive pour les programmes RÉPÉTER

On peut voir les programmes répéter comme une structure récursive :

- $\mathcal{S}_0 = \{\texttt{inc}(r_i) \mid i \in \mathbb{N}\} \cup \{r_i \leftarrow r_j \mid i, j \in \mathbb{N}\}$
- R_1 : La concaténation de deux programmes répéter est un programme répéter
- R_2 : Si P est un programme répéter,

$$\texttt{répéter}\ r_i\ \texttt{fois}\ [\ P\ ]$$

en est un aussi pour tout $i \in \mathbb{N}$.

Addition

$$
\begin{aligned}
&\texttt{PLUS}(r_1, r_2) = r_1 + r_2 \\
&\qquad r_0 \leftarrow r_1 \\
&\qquad \texttt{répéter}\ r_2\ \texttt{fois}\ [\\
&\qquad\qquad \texttt{inc}(r_0) \\
&\qquad]
\end{aligned}
$$

Multiplication

$$\texttt{MULT}(r_1, r_2) = r_1 \cdot r_2$$

$$\texttt{répéter } r_1 \texttt{ fois } [$$

$$\texttt{répéter } r_2 \texttt{ fois } [$$

$$\texttt{inc}(r_0)$$

$$]$$

$$]$$

Exponentiation

$\texttt{EXP}(r_1, r_2) = r_1^{r_2}$

$\texttt{inc}(r_0)$

$\texttt{répéter}\ r_2\ \texttt{fois}\ [$

$r_3 \leftarrow r_4$ $\qquad (r_3 \leftarrow 0)$

$\texttt{répéter}\ r_0\ \texttt{fois}\ [$ $\qquad (r_3 \leftarrow r_0 \cdot r_1)$

$\texttt{répéter}\ r_1\ \texttt{fois}\ [$

$\texttt{inc}(r_3)$

$]$

$]$

$r_0 \leftarrow r_3$

$]$

Sucre syntaxique

- L'instruction

$$r_i \leftarrow \texttt{PROC}(r_{j_1},\ \ldots,\ r_{j_k})$$

signifie que l'on doit substituer à cette ligne un bloc d'instructions qui a pour effet de remplacer le contenu du registre r_i par la valeur calculée par $\texttt{PROC}(r_{j_1},\ \ldots,\ r_{j_k})$, en renommant au besoin les variables qui apparaissent dans le code de la procédure PROC.

Les appels récursifs ne sont pas permis.

- L'instruction

$$r_i \leftarrow k$$

signifie que l'on doit substituer à cette ligne k incrémentations, ce qui aura pour effet d'affecter la constante k au registre r_i.

Partout, on peut mettre une constante k au lieu d'utiliser une variable auxiliaire qu'on aurait incrémentée k fois.

$$\texttt{EXP}(r_1, r_2) = r_1^{r_2}$$

$r_0 \leftarrow 1$

répéter r_2 **fois** [

$r_0 \leftarrow \texttt{MULT}(r_0, r_1)$

]

Décrémentation

$$
\begin{aligned}
&\texttt{DEC}(r_1) = \max(0, r_1 - 1) \\
&\qquad \texttt{répéter}\ r_1\ \texttt{fois}\ [\\
&\qquad\qquad r_0 \leftarrow r_2 \\
&\qquad\qquad \texttt{inc}(r_2) \\
&\qquad]
\end{aligned}
$$

Soustraction

$$\texttt{MOINS}(r_1, r_2) = \max(0, r_1 - r_2)$$

$r_0 \leftarrow r_1$

$\texttt{répéter}\ r_2\ \texttt{fois}\ [$

$r_0 \leftarrow \texttt{DEC}(r_0)$

$]$

Factorielle

$$
\begin{aligned}
&\texttt{FACT}(r_1) = r_1! \\
&\qquad r_0 \leftarrow 1 \\
&\qquad \texttt{répéter}\ r_1\ \texttt{fois}\ [\\
&\qquad\qquad \texttt{inc}(r_2) \\
&\qquad\qquad r_0 \leftarrow \texttt{MULT}(r_0, r_2) \\
&\qquad]
\end{aligned}
$$

Sucre syntaxique pour les variables booléennes

Nous adoptons les conventions syntaxiques suivantes :

- `vrai` pour la constante 1,
- `faux` pour la constante 0.

