Représentation des nombres …
Programmes et langages de programmation

Louis Nolin

Avant-Propos. – Dans cette dernière partie de *l'introduction à la programmation des problèmes scientifiques,* Louis Nolin aborde :

- La représentation des nombres dans les machines usuelles et sa répercussion sur les méthodes de calcul.
- Les programmes et les langages de programmation.

La représentation des nombres dans les machines usuelles est le règne du format fixe ou du format flottant avec ou sans arrondi.

Quelques définitions pour fixer les idées, qui sont données par Adrien Guatto de l'Université de Paris Cité et de l'IRIF (Institut de Recherche en Informatique Fondamentale), reprennent les points forts de l'exposé de la partie programmes et les langages de programmation de Louis Nolin.

Programme et langage.
Un programme est une expression donnée dans un langage artificiel et symbolique fixé appelé langage de programmation.

Exécution.
L'expression symbolise une suite de calculs et d'actions qui peut être réalisée automatiquement, ou exécutée, par un dispositif matériel.

Sémantique.
Un langage de programmation définit l'ensemble des programmes et la suite de calculs et d'actions symbolisées par chacun, sa sémantique.

Pour terminer Louis Nolin vous souhaite bonne chance pour les 5 exercices qui clôturent cette introduction (N.D.L.R.)

V. LA REPRESENTATION DES NOMBRES DANS LES MACHINES USUELLES ET SA REPERCUSSION SUR LES METHODES DE CALCUL

Si on n'utilise pas les machines de Turing en calcul numérique, c'est pour deux raisons: la représentation des nombres qu'elles imposent est peu commode et les opérations qu'elles savent exécuter sont vraiment trop "élémentaires". Songez à l'effort d'imagination qu'il faudrait pour exprimer, dans ces conditions, les algorithmes examinés plus haut!

On serait même tenté d'ajouter à cela une autre raison; à savoir que les machines de Turing ne peuvent traiter que des nombres entiers. Mais là, on aurait tort : toutes les machines actuelles et vraisemblablement aussi toutes les machines futures destinées au calcul numérique ne traitent et ne traiteront jamais que des entiers; comment, en effet, une structure discrète pourrait-elle traiter le continu autrement qu'en le représentant de manière approchée par des nombres décimaux, c'est-à-dire, finalement par des entiers ?

Nous avons dit plus haut que les calculateurs usuels contenaient un certain nombre de registres ou mémoires destinés à recevoir des nombres ou plutôt des représentations des nombres. Le plus souvent, toutes ces mémoires comportent un même nombre de "positions" qui peuvent contenir chacune la représentation d'un chiffre ou d'un signe.

Il y a deux manières de représenter les nombres dans ces mémoires: "en virgule fixe" et en "virgule flottante" (1).

En virgule fixe les nombres sont représentés par une paire ordonnée

$$\langle s , v \rangle$$

où s représente un signe (+ ou −) et v une suite de chiffres dans une base déterminée. Pour fixer les idées, supposons qu'il s'agisse d'une machine décimale dont les mémoires ont 5 positions:

---

(1)termes consacrés par l'usage ...

une pour le signe et 4 pour des chiffres décimaux[1].

Ainsi tous les entiers compris entre -10^4+1 et $+10^4-1$ peuvent être représentés.

Le nombre entier 748 sera figuré, dans une mémoire par la suite

+ 0748

et le nombre entier -3 par la suite

- 0003

On remarquera, au passage, que le nombre 0 a deux représentations

+ 0000 et - 0000

L'addition, la soustraction, la multiplication ne sont définies que si leur résultat peut être représenté dans une mémoire. Ainsi,

748 + (-3) est représenté par + 0745

748 × (-3) est représenté par - 2244

mais

748 + 9643 n'est pas défini; on dit qu'il y a "dépassement de capacité".

La division est toujours définie sauf si le diviseur est 0 ; le signe du résultat est le produit des signes et sa valeur absolue est le quotient entier des valeurs absolues du dividende et du diviseur à une unité près par défaut. Ainsi,

15/4 est représenté par + 0003 .

Il est évident qu'on peut toujours supposer que le contenu d'un registre représente un nombre fractionnaire: il suffira de tenir compte, dans les opérations, de la place supposée de la virgule dans les facteurs pour déterminer l'emplacement de la virgule dans le résultat.

Mais malgré cela cette représentation des nombres a un grave défaut: dans le résultat d'une opération, les chiffres que l'on désire en général conserver sont les chiffres les plus "significatifs", ceux qui sont placés à gauche du nombre, les autres étant souvent douteux sinon dépourvus de signification en

---

(1) Dans les machines décimales actuelles les mémoires comportent, en général, une dizaine de positions pour des chiffres.

raison des erreurs de toutes sortes qui sont commises au cours des calculs.

C'est pourquoi on utilise beaucoup la représentation en virgule flottante. Ici les nombres sont représentés par des quadruplets

$$\langle sm, m, se, e \rangle$$

où sm et se représentent des signes, m et e des suites de chiffres dans une base de numération déterminée; on appelle les paires $\langle sm , m \rangle$ et $\langle se , e \rangle$ la mantisse et l'exposant du nombre considéré.

Pour fixer les idées, on suppose encore qu'on a affaire à une machine décimale et que la mantisse a 4 chiffres et l'exposant un seul.

Le nombre $-148,2$, ou $-0,1482 \times 10^{3}$, est alors représenté par la configuration

$$-1482 + 3$$

le nombre $-0,0073$, ou $-0,0073 \times 10^{0}$, par

$$-0073 + 0 \quad \text{ou} \quad -0073-0$$

le nombre $-0,0073$, ou $-0,7300 \times 10^{-2}$, par

$$-0,7300-2 \ldots$$

Ainsi on peut représenter -souvent de plusieurs manières- des valeurs approchées, avec 4 décimales au plus, de tous les nombres réels dont la valeur absolue est comprise entre $0,0001 \times 10^{-9}$ et $0,9999 \times 10^{9}$.

