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 quatre modèles de calcul :

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

Les programmes RÉPÉTER

- Un nombre arbitrairement grand de registres est disponible : $r_0,\ r_1\ \ldots$;
- chaque registre contient un entier positif ou nul ;
- les registres sont implicitement initialisés à 0 avant utilisation ;
- l'instruction $r_i \leftarrow r_j$ remplace le contenu du registre r_i par celui de r_j ;
- l'instruction $\texttt{inc}(r_i)$ incrémente de 1 le registre r_i ;

- `répéter` r_i `fois` [⟨BLOC⟩] répète l'éxécution d'un bloc d'instructions r_i fois ;
- le nombre d'éxécution de ⟨BLOC⟩ 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

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

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

Grammaire pour la syntaxe des programmes RÉPÉTER

$$
\begin{aligned}
S &\to \varepsilon \\
&\quad \mid \langle \textsc{Incrémentation} \rangle S \\
&\quad \mid \langle \textsc{Affectation} \rangle S \\
&\quad \mid \langle \textsc{Répéter} \rangle S \\
\langle \textsc{Incrémentation} \rangle &\to \texttt{inc}(V) \\
\langle \textsc{Affectation} \rangle &\to V \leftarrow V \\
\langle \textsc{Répéter} \rangle &\to \texttt{répéter}\ V\ \texttt{fois}\ [S] \\
V &\to r_N \\
N &\to C \mid CN \\
C &\to \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}
\end{aligned}
$$

Addition

$\texttt{PLUS}(r_1, r_2) = r_1 + r_2$

$r_0 \leftarrow r_1$

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

$\texttt{inc}(r_0)$

$]$

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

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

$\texttt{inc}(r_0)$

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

$r_3 \leftarrow r_4$

$\texttt{répéter}\ r_0\ \texttt{fois}\ [$

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

$\texttt{inc}(r_3)$

$]$

$]$

$r_0 \leftarrow r_3$

$]$

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.

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

$r_0 \leftarrow 1$

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

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

$]$

Décrémentation

$$\texttt{DEC}(r_1) = \max(0, r_1 - 1)$$

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

$$r_0 \leftarrow r_2$$

$$\texttt{inc}(r_2)$$

$$]$$

Soustraction

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

$r_0 \leftarrow r_1$

répéter r_2 **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 $\langle$BLOC$\rangle$ conditionnellement à la valeur booléenne r_i on répète $\langle$BLOC$\rangle$ r_i fois.

L'instruction

$$\texttt{si}\ r_i\ \texttt{alors}\ [\langle \text{BLOC} \rangle]$$

sera mise pour

$$\texttt{répéter}\ r_i\ \texttt{fois}\ [\langle \text{BLOC} \rangle].$$

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

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

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

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

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

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

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

$$
\begin{array}{l}
\texttt{DIV}(r_1, r_2) = \lfloor \dfrac{r_1}{r_2} \rfloor \\
\qquad \texttt{répéter}\ r_1\ \texttt{fois}\ [\\
\qquad\qquad r_3 \leftarrow \texttt{PLUS}(r_3, r_2) \\
\qquad\qquad r_4 \leftarrow \texttt{PG?}(r_3, r_1) \\
\qquad\qquad r_4 \leftarrow \texttt{NEG}(r_4) \\
\qquad\qquad \texttt{si}\ r_4\ \texttt{alors}\ [\\
\qquad\qquad\qquad \texttt{inc}(r_0) \\
\qquad\qquad] \\
\qquad]
\end{array}
$$

Modulo

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

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

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

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

Test de primalité

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

$r_0 \leftarrow \texttt{faux}$

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

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

$r_0 \leftarrow \texttt{vrai}$

$r_3 \leftarrow 1$

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

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

$\texttt{inc}(r_3)$

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

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

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

$]$

$]$

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

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

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

$]$

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

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

est représenté sans ambiguïté par l'entier

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

Extraction d'un élément d'un tableau

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

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

$r_4 \leftarrow r_3$

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

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

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

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

$\texttt{inc}(r_0)$

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

$]$

$]$

Assignation d'un élément dans un tableau

$\texttt{TABLASS}(r_1, r_2, r_3) =$ le 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. ▲