Pour évaluer $\langle \textsc{Bloc} \rangle$ conditionnellement à la valeur booléenne r_i, on répète $\langle \textsc{Bloc} \rangle$ r_i fois.

L'instruction

$$\texttt{si}\ r_i\ \texttt{alors}\ [\langle \textsc{Bloc} \rangle]$$

sera mise pour

$$\texttt{répéter}\ r_i\ \texttt{fois}\ [\langle \textsc{Bloc} \rangle].$$

Et

$\texttt{ET}(r_1, r_2)$

$r_0 \leftarrow \texttt{MULT}(r_1, r_2)$

Négation

$\texttt{NEG}(r_1)$

$r_0 \leftarrow \texttt{MOINS}(1, r_1)$

Ou

On utilise l'identité de De Morgan :

$$x \vee y = \overline{\bar{x} \wedge \bar{y}}$$

$\texttt{OU}(r_1, r_2)$

$r_1 \leftarrow \texttt{NEG}(r_1)$

$r_2 \leftarrow \texttt{NEG}(r_2)$

$r_0 \leftarrow \texttt{ET}(r_1, r_2)$

$r_0 \leftarrow \texttt{NEG}(r_0)$

Plus grand que

$$
\begin{aligned}
&\texttt{PG?}(r_1, r_2) = (r_1 > r_2) \\
&\qquad r_3 \leftarrow \texttt{MOINS}(r_1, r_2) \\
&\qquad \texttt{répéter}\ r_3\ \texttt{fois}\ [\\
&\qquad\qquad r_0 \leftarrow \texttt{vrai} \\
&\qquad]
\end{aligned}
$$

Division

$$\texttt{DIV}(r_1, r_2) = \left\lfloor \frac{r_1}{r_2} \right\rfloor$$

répéter r_1 fois [

$r_3 \leftarrow \texttt{PLUS}(r_3, r_2)$

$r_4 \leftarrow \texttt{PG?}(r_3, r_1)$

$r_4 \leftarrow \texttt{NEG}(r_4)$

si r_4 alors [

$\texttt{inc}(r_0)$

]

]

Modulo

$$a = x \bmod y$$

$$b = x \operatorname{div} y$$

$$x = by + a$$

$$\texttt{MOD}(r_1, r_2) = r_1 \bmod r_2$$

$$r_0 \leftarrow \texttt{DIV}(r_1, r_2)$$

$$r_0 \leftarrow \texttt{MULT}(r_0, r_2)$$

$$r_0 \leftarrow \texttt{MOINS}(r_1, r_0)$$

Test de primalité

$\texttt{PREMIER?}(r_1) = (r_1 \in \mathbb{P})$

$r_0 \leftarrow \texttt{faux}$

$r_5 \leftarrow \texttt{PG?}(r_1, 1)$

$\texttt{si } r_5 \texttt{ alors } [$

$r_0 \leftarrow \texttt{vrai}$

$r_3 \leftarrow 1$

$r_2 \leftarrow \texttt{MOINS}(r_1, 2)$

$\texttt{répéter } r_2 \texttt{ fois } [$

$\texttt{inc}(r_3)$

$r_4 \leftarrow \texttt{MOD}(r_1, r_3)$

$r_5 \leftarrow \texttt{PG?}(1, r_4)$

$\texttt{si } r_5 \texttt{ alors } [\, r_0 \leftarrow \texttt{faux} \,]$

$]$

$]$

$r_5 \leftarrow (r_4 = 0)$

Prochain nombre premier

$\texttt{PREMIERSUIV}(r_1) =$ le plus petit nombre premier plus grand que r_1

$r_2 \leftarrow \texttt{FACT}(r_1)$
$\texttt{inc}(r_2)$
$r_3 \leftarrow \texttt{vrai}$
$\texttt{répéter}\ r_2\ \texttt{fois}\ [$
$\texttt{inc}(r_1)$
$r_4 \leftarrow \texttt{PREMIER?}(r_1)$
$r_4 \leftarrow \texttt{ET}(r_3, r_4)$
$\texttt{si}\ r_4\ \texttt{alors}\ [$
$r_0 \leftarrow r_1$
$r_3 \leftarrow \texttt{faux}$
$]$
$]$

