

# 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 :  $r_0, r_1 \dots$ ;
- 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  $\text{inc}(r_i)$  incrémente de 1 le registre  $r_i$  ;

- répéter  $r_i$  fois [ $\langle \text{BLOC} \rangle$ ] répète l'exécution d'un bloc d'instructions  $r_i$  fois ;
- le nombre d'exécution de  $\langle \text{BLOC} \rangle$  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

$$f : \mathbb{N} \times \mathbb{N} \times \dots \times \mathbb{N} \rightarrow \mathbb{N}$$

$$(r_1, r_2, \dots, r_k) \mapsto r_0.$$

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

# Grammaire pour la syntaxe des programmes RÉPÉTER

$$\begin{aligned} S &\rightarrow \varepsilon \\ &\quad | \langle \text{INCRÉMENTATION} \rangle S \\ &\quad | \langle \text{AFFECTATION} \rangle S \\ &\quad | \langle \text{RÉPÉTER} \rangle S \\ \langle \text{INCRÉMENTATION} \rangle &\rightarrow \text{inc}(V) \\ \langle \text{AFFECTATION} \rangle &\rightarrow V \leftarrow V \\ \langle \text{RÉPÉTER} \rangle &\rightarrow \text{répéter } V \text{ fois } [S] \\ V &\rightarrow r_N \\ N &\rightarrow C \mid CN \\ C &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

# Addition

PLUS( $r_1, r_2$ ) =  $r_1 + r_2$

$r_0 \leftarrow r_1$

répéter  $r_2$  fois [

    inc( $r_0$ )

]

# Multiplication

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

```
répéter  $r_1$  fois [  
    répéter  $r_2$  fois [  
        inc( $r_0$ )  
    ]  
]
```

# Exponentiation

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

inc( $r_0$ )

répéter  $r_2$  fois [

$r_3 \leftarrow r_4$

répéter  $r_0$  fois [

répéter  $r_1$  fois [

inc( $r_3$ )

]

]

$r_0 \leftarrow r_3$

$r_0 * r_1$  fois

$r_0 = \{1, r_1, r_1 * r_1, r_1 * r_1 * r_1, \dots, r_1^{r_2}\}$

# Sucre syntaxique

- L'instruction

$$r_i \leftarrow \text{PROC}(r_{j_1}, \dots, r_{j_k})$$

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  $\text{PROC}(r_{j_1}, \dots, r_{j_k})$ , 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.

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

$$r_0 \leftarrow 1$$

répéter  $r_2$  fois [

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

]

# Décrémentation

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

répéter  $r_1$  fois [

$$r_0 \leftarrow r_2$$

inc( $r_2$ )

]

# Soustraction

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

$$r_0 \leftarrow r_1$$

répéter  $r_2$  fois [

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

]

# Factorielle

$\text{FACT}(r_1) = r_1!$

$r_0 \leftarrow 1$

```
répéter  $r_1$  fois [  
     $\text{inc}(r_2)$   
     $r_0 \leftarrow \text{MULT}(r_0, r_2)$   
]
```



$r_0 = \{1*1, 1*1*2, 1*1*2*3, 1*1*2*3*4, \dots, 1*1*2*3*4 \dots r_1\}$

# Sucre syntaxique pour les variables booléennes

Nous adoptons les conventions syntaxiques suivantes :

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

Pour évaluer  $\langle \text{BLOC} \rangle$  conditionnellement à la valeur booléenne  $r_i$  on répète  $\langle \text{BLOC} \rangle$   $r_i$  fois.

L'instruction

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

sera mise pour

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

# Et

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

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

# Négation

NEG( $r_1$ )

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

# Ou

$OU(r_1, r_2)$

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

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

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

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

# Plus grand que

$PG?(r_1, r_2) = (r_1 > r_2)$

$r_3 \leftarrow \text{MOINS}(r_1, r_2)$

répéter  $r_3$  fois [

$r_0 \leftarrow \text{vrai}$

]

# Division

$$\text{DIV}(r_1, r_2) = \lfloor \frac{r_1}{r_2} \rfloor$$

répéter  $r_1$  fois [

$r_3 = \{0, r_2, 2*r_2, 3*r_2, \dots, r_1*r_2\}$   
 $r_4 = \{1, 1, 1, 1, \dots, 1, 0, 0, \dots, 0\}$



$\text{DIV}(r_1, r_2)$

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

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

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

si  $r_4$  alors [

$\text{inc}(r_0)$

]

]

# Modulo

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

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

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

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

# Test de primalité

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

$r_0 \leftarrow \text{faux}$

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

si  $r_5$  alors [

$r_0 \leftarrow \text{vrai}$

$r_3 \leftarrow 1$

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

répéter  $r_2$  fois [

$\text{inc}(r_3)$

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

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

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

]