Notation 2.3. Pour une fonction $f : \mathbb{N} \to \mathbb{N}$, et un entier n, on note :

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

▲

Boucles imbriquées

Définition 2.4. Pour $i \geq 0$:

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

$$x \mapsto B_i(x) = \begin{cases} 1 & \text{si} \quad i = 0,\ x = 0 \\ 2 & \text{si} \quad i = 0,\ x = 1 \\ x + 2 & \text{si} \quad i = 0,\ x \geq 2 \\ {B_{i-1}}^{\langle x \rangle}(1) & \text{si} \quad i > 0 \end{cases}$$

▲

On remarque que

$$\begin{aligned} B_0(x) &= x+2 \quad \text{si} \quad x \geq 2 \\ B_1(x) &= 2x \quad \text{si} \quad x \geq 1 \\ B_2(x) &= 2^x \quad \text{si} \quad x \geq 0 \\ B_3(x) &= \underbrace{2^{2^{2^{\cdots}}}}_{x \text{ fois}} \quad \text{si} \quad x \geq 1 \end{aligned}$$

Il est clair que plus i est grand, plus B_i est une fonction qui croît rapidement.

La valeur de $B_3(5)$ compte 19729 chiffres.

La valeur de $B_3(6)$ compte plus de chiffres que le nombre d'atomes dans l'univers.

La fonction B_3 croît **très** rapidement, mais ce n'est rien si on la compare à B_4.

Le taux de croissance de la fonction B_{100} dépasse l'entendement...

Lemme 2.5. Pour tout $i \geq 0$, B_i est calculable par un programme RÉPÉTER.

Preuve.

$\mathtt{B}_0(r_1)$

$r_0 \leftarrow \mathtt{PLUS}(r_1, 1)$

$r_2 \leftarrow \mathtt{PG?}(r_0, 2)$

$\mathtt{si}\ r_2\ \mathtt{alors}\ [$

$\mathtt{inc}(r_0)$

$]$

Pour $i > 0$ fixé :

$\mathtt{B}_i(r_1)$

$\quad \mathtt{inc}(r_0)$

$\quad \mathtt{répéter}\ r_1\ \mathtt{fois}\ [$

$\quad\quad r_0 \leftarrow \mathtt{B}_{i-1}(r_0)$

$\quad]$

■

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

Un programme RÉPÉTER peut calculer des fonctions qui croissent très rapidement.

Définition 2.6. Pour tout programme RÉPÉTER P, $\mathcal{B}(\mathtt{P})$ est le nombre maximal d'imbrications des boucles de P. ▲

Définition 2.7. On note $\mathcal{M}(\mathtt{P}, r_1, \ldots, r_k)$ la valeur maximale des variables $r_1, \ldots, r_k$ après l'exécution de P. ▲

Propriétés de la famille des fonctions B_i

Remarques 2.8. Tous les énoncés ci-dessous peuvent être facilement prouvés par induction sur i, x ou k.

- $\mathcal{B}(\mathtt{B}_i) = i$;
- $B_i(x) \geq x + 1$;
- ${B_i}^{\langle k \rangle}(x)$ est croissante en i, x et k ;
- $2{B_i}^{\langle k \rangle}(x) \leq {B_i}^{\langle k+1 \rangle}(x)$ pour $i \geq 1$;
- ${B_i}^{\langle k \rangle}(x) + x \leq {B_i}^{\langle k+1 \rangle}(x)$ pour $i \geq 1$.

▲

Théorème 2.9. Pour tout programme RÉPÉTER P, si $\mathcal{B}(\mathtt{P}) = i$, alors il existe un entier s tel que

$$\forall r_1, \ldots, r_k \ : \ \mathcal{M}(\mathtt{P}, r_1, \ldots, r_k) \leq {B_i}^{\langle s \rangle}(\max(r_1, \ldots, r_k)).$$

Preuve. La preuve est par induction sur $i = \mathcal{B}(\mathtt{P})$.

Base de l'induction. Soit $i = 0$.

Soit c le nombre d'instructions `inc` dans `P`.

