Chapitre 2

Première tentative de cerner la calculabilité : les fonctions primitives récursives et leur généralisation

Motivation (I)

Calculer = ? ? ? ? ?

- compter ?
- compter les étoiles ?
- calculer la trajectoire d'un objet ?
- déterminer la circonférence d'un cercle ?
- calculer la dimension requise d'une poutre ?
- obtenir le solde d'un compte bancaire ?
- traduire une phrase de l'anglais au français ?
- déterminer si un énoncé en français est vrai ?

Motivation (II)

Calculer comment ?

- à la main ?
- mentalement ?
- à la machine ?
- avec quelle type de machine ?
- permission d'appeler un tout-puissant oracle qui nous souffle des réponses à l'occasion ?

Tout problème peut-il être résolu par un mécanisme ayant suffisamment de ressources ?

Motivation (III)

Calculer quoi ?

Réponse : des fonctions

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

Est-ce que toutes ces fonctions sont "calculables" ?

Dans ce chapitre, nous étudierons :

- les programmes `RÉPÉTER`,
- les programmes `TANTQUE`.

Nous verrons que le premier est déjà puissant, mais trop faible pour cerner entièrement la calculabilité.

Alors, quelles fonctions

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

sont calculables ?

On y réfléchit depuis 1880... mais utilisons notre intuition moderne : prenons un langage de programmation dont on élimine le superflu.

Les programmes `RÉPÉTER`

- registres disponibles $r_0, \ r_1 \ \ldots$
- chaque r_i contient un entier positif ou nul
- les r_i sont initialisés à 0
- l'instruction $r_i \leftarrow r_j$ remplace le contenu du registre r_i par celui de r_j

Les programmes `RÉPÉTER` (suite)

- l'instruction `inc`(r_i) ajoute 1 à r_i ;
- `répéter` r_i `fois` [⟨BLOC⟩] répète l'éxécution d'un bloc d'instructions r_i fois ;
- le nombre d'éxécutions de ⟨BLOC⟩ est fixé une fois pour toutes avant l'entrée dans la boucle, que r_i y soit modifié ou non.

Les programmes RÉPÉTER (fin)

- C'est tout !
- Un programme RÉPÉTER implante une fonction (certes intuitivement "calculable")

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

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

Grammaire pour la syntaxe des programmes RÉPÉTER

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

Quelles sont les fonctions qu'un programme RÉPÉTER peut "calculer" ?

Addition

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

$r_0 \leftarrow r_1$
répéter r_2 fois [
  inc(r_0)
]

Multiplication

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

```
répéter r1 fois [
    répéter r2 fois [
        inc(r0)
    ]
]
```


Exponentiation

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

```
inc(r0)
répéter r2 fois [
    r3 ← r4
    répéter r0 fois [
        répéter r1 fois [
            inc(r3)
        ]
    ]
    r0 ← r3
]
```


Un expéditif syntaxique

- La ligne

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

tient lieu d'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.

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

```
inc(r0)
répéter r2 fois [
    r0 ← MULT(r0, r1)
]
```

Décrémentation

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

```
répéter r1 fois [
    r0 ← r2
    inc(r2)
]
```


Soustraction

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

```
r0 ← r1
répéter r2 fois [
    r0 ← DEC(r0)
]
```


Un autre expéditif syntaxique

- L'instruction

$$r_i \leftarrow k$$

tient lieu de

$$r_i \leftarrow \texttt{MOINS}(r_i, r_i)$$

suivie de 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.

Factorielle

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

```
r0 ← 1
répéter r1 fois [
    inc(r2)
    r0 ← MULT(r0, r2)
]
```

Expéditif pour représenter des variables booléennes

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

Pour évaluer ⟨BLOC⟩ conditionnellement à la valeur booléenne r_i on répète ⟨BLOC⟩ r_i fois.

La ligne

`si` r_i `alors` [⟨BLOC⟩]

signifie

`répéter` r_i `fois` [⟨BLOC⟩].

Et

`ET`(r_1, r_2)

- $r_0 \leftarrow$ `MULT`(r_1, r_2)

Négation

`NEG`(r_1)

- $r_0 \leftarrow$ `MOINS`$(1, r_1)$

Ou

`OU`(r_1, r_2)

- $r_1 \leftarrow$ `NEG`(r_1)
  $r_2 \leftarrow$ `NEG`(r_2)
  $r_0 \leftarrow$ `ET`(r_1, r_2)
  $r_0 \leftarrow$ `NEG`(r_0)

Plus grand que

$\texttt{PG?}(r_1, r_2) = (r_1 > r_2)$

$r_3 \leftarrow \texttt{MOINS}(r_1, r_2)$ {sûrement, et ensuite ?}
`répéter` r_3 `fois` [
  $r_0 \leftarrow$ `vrai`
]