]

# Prochain nombre premier

PREMIERSUIV( $r_1$ ) = le plus petit nombre premier plus grand que  $r_1$

$r_2 \leftarrow \text{FACT}(r_1)$

$\text{inc}(r_2)$

$r_3 \leftarrow \text{vrai}$

répéter  $r_2$  fois [

$\text{inc}(r_1)$

$r_4 \leftarrow \text{PREMIER?}(r_1)$

$r_4 \leftarrow \text{ET}(r_3, r_4)$

    si  $r_4$  alors [

$r_0 \leftarrow r_1$

$r_3 \leftarrow \text{faux}$

    ]

]

# $k$ -ème nombre premier

PREMIERK( $r_1$ ) = le  $r_1$ -ème nombre premier

répéter  $r_1$  fois [

$r_0 \leftarrow$  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 représenté sans ambiguïté par l'entier

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

# Extraction d'un élément d'un tableau

TABLVAL( $r_1, r_2$ ) =  $r_2$ -ème élément du tableau  $r_1$

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

$r_4 \leftarrow r_3$

répéter  $r_1$  fois [

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

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

si  $r_5$  alors [

$\text{inc}(r_0)$

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

]

$r_0 = v$  à la fin

Exécuté si  $r_1 \bmod r_4 = 0$

1.  $r_4 = r_2$ -ième nbre premier  $p$

2.  $r_4 = p^2$

3.  $r_4 = p^3$

....

4.  $r_4 = p^v$ ,  $v$  est la valeur du  $r_2$ -ième élément du tableau ]

# Assignment d'un élément dans un tableau

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

$$r_4 \leftarrow \text{TABLVAL}(r_1, r_2)$$

$$r_5 \leftarrow \text{PREMIERK}(r_2)$$

$$r_6 \leftarrow \text{EXP}(r_5, r_4)$$

$$r_0 \leftarrow \text{DIV}(r_1, r_6)$$

$$r_7 \leftarrow \text{EXP}(r_5, r_3)$$

$$r_0 \leftarrow \text{MULT}(r_0, r_7)$$

Soit  $v$  la valeur initiale du tableau en position  $r_2$

Soit  $v'=r_3$  la nouvelle valeur à assigner en position  $r_2$

Soit  $p$  le  $r_2$ -ième nombre premier:

$$r_4=v$$

$$r_5=p$$

$$r_6=p^v$$

$r_0=r_1$  sans l'élément  $v$  en position  $r_2$

$$r_7=p^{v'}$$

$$r_0 = r_0 * p^{v'}$$

# 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  $f : \mathbb{N} \rightarrow \mathbb{N}$ , et un entier  $n$ , on note :

$$f^{\langle 0 \rangle}(x) = x$$

$$f^{\langle 1 \rangle}(x) = f(x)$$

$$f^{\langle 2 \rangle}(x) = f(f(x))$$

$$\vdots$$

$$f^{\langle n \rangle}(x) = \underbrace{f(f(\dots x \dots))}_{n \text{ fois}}$$



# Boucles imbriquées

**Définition 2.4.** Pour  $i \geq 0$  :

$B_i$  :  $\mathbb{N} \rightarrow \mathbb{N}$

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



On remarque que

$$B_0(x) = x + 2 \quad \text{si } x \geq 2$$

$$B_1(x) = 2x \quad \text{si } x \geq 1$$

$$B_2(x) = 2^x \quad \text{si } x \geq 0$$

$$B_3(x) = \underbrace{2^{2^{2^{\dots}}}}_{x \text{ fois}} \quad \text{si } x \geq 1$$

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.**

```
B0(r1)
  r0 ← PLUS(r1, 1)
  r2 ← PG?(r0, 2)
  si r2 alors [
    inc(r0)
  ]
```

Pour  $i > 0$  fixé :

```
Bi(r1)  
  inc(r0)  
  répéter r1 fois [  
    r0 ← Bi-1(r0)  
  ]
```



On remarque que le programme  $B_i$  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  $P$ ,  $\mathcal{B}(P)$  est le nombre maximal d'imbrications des boucles de  $P$ . ▲

**Définition 2.7.** On note  $\mathcal{M}(P, r_1, \dots, r_k)$  la valeur maximale des variables  $r_1, \dots, r_k$  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  $k$ .