k-ème nombre premier

$\texttt{PREMIERK}(r_1) =$ le r_1-ème nombre premier

$\texttt{inc}(r_0)$

$\texttt{répéter}\ r_1\ \texttt{fois}\ [$

$r_0 \leftarrow \texttt{PREMIERSUIV}(r_0)$

$]$

Structure de données : tableau

Nous allons implanter les tableaux d'entiers à l'aide du codage de Gödel. Soit p_k le k-ème nombre premier. Le tableau infini

$$(a_1, a_2, \dots, a_n, 0, 0, \dots), \text{ où } a_k \in \mathbb{N},$$

est représentée sans ambiguïté par l'entier

$$p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}.$$

Structure de données : codage de Gödel

La liste

$$(5, 2, 3, 17, 13, 0, 0, 0, ...)$$

est représenté par l'entier

$$2^5\, 3^2\, 5^3\, 7^{17}\, 11^{13} = 289117305729902212346760865812000.$$

Une autre variable peut accessoirement être utilisée pour indiquer la position du dernier élément.

Structure de données : codage de Gödel

Il est aussi possible d'interpréter un entier comme une liste de liste.

$$((1, 2, 3, 0, ...), (4, 5, 0, ...), (6, 7, 0, ...), 0, ...)$$

est représentée par l'entier

$$2^{(2^1\, 3^2\, 5^3)}\, 3^{(2^4\, 3^5)}\, 5^{(2^6\, 3^7)}$$

ayant 100366 décimales.

Les listes de ce type peuvent être utilisées pour encoder des structures de données complexes comme des piles, files, arbres ou graphes.

Extraction d'un élément d'un tableau

$\texttt{TABLVAL}(r_1, r_2) = r_2$-ème élément du tableau r_1

$r_3 \leftarrow \texttt{PREMIERK}(r_2)$

$r_4 \leftarrow r_3$

répéter r_1 **fois** [

$r_5 \leftarrow \texttt{MOD}(r_1, r_4)$

$r_5 \leftarrow \texttt{PG?}(1, r_5)$

si r_5 **alors** [$\qquad$ (**si** $\texttt{MOD}(r_1, r_4) = 0$ **alors**)

$\texttt{inc}(r_0)$

$r_4 \leftarrow \texttt{MULT}(r_3, r_4)$

]

]

Assignation d'un élément dans un tableau

$\texttt{TABLASS}(r_1, r_2, r_3) =$ tableau r_1 où le r_2-ème élément est remplacé par r_3

$$r_4 \leftarrow \texttt{TABLVAL}(r_1, r_2)$$
$$r_5 \leftarrow \texttt{PREMIERK}(r_2)$$
$$r_6 \leftarrow \texttt{EXP}(r_5, r_4)$$
$$r_0 \leftarrow \texttt{DIV}(r_1, r_6)$$
$$r_7 \leftarrow \texttt{EXP}(r_5, r_3)$$
$$r_0 \leftarrow \texttt{MULT}(r_0, r_7)$$

Puissance des programmes RÉPÉTER

Il semble que les programmes RÉPÉTER peuvent calculer des fonctions complexes.

Peut-on calculer toutes les fonctions à valeurs entières avec un programme RÉPÉTER ?

Remarque 2.1

Un programme RÉPÉTER ne peut pas entrer dans une boucle infinie : son exécution se termine toujours.

Définition 2.2

Les fonctions calculables par un programme RÉPÉTER sont appelées primitives récursives.

Pour une fonction $f : \mathbb{N} \to \mathbb{N}$ et un entier n, on note :

$$
\begin{aligned}
f^{\langle 0 \rangle}(x) &= x \\
f^{\langle 1 \rangle}(x) &= f(x) \\
f^{\langle 2 \rangle}(x) &= f(f(x)) \\
&\vdots \\
f^{\langle n \rangle}(x) &= \underbrace{f(f(\cdots x \cdots))}_{n \text{ fois}}
\end{aligned}
$$

$$
\begin{array}{l}
\texttt{répéter}\ n\ \texttt{fois}\ [\\
\qquad r_0 \leftarrow f(r_0) \\
]
\end{array}
$$

Boucles imbriquées

Définition 2.3

Pour tout $i \in \mathbb{N}$,

$$B_i \quad : \quad \mathbb{N} \to \mathbb{N}$$

$$x \mapsto B_i(x) = \begin{cases} x+1 & \text{si} \quad i = 0 \\ {B_{i-1}}^{\langle x+1 \rangle}(1) & \text{si} \quad i > 0 \end{cases}$$

Lemme 2.4

Pour tout $i \geq 0$, B_{i+1} est calculable par un programme RÉPÉTER avec profondeur de boucle i.