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` [
    `inc`(r_0)
  ]
]

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

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

$r_0 \leftarrow$ `faux` {car 1 n'est pas premier}
$r_5 \leftarrow \texttt{PG?}(r_1, 1)$
`si` r_5 `alors` [
  $r_0 \leftarrow$ `vrai` {supposons premier à moins de trouver un facteur}
  $r_3 \leftarrow 1$
  $r_2 \leftarrow \texttt{MOINS}(r_1, 2)$
  `répéter` r_2 `fois` [
    `inc`(r_3) {essaie chaque diviseur partant de 2}
    $r_4 \leftarrow \texttt{MOD}(r_1, r_3)$
    $r_5 \leftarrow \texttt{PG?}(1, r_4)$
    `si` r_5 `alors` [$r_0 \leftarrow$ `faux`]
  ]
]

Prochain nombre premier

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

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

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

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

Extraction d'un élément d'un tableau

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

$r_3 \leftarrow$ PREMIERK(r_2)
$r_4 \leftarrow r_3$
répéter r_1 fois [
    $r_5 \leftarrow$ MOD(r_1, r_4)
    $r_5 \leftarrow$ PG?$(1, r_5)$ {r_4 divise r_1 ?}
    si r_5 alors [
        inc(r_0)
        $r_4 \leftarrow$ MULT(r_3, r_4)
    ]
]

Affectation 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

Ainsi, les programmes RÉPÉTER peuvent calculer des fonctions complexes.

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

NON ! Il existe des fonctions intuitivement calculables qu'aucun programme RÉPÉTER ne peut calculer ! !

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 :

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

▲

Les fonctions B_i

Définition 2.4. Pour chaque $i \geq 0$, posons

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

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

▲

On remarque que

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

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

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

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

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...

Mais... surprise !

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

Preuve. Pour $i = 0$ on a :

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

et pour $i > 0$ fixé on a :

$\mathtt{B}_i(r_1)$
  `inc`(r_0)
  `répéter` r_1 `fois` [
    $r_0 \leftarrow \mathtt{B}_{i-1}(r_0)$
  ] ▲

Remarque 2.6. Le programme $\mathtt{B}_i$ compte exactement i boucles `répéter` et la profondeur d'imbrication est aussi i. ▲

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

Soit un programme RÉPÉTER P calculant une fonction $f : \mathbb{N}^k \to \mathbb{N}$.

Définition 2.7. $\mathcal{B}(\mathtt{P})$ est le niveau maximal d'imbrications des boucles de P. ▲

Définition 2.8. En fonction des valeurs initiales $r_1, r_2, \ldots, r_k$, on note $\mathcal{M}(\mathtt{P}, r_1, \ldots, r_k)$ la valeur maximale de toute variable de P après l'exécution de P. ▲

Propriétés de la famille des B_i

Remarques 2.9. Les énoncés ci-dessous sont 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 ;
- $2B_i^{\langle k \rangle}(x) \leq B_i^{\langle k+1 \rangle}(x)$ pour $i \geq 1$;
- $B_i^{\langle k \rangle}(x) + x \leq B_i^{\langle k+1 \rangle}(x)$ pour $i \geq 1$.

▲

Théorème 2.10. Pour tout programme RÉPÉTER P calculant une fonction $f : \mathbb{N}^k \to \mathbb{N}$, si $\mathcal{B}(\mathtt{P}) = i$, alors il existe un entier s qui ne dépend que de P, 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. 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 $s = \lceil \frac{c}{2} \rceil + 1$.

Comme P ne contient pas de boucle, on a :

$$\begin{aligned} \mathcal{M}(\mathtt{P}, r_1, \ldots, r_k) &\leq \max(r_1, \ldots, r_k) + c \\ &\leq {B_0}^{\langle \lceil \frac{c}{2} \rceil + 1 \rangle}(\max(r_1, \ldots, r_k)) \\ &= {B_0}^{\langle s \rangle}(\max(r_1, \ldots, r_k)). \end{aligned}$$

Pas d'induction. Soit $i > 0$ et P un programme tel que $\mathcal{B}(\mathtt{P}) = i$.

Hypothèse d'induction. Pour tout programme 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 (suite).
Clairement, P peut être décomposé en la forme suivante où, pour chaque j, $\mathcal{B}(\mathtt{Q}_j) = i - 1$ et $\mathcal{B}(\mathtt{P}_j) \leq i - 1$:

P$(r_1, \ldots, r_k)$
  $\mathtt{P}_1$
  répéter r_{α_1} **fois** $[\mathtt{Q}_1]$
  ⋮
  $\mathtt{P}_m$
  répéter r_{α_m} **fois** $[\mathtt{Q}_m]$

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 :

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

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

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 2.11. 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

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

Définition 2.12.