- $\mathcal{B}(B_i) = i$  ;
- $B_i(x) \geq x + 1$  ;
- $B_i^{\langle k \rangle}(x)$  est croissante en  $i$ ,  $x$  et  $k$  ;
- $2B_i^{\langle k \rangle}(x) \leq B_i^{\langle k+1 \rangle}(x)$  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  $\mathcal{B}(P) = i$ , alors il existe un entier  $s$  tel que

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

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

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

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

Choisissons  $s = \text{plancher}(c/2) + 1$

Comme P ne contient pas de boucle, on a :

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

plutôt  $\text{plancher}(c/2) + 1$

Si  $x \geq 2$ :

$$B_0(x) = x + 2$$

et

$$\begin{aligned} B_0^{\langle s \rangle}(x) &= B_0(B_0(\dots(B_0(x))\dots)) \\ &= B_0(B_0(\dots(x+2)\dots)) \\ &= x + 2^s \geq x + c \end{aligned}$$

Si  $x = 1$ :

$$\begin{aligned} B_0^{\langle s \rangle}(1) &= B_0(B_0(\dots(B_0(1))\dots)) + 1 \\ &= B_0(B_0(\dots(B_0(2))\dots)) + 1 \\ &= 2 + 2^{s-1} + 1 = 2s + 1 \geq c + 1 \end{aligned}$$

Si  $x = 0$  alors vérifiez vous même...

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

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

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

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

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

$$P(r_1, \dots, r_k)$$
$$P_1$$
$$\text{répéter } r_{\alpha_1} \text{ fois } [Q_1]$$
$$\vdots$$
$$P_m$$
$$\text{répéter } r_{\alpha_m} \text{ fois } [Q_m]$$

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

Par hypothèse d'induction, la valeur maximale d'une variable après l'exécution de  $P_1$  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 :

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

$$\leq B_i(B_{i-1}^{\langle 2s \rangle}(v) + v)$$

$$\leq B_i(B_{i-1}^{\langle 2s+1 \rangle}(v))$$

$$\leq B_i(B_i^{\langle 2s+1 \rangle}(v))$$

$$= B_i^{\langle 2s+2 \rangle}(v)$$

$$= B_i^{\langle s_1 \rangle}(v).$$

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.**

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



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 réursive.

**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 profondeur 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  $B_i(x) \dots$

# Les programmes TANTQUE

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

- Un nombre arbitrairement grand de registres est disponible :  $r_0, r_1 \dots$  ;
- 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  $\text{inc}(r_i)$  incrémente de 1 le registre  $r_i$  ;

- tant que  $r_i \neq r_j$  faire  $[\langle \text{BLOC} \rangle]$  répète l'exécution 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

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

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

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  $f(r_1, \dots, r_k)$ .

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

BOUCLE( $r_1$ ) = ↑

inc( $r_1$ )

tant que  $r_1 \neq r_0$  faire [ ]

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

Il suffit de remplacer les boucles de la forme

répéter  $r_i$  fois [...]

par

$r_k \leftarrow r_i$

tant que  $r_j \neq r_k$  faire [

...

inc( $r_j$ )

]

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 $x > 0$

ACKERMANN( $r_1, r_2$ ) =  $A(r_1, r_2)$

$r_3 \leftarrow 1$

inc( $r_4$ )       $r_3 \leftarrow \text{TABLASS}(r_3, r_4, r_1)$

inc( $r_4$ )       $r_3 \leftarrow \text{TABLASS}(r_3, r_4, r_2)$

tant que  $r_4 \neq 1$  faire [

$r_5 \leftarrow \text{DEC}(r_4)$        $r_7 \leftarrow \text{TABLVAL}(r_3, r_5)$

$r_6 \leftarrow r_4$        $r_8 \leftarrow \text{TABLVAL}(r_3, r_6)$

$r_9 \leftarrow \text{PG?}(r_8, 0)$        $r_{10} \leftarrow \text{NEG}(r_9)$

$r_{11} \leftarrow \text{PG?}(r_7, 0)$        $r_{12} \leftarrow \text{NEG}(r_{11})$

$r_{13} \leftarrow \text{PG?}(r_8, 1)$        $r_{14} \leftarrow \text{NEG}(r_{13})$

    si  $r_{10}$  alors [

$r_4 \leftarrow \text{DEC}(r_4)$        $\text{TABLASS}(r_3, r_4, 1)$

    ]

```

si r9 alors [
  si r12 alors [
    si r14 alors [ r4 ← DEC(r4)  TABLASS(r3, r4, 2) ]
    si r13 alors [ r15 ← PLUS(r8, 2)  r4 ← DEC(r4)  TABLASS(r3, r4, r15) ]
  ]
  si r11 alors [
    r15 ← DEC(r7)  TABLASS(r3, r5, r15)
    TABLASS(r3, r6, r7)
    r15 ← DEC(r8)  inc(r4)  TABLASS(r3, r4, r15)
  ]
]
]
r0 ← TABLVAL(r3, r4)

```