Démonstration :

Procédons par récurrence sur i. Cas de base : $i = 0$:

$$\begin{aligned} &\mathtt{B}_1(r_1) \\ &\qquad \mathtt{inc}(r_1) \\ &\qquad \mathtt{inc}(r_1) \\ &\qquad r_0 \leftarrow r_1 \end{aligned}$$

Pas de récurrence : supposons qu'il existe un programme répéter $\mathtt{B}_{i+1}$ de profondeur de boucle i qui calcule B_{i+1}. Alors, pour $i+1$, considérons le programme suivant :

$\mathtt{B}_{i+2}(r_1)$
    $\mathtt{inc}(r_0)$
    $\mathtt{inc}(r_1)$
    $\mathtt{répéter}\ r_1\ \mathtt{fois}\ [$
        $r_0 \leftarrow \mathtt{B}_i(r_0)$
    $]$

□

On remarque que le programme $\mathtt{B}_{i+2}$ compte exactement $i+1$ boucles répéter et la profondeur d'imbrication est aussi $i+1$.

Que calcule B_i ?

$$
\begin{aligned}
B_0(x) &= x+1 \\
B_1(x) &= B_0(B_0(\cdots 1 \cdots)) = 1 + (x+1) = x+2 \\
&= 2 + (x+3) - 3 \\
B_2(x) &= B_1(B_1(\cdots 1 \cdots)) = 1 + (x+1) \times 2 = 2x+3 \\
&= 2(x+3) - 3 \\
B_3(x) &= B_2(B_2(\cdots 1 \cdots)) \\
&= 2^{x+3} - 3 \\
&\vdots
\end{aligned}
$$

Cette fonction croît à une vitesse hallucinante...

Hyperopérations : flèche de Knuth

Pour aller plus loin, il faut une notation permettant de généraliser les opérations répétées :

Définition 2.5 (Flèche de Knuth)

Soient $a, b \in \mathbb{N}$. On définit

$$a \uparrow^0 b := ab$$

Pour $n > 0$, on définit

$$a \uparrow^n b := a \uparrow^{n-1} a \uparrow^{n-1} \cdots \uparrow^{n-1} a,$$

où on a b copies de a.

Pour revenir à B_i :

$$
\begin{aligned}
B_0(x) &= x+1 \\
B_1(x) &= B_0(B_0(\cdots 1 \cdots)) = 1 + (x+1) = x+2 \\
&= 2 + (x+3) - 3 \\
B_2(x) &= B_1(B_1(\cdots 1 \cdots)) = 1 + (x+1) \times 2 = 2x+3 \\
&= 2 \uparrow^0 (x+3) - 3 \\
B_3(x) &= B_2(B_2(\cdots 1 \cdots)) \\
&= 2 \uparrow^1 (x+3) - 3 \\
&\vdots \\
B_i(x) &= B_{i-1}(B_{i-1}(\cdots 1 \cdots)) \\
&= 2 \uparrow^{i-2} (x+3) - 3.
\end{aligned}
$$

Conjecture tentante... encore faut-il la prouver !

Théorème 2.6

Pour tout $i \geqslant 2$ et tout $x \in \mathbb{N}$,

$$B_i(x) = 2 \uparrow^{i-2} (x+3) - 3.$$

Démonstration :

Nous procéderons par récurrence.

Base de la récurrence. Il s'agit de $i = 2$, on l'a déjà calculé directement.