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

▲

Intuitivement, on voit que A se calcule puisque

- $A(0,x)$ se calcule facilement ;
- $A(1,0) = 1$;
- si $A(1,x)$ se calcule, alors $A(1,x+1) = A(0, A(1,x))$ se calcule aussi ;
- $A(2,0) = 1$;
- si $A(2,x)$ se calcule, alors $A(2,x+1) = A(1, A(2,x))$ se calcule aussi ;
- etc.

Vous sauriez donc écrire un programme en Java qui calculerait la fonction d'Ackermann.

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

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

Lemme 2.13.

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

Preuve. Le lemme sera prouvé par induction d'abord sur i et ensuite sur x.
Base de l'induction externe. Soit $i = 0$. Par définition de A et de B_0, on a :

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

Pas de l'induction externe. Soit $i > 0$.
Hypothèse d'induction externe :

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

Pas de l'induction externe (suite). Montrons que $A(i,x) = B_i(x)$ par induction sur x.

Base de l'induction interne. Soit $x = 0$.

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

Pas de l'induction interne. Soit $x > 0$.
Hypothèse d'induction interne :

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

Pas de l'induction interne (suite).

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

▲

Ackermann n'est pas primitive récursive.

Théorème 2.14. La fonction d'Ackermann A n'est pas calculable par un programme RÉPÉTER.

Preuve. Supposons au contraire $A(y,x)$ calculable par un programme RÉPÉTER, avec profondeur de boucle maximale i.

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

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

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. ▲

Remarque 2.15. La fonction

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

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

Que faut-il inventer pour permettre le calcul de la fonction d'Ackermann ? ? ? ? ?

Les programmes TANTQUE

Comme les programmes RÉPÉTER, sauf que :

- Les boucles RÉPÉTER sont généralisés aux boucles TANTQUE :
  `tantque` $r_i \neq r_j$ `faire` [⟨BLOC⟩] répète l'éxécution de ⟨BLOC⟩ tant que les valeurs des registres r_i et r_j diffèrent ;
- dans une boucle TANTQUE, l'inégalité est réévaluée à chaque itération et les valeurs de r_i et r_j peuvent changer.

Les programmes TANTQUE (suite)

- Un programme TANTQUE P implante une fonction

$$f : \mathbb{N} \times \mathbb{N} \times \cdots \times \mathbb{N} \to \mathbb{N} \cup \{\uparrow\}$$
$$(r_1, r_2, \ldots, r_k) \mapsto \begin{cases} r_0 & \text{si } P \text{ s'arrête,} \\ \uparrow & \text{sinon.} \end{cases}$$

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

Remarque 2.16. Contrairement aux programmes RÉPÉTER, un programme TANTQUE peut ne jamais s'arrêter, par exemple :

$$\texttt{BOUCLE}(r_1) = \uparrow$$

```
inc(r1)
tantque r1 ≠ r0 faire [ ]
```

▲

Remarque 2.17. Tout programme RÉPÉTER peut être simulé par un programme TANTQUE.
Il suffit de remplacer les boucles de la forme

répéter r_i fois [...]

par

$r_k \leftarrow r_i$
tantque $r_j \neq r_k$ faire [
    ...
    inc(r_j)
]

où r_j et r_k sont des registres non utilisés. ▲

Expéditif syntaxique

À la lumière de la remarque 2.17, on se permettra d'utiliser les instructions répéter dans les programmes TANTQUE.

On peut donc recycler tous nos programmes RÉPÉTER en programmes TANTQUE.

La fonction d'Ackermann conquise

Nous exhiberons sommairement un programme TANTQUE qui implante la fonction d'Ackermann telle que présentée à la définition 2.12.

Théorème 2.18. La fonction d'Ackermann A est calculable par un programme TANTQUE.

Esquisse de preuve

Implantation d'une pile réalisée à l'aide des programmes TABLVAL et TABLASS déjà vus.

À l'entrée d'une boucle tantque , les deux premiers éléments au haut de la pile sont les arguments i et x de la définition 2.12.

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 $x > 0$.

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 $x > 0$

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

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

▲

Remarque 2.19. Pour réussir à calculer la fonction d'Ackermann, et bien d'autres fonctions du même acabit, il aura fallu abandonner le confort d'un modèle de calcul qui s'arrête à tout coup sur tous ses arguments ! ▲

Conclusion : notre premier essai

Ainsi les programmes `TANTQUE` sont

- puissants
- plus puissants que les programmes `RÉPÉTER`
- vraisemblablement aussi puissants que nos langages de programmation modernes.

Définition 2.20. Appelons calculables les fonctions qui peuvent être implantées par un programme `TANTQUE`. ▲