Choisissons $s = \lceil \frac{c}{2} \rceil$.

Comme `P` ne contient pas de boucle, on a :

$$\begin{aligned} \mathcal{M}(\mathtt{P}, r_1, \ldots, r_k) &\leq \max(r_1, \ldots, r_k) + c \\ &\leq {B_0}^{\langle \lceil \frac{c}{2} \rceil \rangle}(\max(r_1, \ldots, r_k)) \\ &= {B_0}^{\langle s \rangle}(\max(r_1, \ldots, r_k)). \end{aligned}$$

Hypothèse d'induction. Soit $i > 0$. Pour tout programme $\mathtt{Q}$ tel que $\mathcal{B}(\mathtt{Q}) = i - 1$, il existe un entier s tel que

$$\mathcal{M}(\mathtt{Q}, r_1, \ldots, r_k) \leq {B_{i-1}}^{\langle s \rangle}(\max(r_1, \ldots, r_k)).$$

Pas d'induction. Il faut obtenir le résultat pour $i > 0$.

Soit $\mathtt{P}$ un programme tel que $\mathcal{B}(\mathtt{P}) = i$.

Clairement, $\mathtt{P}$ peut être décomposé en la forme suivante où, pour chaque j, $\mathcal{B}(\mathtt{Q}_j) = i - 1$ et $\mathcal{B}(\mathtt{P}_j) \leq i - 1$:

$$
\begin{array}{l}
\mathtt{P}(r_1, \ldots, r_k) \\
\qquad \mathtt{P}_1 \\
\qquad \mathtt{répéter}\ r_{\alpha_1}\ \mathtt{fois}\ [\mathtt{Q}_1] \\
\qquad \vdots \\
\qquad \mathtt{P}_m \\
\qquad \mathtt{répéter}\ r_{\alpha_m}\ \mathtt{fois}\ [\mathtt{Q}_m]
\end{array}
$$

Soit $v = \max(r_1, \ldots, r_k)$.

Par hypothèse d'induction, la valeur maximale d'une variable après l'exécution de $\mathtt{P}_1$ sera $u = {B_{i-1}}^{\langle s \rangle}(v)$, pour un certain s.

La première boucle sera donc répétée au plus u fois.

Après cette boucle, la valeur maximale d'une variable sera donc, par hypothèse d'induction :

$$({B_{i-1}}^{\langle s \rangle})^{\langle u \rangle}(u).$$

D'où :

$$
\begin{aligned}
({B_{i-1}}^{\langle s \rangle})^{\langle u \rangle}(u) &= {B_{i-1}}^{\langle su \rangle}(u) \\
&= {B_{i-1}}^{\langle su \rangle}({B_{i-1}}^{\langle s \rangle}(v)) \\
&\leq {B_{i-1}}^{\langle su \rangle}({B_{i-1}}^{\langle s \rangle}({B_{i-1}}^{\langle v \rangle}(1))) \\
&= {B_{i-1}}^{\langle s(u+1)+v \rangle}(1) \\
&= B_i(s({B_{i-1}}^{\langle s \rangle}(v)+1)+v)
\end{aligned}
$$

$$
\begin{aligned}
&\leq \quad B_i(B_{i-1}{}^{\langle 2s \rangle}(v) + v) \\
&\leq \quad B_i(B_{i-1}{}^{\langle 2s+1 \rangle}(v)) \\
&\leq \quad B_i(B_i{}^{\langle 2s+1 \rangle}(v)) \\
&= \quad B_i{}^{\langle 2s+2 \rangle}(v) \\
&= \quad B_i{}^{\langle s_1 \rangle}(v).
\end{aligned}
$$

En continuant de la même façon, on montre qu'après la boucle j la valeur maximale est bornée par

$$B_i{}^{\langle s_j \rangle}(v)$$

pour un certain s_j, ce qui conclut la preuve du théorème. ■

Corollaire 2.10. 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

En 1928, Ackermann définit la fonction à deux variables suivante :

Définition 2.11.

$$A(i,x) = \begin{cases} 1 & \text{si} \quad x = 0 \\ 2 & \text{si} \quad i = 0,\ x = 1 \\ x+2 & \text{si} \quad i = 0,\ x \geq 2 \\ A(i-1, A(i, x-1)) & \text{si} \quad i > 0,\ x > 0 \end{cases}$$

▲

Intuitivement, on peut voir que A peut être calculée.