Pas de récurrence. Supposons que l'énoncé est vrai pour $i-1$:

$$B_{i-1}(x) = 2 \uparrow^{i-3} (x+3) - 3.$$

On a alors que :

$$\begin{aligned}
B_i(x) &= {B_{i-1}}^{\langle x+1 \rangle}(1) \\
&= 2 \uparrow^{i-3} ({B_{i-1}}^{\langle x \rangle}(1) + 3) - 3 \\
&= 2 \uparrow^{i-3} (2 \uparrow^{i-3} ({B_{i-1}}^{\langle x-1 \rangle}(1) + 3) - 3 + 3) - 3 \\
&= 2 \uparrow^{i-3} (2 \uparrow^{i-3} ({B_{i-1}}^{\langle x-1 \rangle}(1) + 3)) - 3 \\
&\vdots \\
&= 2(\uparrow^{i-3} (2 \uparrow^{i-3} (\ldots (B_{i-1}(1) + 3) \ldots)) - 3 \\
&= 2(\uparrow^{i-3} (2 \uparrow^{i-3} (\ldots (2 \uparrow^{i-3} 4) \ldots)) - 3 \\
&= 2(\uparrow^{i-3} (2 \uparrow^{i-3} (\ldots (2 \uparrow^{i-3} (2 \uparrow^{i-3} 2) \ldots)) - 3 \\
&= 2 \uparrow^{i-2} (x+3) - 3.
\end{aligned}$$

Définition 2.7

Pour tout programme RÉPÉTER P, $\mathcal{B}(\mathtt{P})$ est le nombre maximal d'imbrications des boucles de P.

Définition 2.8

On note $\mathcal{M}(\mathtt{P}, r_1, \ldots, r_k)$ la valeur maximale de tous les registres après l'exécution de P en fonction des inputs $r_1, \ldots, r_k$.

Propriétés de la famille des fonctions B_i

Remarque 2.9

Tous les énoncés suivants peuvent être facilement prouvés par récurrence sur i, x ou k.

- $\forall i \in \mathbb{N}, B_i(x) \geq x + 1$;
- $B_i^{\langle k \rangle}(x)$ *est croissante en* i, x *et* k ;
- $2^s x \leq B_i^{\langle s \rangle}(x)$ *pour* $i \geq 2$ *et tout* $s \geq 0$.
- $\forall k \in \mathbb{N}, B_2^{\langle k \rangle}(1) > 2k$.

Théorème 2.10

Pour tout programme RÉPÉTER $\mathtt{P}$, *si* $\mathcal{B}(\mathtt{P}) = i$, *alors*

$$\forall r_1, \dots, r_k \ : \ \mathcal{M}(\mathtt{P}, r_1, \dots, r_k) \leq {B_{i+1}}^{\langle \ell(\mathtt{P}) \rangle}(\max(r_1, \dots, r_k)),$$

où $\ell(\mathtt{P})$ *est le nombre de lignes de code de* $\mathtt{P}$.

Démonstration :

On prouvera ce théorème par récurrence structurelle sur $\mathtt{P}$.

(On peut aussi prouver le théorème par récurrence sur $\ell = \ell(\mathtt{P})$.)

Rappelons la structure récursive des programmes répéter :

- $\mathcal{S}_0 = \{\texttt{inc}(r_i) \mid i \in \mathbb{N}\} \cup \{r_i \leftarrow r_j \mid i, j \in \mathbb{N}\}$
- R_1 : La concaténation de deux programmes répéter est un programme répéter
- R_2 : Si P est un programme répéter,

$$\texttt{répéter}\ r_i\ \texttt{fois}\ [\ P\ ]$$

en est un aussi pour tout $i \in \mathbb{N}$.

Base de la récurrence. On doit prouver l'énoncé pour tous les programmes dans $\mathcal{S}_0$, donc soit une assignation ou une incrémentation de registre. On a donc :

$$\begin{aligned}\mathcal{M}(\mathtt{P}, r_1, \dots, r_k) &\leq \max(r_1, \dots, r_k) + 1 \\ &\leq B_1^{\langle 1 \rangle}(\max(r_1, \dots, r_k),\end{aligned}$$

où on a utilisé le fait que $B_1(x) = x + 2$.

Règle R_1 : Concaténation de deux programmes

On suppose que $\mathtt{P}$ est la concaténation de $\mathtt{P}_1$ et $\mathtt{P}_2$ avec $\ell_1 \geq 1$ et $\ell_2 \geq 1$ lignes de code respectivement, et une profondeur de boucle de i_1 et i_2. Supposons que le théorème est vrai pour $\mathtt{P}_1$ et $\mathtt{P}_2$, et soit $i = \max\{i_1, i_2\}$ la profondeur de boucle de $\mathtt{P}$.

On a alors que :

$$\begin{aligned}
\mathcal{M}(\mathtt{P}, r_1, \dots, r_k) &\leq B_{i_2+1}^{\langle \ell_2 \rangle}\left(B_{i_1+1}^{\langle \ell_1 \rangle}(\max(r_1, \dots, r_k))\right) \\
&\leq B_{i+1}^{\langle \ell_2 \rangle}\left(B_{i+1}^{\langle \ell_1 \rangle}(\max(r_1, \dots, r_k))\right) \\
&= B_{i+1}^{\langle \ell_1+\ell_2 \rangle}(\max(r_1, \dots, r_k)).
\end{aligned}$$

Puisque $\mathtt{P}$ a $\ell_1 + \ell_2$ lignes de code, ceci conclut la preuve du premier cas.

Règle R_2 : Boucle répéter

Soit `Q` un programme répéter, et supposons que le théorème est vrai pour celui-ci, et soit `P` le programme suivant :

$$\texttt{répéter } r_\alpha \texttt{ fois } [\texttt{Q}]$$

Soit $\ell \geq 2$ le nombre de lignes de code de `P`, (et donc `Q` a $\ell - 1 \geq 1$ lignes), et supposons que `P` a une profondeur de boucle i, ce qui veut dire que celle de `Q` est $i - 1$.

Par l'hypothèse de récurrence, $\mathtt{Q}$ peut faire augmenter la valeur des registres par la fonction $B_i^{\langle \ell-1 \rangle}$ à chaque itération, et la boucle est répétée au plus $v = \max(r_1, \ldots, r_k)$ fois. On a donc que :

$$\begin{aligned}
\mathcal{M}(\mathtt{P}, r_1, \ldots, r_k) &\leq (B_i^{\langle \ell-1 \rangle})^{\langle v \rangle}(v) \\
&= B_i^{\langle (\ell-1)v \rangle}(v) \\
&\leq B_i^{\langle (\ell-1)v \rangle}(B_i^{\langle v \rangle}(1)) \\
&= B_i^{\langle \ell v \rangle}(1) \\
&\leq B_i^{\langle \ell v+1 \rangle}(1) \\
&= B_{i+1}(\ell v) \\
&\leq B_{i+1}(2^{\ell-1} v) \\
&\leq B_{i+1}(B_{i+1}^{\langle \ell-1 \rangle}(v)) \\
&= B_{i+1}^{\langle \ell \rangle}(v).
\end{aligned}$$

Ceci conclut la preuve.

Corollaire 2.11

Pour tout $i \geq 0$, il existe une fonction qui n'est pas calculable par un programme `RÉPÉTER` avec une profondeur de boucle i, mais qui est calculable par un programme avec une profondeur de boucle $i + 1$.

La fonction d'Ackermann

Et si la profondeur de boucle devient un input du programme ? En 1928, Wilhelm Ackermann définit « sa » fonction :

Définition 2.12 (Fonction d'Ackermann)

La fonction d'Ackermann $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ est définie comme

$$A(i, x) := B_i(x).$$

Ackermann n'est pas primitive récursive.

Théorème 2.13

La fonction d'Ackermann A n'est pas calculable par un programme RÉPÉTER.

Démonstration :

Supposons que $A(y,x)$ soit calculable par un programme RÉPÉTER avec ℓ lignes de code. Ceci implique que sa profondeur de boucle i est au plus $\ell - 1$.

On doit donc avoir, à l'aide du théorème 2.10,

$$A(y,x) \leq {B_\ell}^{\langle \ell \rangle}(\max(y,x)).$$

En particulier,

$$\begin{aligned} A(\ell+2,\ell) &\leq B_\ell^{\langle \ell \rangle}(\ell+2) \\ &\leq B_\ell^{\langle \ell \rangle}(B_\ell^{\langle \ell+1 \rangle}(1)) \\ &= B_\ell^{\langle 2\ell+1 \rangle}(1) \\ &= B_{\ell+1}(2\ell). \end{aligned}$$

Donc, d'une part, $A(\ell+2,\ell) \leq B_{\ell+1}(2\ell)$.

Or, d'autre part, par la définition de A, on a que :

$$\begin{aligned} A(\ell+2,\ell) &= B_{\ell+2}(\ell) \\ &= {B_{\ell+1}}^{\langle \ell+1 \rangle}(1) \\ &= B_{\ell+1}(B_{\ell+1}^{\langle \ell \rangle}(1)) \\ &\geq B_{\ell+1}(B_2^{\langle \ell \rangle}(1)) \\ &> B_{\ell+1}(2\ell), \end{aligned}$$

et donc $A(\ell+2,\ell) > B_{\ell+1}(2\ell)$, et on a une contradiction.