- $A(0, x)$ est facilement calculable ;
- $A(1, 0) = 1$;
- si $A(1, x)$ est calculable, alors $A(1, x + 1) = A(0, A(1, x))$ l'est aussi ;
- $A(2, 0) = 1$;
- si $A(2, x)$ est calculable alors $A(2, x + 1) = A(1, A(2, x))$ l'est aussi ;
- etc.

Mais la fonction d'Ackermann est-elle primitive récursive ?

C'est-à-dire : peut-elle être calculée par un programme `RÉPÉTER` ?

Lemme 2.12.

$$\forall i \geq 0,\ \forall x \geq 0\ :\ A(i,x) = B_i(x).$$

Preuve. Le lemme sera prouvé par induction d'abord sur i et ensuite sur x.

Base de l'induction. Soit $i = 0$. Par définition de A et de B_0, on a :

$$\forall x \geq 0\ :\ A(0,x) = B_0(x).$$

Hypothèse d'induction. Pour $i > 0$ on a :

$$\forall x \geq 0\ :\ A(i-1,x) = B_{i-1}(x).$$

Pas d'induction. Soit $i > 0$. Montrons que $A(i, x) = B_i(x)$ par induction sur x.

Base de l'induction. Soit $x = 0$. On a :

$$A(i, 0) = 1 = {B_{i-1}}^{\langle 0 \rangle}(1) = B_i(0).$$

Hypothèse d'induction. Pour $x > 0$ on a :

$$A(i, x - 1) = B_i(x - 1).$$

Pas d'induction. Soit $x > 0$. On a :

$$\begin{aligned} A(i,x) &= A(i-1, A(i,x-1)) \\ &= A(i-1, B_i(x-1)) \\ &= B_{i-1}(B_i(x-1)) \\ &= B_{i-1}({B_{i-1}}^{\langle x-1 \rangle}(1)) \\ &= {B_{i-1}}^{\langle x \rangle}(1) \\ &= B_i(x). \end{aligned}$$

■

Ackermann n'est pas primitive récursive.

Théorème 2.13. La fonction d'Ackermann A n'est pas calculable par un programme RÉPÉTER.

Preuve. Supposons que $A(y, x)$ soit calculable par un programme RÉPÉTER avec boucle de prodondeur maximale i.

On doit donc avoir, à l'aide du théorème 2.9,

$$A(y, x) \leq {B_i}^{\langle s \rangle}(\max(y, x))$$

pour un certain s.

Pour $y = i + 1$ et x suffisamment grand :

$$\begin{aligned} A(y,x) &= A(i+1,x) \\ &= B_{i+1}(x) \\ &> {B_i}^{\langle s \rangle}(x) \\ &= {B_i}^{\langle s \rangle}(\max(i+1,x)) \\ &= {B_i}^{\langle s \rangle}(\max(y,x)), \end{aligned}$$

ce qui contredit l'hypothèse. ■

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 : $r_0,\ r_1\ \ldots$;
- chaque registre contient un entier positif ou nul ;
- les registres sont implicitement initialisés à 0 avant utilisation ;
- l'instruction $r_i \leftarrow r_j$ remplace le contenu du registre r_i par celui de r_j ;
- l'instruction $\texttt{inc}(r_i)$ incrémente de 1 le registre r_i ;

- `tant que` $r_i \neq r_j$ `faire` [⟨BLOC⟩] 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 :

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

Remarque 2.14. Tout programme RÉPÉTER peut être simulé par un programme TANTQUE.

Il suffit de remplacer les boucles de la forme

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

par

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

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 comme des 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 2.11. 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 preuve. La fonction d'Ackermann sera implantée à l'aide d'une pile.

À l'entrée d'une boucle `tant que`, les deux premiers éléments au haut de la pile sont les arguments i et x de la définition 2.11.