□

Remarquons que la fonction

$$F(x) = A(x, x)$$

croît plus rapidement que n'importe quelle des fonctions $B_i(x)$ . . .

Les programmes TANTQUE

Les programmes TANTQUE sont semblables aux programmes RÉPÉTER, mais les boucles sont différentes :

- Un nombre arbitrairement grand de registres est disponible : r_0, r_1 ... ;
- chaque registre contient un entier positif ou nul ;
- les registres sont implicitement initialisés à 0 avant utilisation ;
- Trois types d'instructions permises :
  - $r_i \leftarrow r_j$: remplace le contenu du registre r_i par celui de r_j ;
  - `inc`(r_i) : incrémente de 1 le registre r_i ;
  - `tant que` $r_i \neq r_j$ `faire` $[\langle \text{Bloc} \rangle]$ répète l'éxécution d'un bloc d'instructions tant que les valeurs des registres r_i et r_j diffèrent.

- dans une boucle, l'inégalité est réévaluée à chaque itération et les valeurs de r_i et r_j peuvent changer.
- Un programme `TANTQUE` implante une fonction

$$f : \mathbb{N} \times \mathbb{N} \times \cdots \times \mathbb{N} \rightarrow \mathbb{N} \cup \{\uparrow\}$$

$$(r_1, r_2, \ldots, r_k) \mapsto \begin{cases} r_0 & \text{si le programme s'arrête,} \\ \uparrow & \text{si le programme boucle à l'infini.} \end{cases}$$

Au début de l'exécution, les registres r_1 à r_k contiennent les arguments de f, et à la fin, si le programme s'arrête, r_0 contient $f(r_1, \ldots, r_k)$.

Contrairement aux programmes `RÉPÉTER`, un programme `TANTQUE` peut ne jamais s'arrêter, par exemple :

$$\begin{aligned}
&\texttt{BOUCLE}(r_1) = \uparrow \\
&\qquad \texttt{inc}(r_1) \\
&\qquad \texttt{tant que } r_1 \neq r_0 \texttt{ faire } [\ \texttt{inc}(r_2)\ ]
\end{aligned}$$

Remarque 2.14

Tout programme RÉPÉTER peut être simulé par un programme TANTQUE.

Il suffit de remplacer les boucles de la forme

$$\texttt{répéter}\ r_i\ \texttt{fois}\ [\,...\,]$$

par

$$\begin{array}{l} r_k \leftarrow r_i \\ \texttt{tant que}\ r_j \neq r_k\ \texttt{faire}\ [\\ \qquad ... \\ \qquad \texttt{inc}(r_j) \\ ] \end{array}$$

où r_j et r_k sont des registres non utilisés.

Sucre syntaxique

À la lumière de la remarque 2.14, on se permettra d'utiliser les instructions `répéter` dans les programmes `TANTQUE`.

On peut donc recycler en programmes `TANTQUE` tous les programmes `RÉPÉTER` que nous avons vus.

Ackermann est calculable par un programme TANTQUE.

Nous allons exhiber un programme TANTQUE qui implante la fonction d'Ackermann telle que présentée à la définition 2.12. Les détails de la preuve qui montrent que ce programme implante effectivement la fonction souhaitée seront omis.

Théorème 2.15

La fonction d'Ackermann A est calculable par un programme TANTQUE.

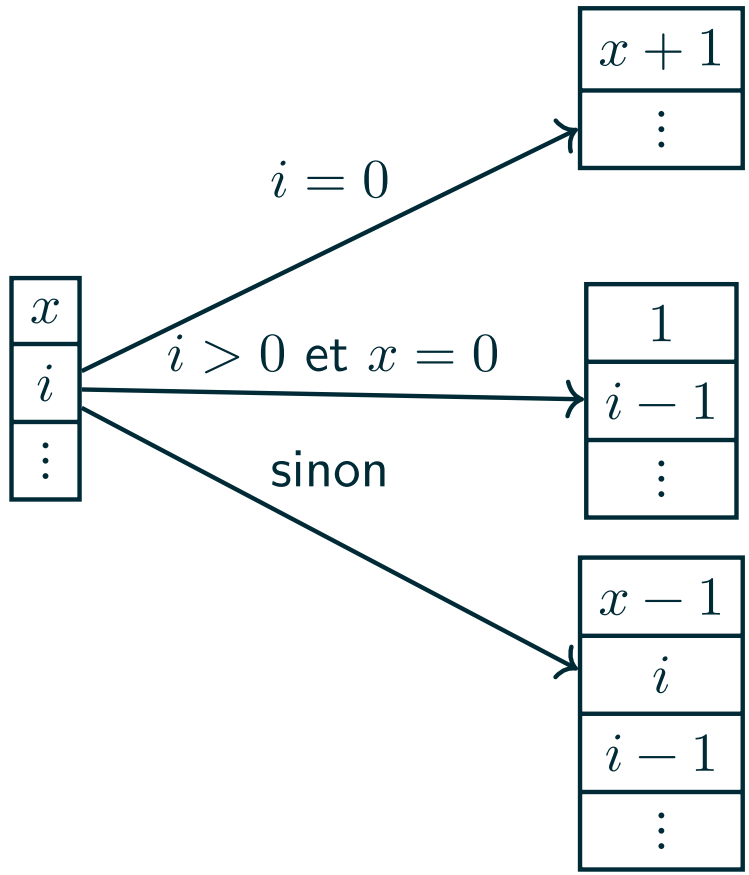
Aperçu de la démonstration :

La fonction d'Ackermann sera implantée à l'aide d'une pile.

Idée :

Notons que $A(i,x)$ peut s'écrire sans les B_i comme :

$$A(i,x) = \begin{cases} x+1 & \text{si } i = 0 \\ A(i-1,1) & \text{si } i > 0 \text{ et } x = 0 \\ A(i-1,A(i,x-1)) & \text{sinon.} \end{cases}$$



Écrivons ça d'abord en « pseudo-tantque » :

$\texttt{ACKERMANN}(i, x) = A(i, x)$

$\texttt{empile}(i)$

$\texttt{empile}(x)$

tant que la pile contient plus d'un élément `faire` [

$x' \leftarrow \texttt{dépile}$

$i' \leftarrow \texttt{dépile}$

`si` $i' = 0$ `alors` [$\texttt{empile}(x' + 1)$]

`si` $i' > 0$ et $x' = 0$ `alors` [

$\texttt{empile}(i' - 1)$ $\texttt{empile}(1)$

]

`sinon` [

$\texttt{empile}(i' - 1)$ $\texttt{empile}(i')$ $\texttt{empile}(x' - 1)$

]

]

$r_0 \leftarrow \texttt{dépile}$

En TANTQUE plus orthodoxe, avec P et t jouant le rôle de la pile et de l'adresse du haut de la pile :

$\texttt{ACKERMANN}(i, x) = A(i, x)$

$P \leftarrow 1$

$\texttt{inc}(t) \qquad P \leftarrow \texttt{TABLASS}(P, t, i)$

$\texttt{inc}(t) \qquad P \leftarrow \texttt{TABLASS}(P, t, x)$

$\texttt{tant que } t \neq 1 \texttt{ faire } [$

$x' \leftarrow \texttt{TABLVAL}(P, t)$

$i' \leftarrow \texttt{TABLVAL}(P, t - 1)$

$\texttt{si } i' = 0 \texttt{ alors } [$

$t \leftarrow \texttt{DEC}(t) \qquad \texttt{TABLASS}(P, t, x' + 1)$

$]$

$\texttt{si ET}(i' \neq 0, x' = 0) \texttt{ alors } [$

$\texttt{TABLASS}(P, t - 1, i' - 1) \qquad \texttt{TABLASS}(P, t, 1)$

$]$

$$
\begin{array}{l}
\quad \texttt{si ET}(i' \neq 0, x' \neq 0)\ \texttt{alors}\ [\\
\qquad\quad \texttt{inc}(t) \\
\qquad\quad \texttt{TABLASS}(P, t-2, i'-1) \\
\qquad\quad \texttt{TABLASS}(P, t-1, i') \\
\qquad\quad \texttt{TABLASS}(P, t, x'-1) \\
\quad] \\
] \\
r_0 \leftarrow \texttt{TABLVAL}(P, 1)
\end{array}
$$

□