À la sortie de la boucle, ces deux éléments auront été remplacés par $A(i, x)$ si $i = 0$ ou $x = 0$, ou par $(i - 1,\ i,\ x - 1)$ si $i > 0$ et $x > 0$.

La pile elle-même est réalisée à l'aide des programmes RÉPÉTER `TABLVAL` et `TABLASS` définis plus haut.

Voici le rôle joué par certains des registres utilisés dans le programme :

r_3 :	la pile
r_4 :	adresse du premier élément au haut de la pile
r_5 :	adresse de i
r_6 :	adresse de x
r_7 :	i
r_8 :	x
r_{10} est `vrai` :	$x = 0$
r_9 et r_{12} et r_{14} sont `vrai` :	$i = 0$ et $x = 1$
r_9 et r_{12} et r_{13} sont `vrai` :	$i = 0$ et $x \geq 2$
r_9 et r_{11} sont `vrai` :	$i > 0$ et $x > 0$

$$
\begin{array}{l}
\texttt{ACKERMANN}(r_1, r_2) = A(r_1, r_2) \\
\qquad r_3 \leftarrow 1 \\
\qquad \texttt{inc}(r_4) \qquad r_3 \leftarrow \texttt{TABLASS}(r_3, r_4, r_1) \\
\qquad \texttt{inc}(r_4) \qquad r_3 \leftarrow \texttt{TABLASS}(r_3, r_4, r_2) \\
\qquad \texttt{tant que } r_4 \neq 1 \texttt{ faire } [\\
\qquad\qquad r_5 \leftarrow \texttt{DEC}(r_4) \qquad r_7 \leftarrow \texttt{TABLVAL}(r_3, r_5) \\
\qquad\qquad r_6 \leftarrow r_4 \qquad r_8 \leftarrow \texttt{TABLVAL}(r_3, r_6) \\
\qquad\qquad r_9 \leftarrow \texttt{PG?}(r_8, 0) \qquad r_{10} \leftarrow \texttt{NEG}(r_9) \\
\qquad\qquad r_{11} \leftarrow \texttt{PG?}(r_7, 0) \qquad r_{12} \leftarrow \texttt{NEG}(r_{11}) \\
\qquad\qquad r_{13} \leftarrow \texttt{PG?}(r_8, 1) \qquad r_{14} \leftarrow \texttt{NEG}(r_{13}) \\
\qquad\qquad \texttt{si } r_{10} \texttt{ alors } [\\
\qquad\qquad\qquad r_4 \leftarrow \texttt{DEC}(r_4) \qquad \texttt{TABLASS}(r_3, r_4, 1) \\
\qquad\qquad]
\end{array}
$$

$$
\begin{array}{l}
\quad \texttt{si}\ r_9\ \texttt{alors}\ [\\
\qquad \texttt{si}\ r_{12}\ \texttt{alors}\ [\\
\qquad\quad \texttt{si}\ r_{14}\ \texttt{alors}\ [\ r_4 \leftarrow \texttt{DEC}(r_4) \quad \texttt{TABLASS}(r_3, r_4, 2)\ ] \\
\qquad\quad \texttt{si}\ r_{13}\ \texttt{alors}\ [\ r_{15} \leftarrow \texttt{PLUS}(r_8, 2) \quad r_4 \leftarrow \texttt{DEC}(r_4) \quad \texttt{TABLASS}(r_3, r_4, r_{15})\ ] \\
\qquad] \\
\qquad \texttt{si}\ r_{11}\ \texttt{alors}\ [\\
\qquad\quad r_{15} \leftarrow \texttt{DEC}(r_7) \quad \texttt{TABLASS}(r_3, r_5, r_{15}) \\
\qquad\quad \texttt{TABLASS}(r_3, r_6, r_7) \\
\qquad\quad r_{15} \leftarrow \texttt{DEC}(r_8) \quad \texttt{inc}(r_4) \quad \texttt{TABLASS}(r_3, r_4, r_{15}) \\
\qquad] \\
\quad] \\
] \\
r_0 \leftarrow \texttt{TABLVAL}(r_3, r_4)
\end{array}
$$

■