Lorsque le premier chiffre de la mantisse d'une telle représentation est différent de 0 , on dit que la représentation est normale ou normalisée. Aisni, des deux représentations de $-0,0073$,

$$-0073 + 0 \quad \text{et} \quad -7300-2$$

la seconde seule est normale.

Quant au nombre 0 , il est représenté par toutes les configurations

$$sm\ 0\ 0\ 0\ 0\ se\ e$$

où sm et se sont des signes et où e est l'un des chiffres 0 , 1 , ... 9 .

Les opérations arithmétiques sont effectuées exactement sur les facteurs; le résultat est alors normalisé et, suivant le mode opératoire, soit tronqué, soit arrondi puis tronqué, pour satisfaire aux normes de la représentation.

Addition et soustraction

Exemple 1.

$$a = 0,2034 \times 10^{2}$$

$$b = 0,0613 \times 10^{0}$$

$$a+b = \begin{cases} +0,2034 \times 10^{2} \\ \underline{-0,000613 \times 10^{2}} \\ 0,202787 \times 10^{2} \end{cases}$$

résultat tronqué	résultat arrondi[1]
$0,2027 \times 10^{2}$	$0,2028 \times 10^{2}$

Exemple 2.

$$a = -0,2034 \times 10^{1}$$

$$b = -0,0201 \times 10^{2}$$

$$a-b = \begin{cases} -0,2034 \times 10^{1} \\ \underline{+0,2010 \times 10^{1}} \\ -0,0024 \times 10^{1} \end{cases}$$

normalisation $-0,2400 \times 10^{-1}$

résultat tronqué	résultat arrondi
$0,2400 \times 10^{-1}$	$0,2400 \times 10^{-1}$

Multiplication

Exemple 1.

$$a = -0,0650 \times 10^{-2}$$

$$b = +0,0201 \times 10^{3}$$

signe du résultat : - (produit des signes)

valeur absolue du résultat :

$$\begin{array}{r} 0,0650 \times 10^{-2} \\ \times\ 0,0201 \times 10^{3} \\ \hline 650 \\ 000 \\ 1300 \\ \hline 0,00130650 \times 10^{-2+3} \end{array}$$

(1) L'arrondi s'obtient en ajoutant ou en retranchant 0,00005 à la mantisse normalisée, suivant que le résultat est positif ou négatif.

normalisation : $0,130650 \times 10^{-1}$

résultat tronqué	résultat arrondi
$-0,1306 \times 10^{-1}$	$-0,1307 \times 10^{-1}$

Exemple 2.

$a = 0,3331 \times 10^{1}$

$b = 0,3002 \times 10^{2}$

signe du résultat : +

valeur absolue du résultat

$$\begin{array}{r} 0,3331 \times 10^{1} \\ \times\ 0,3002 \times 10^{2} \\ \hline 6662 \\ 9993 \\ \hline 0,09999662 \times 10^{1+2} \end{array}$$

normalisation : $0,9999662 \times 10^{2}$

arrondi

$1,000162 \times 10^{2}$

normalisation et troncature

$0,1000 \times 10^{3}$

résultat tronqué	résultat arrondi
$0,9999 \times 10^{2}$	$0,1000 \times 10^{3}$

Division

$a = -\ 0,0613 \times 10^{-3}$

$b = \ \ 0,2010 \times 10^{2}$

signe du résultat : − (produit des signes)

exposant : $-3-(+2) = -5$

valeur absolue de la mantisse

$$\begin{array}{l} 0,06130 \\ \quad 010000 \\ \quad\quad 19600 \\ \quad\quad\ 15100 \\ \quad\quad\quad 1030 \end{array} \qquad \begin{array}{r} 0,2010 \\ \hline 0,030497 \end{array}$$

5 chiffres

normalisation : $-\ 0,30497 \times 10^{-6}$

résultat tronqué	résultat arrondi
$-0,3049 \times 10^{-6}$	$-0,3050 \times 10^{-6}$

Bien entendu, la représentation des nombres en virgule flottante ne met pas à l'abri des dépassements de capacité : elle les rend simplement moins fréquents.

Les opérations qu'on vient de décrire ne possèdent pas, on s'en doute, toutes les propriétés des opérations mathématiques qu'elles représentent. En particulier, l'addition et la soustraction, si elles restent évidemment commutatives, ne sont plus associatives en général.

En virgule fixe avec 4 chiffres et un signe.

Soit $a = +7000$, $b = +4000$, $c = -2000$;

$a+b$ n'est pas défini (dépassement de capacité), donc $(a+b)+c$ non plus; par contre

$$b+c = +2000 \quad \text{et} \quad a+(b+c) = +9000$$

En virgule flottante avec 4 chiffres de mantisse et un chiffre d'exposant.

Soit $a = +0,7000 \times 10^{9}$, $b = +0,4000 \times 10^{9}$, $c = -0,2000 \times 10^{9}$;

comme dans l'exemple précédent, $(a+b)+c$ n'est pas défini (dépassement de capacité), alors que $a+(b+c) = +0,9000 \times 10^{9}$.

Soit $a = +0,9000 \times 10^{7}$, $b = +0,9000 \times 10^{4}$, $c = +0,9000 \times 10^{-2}$;
$(a \times b) \times c$ n'est pas défini (dépassement de capacité), alors que $a \times (b \times c) = +0,7290 \times 10^{9}$.

Mais la non-associativité des opérations a une autre raison, plus cachée.

Soit $a = +0,2000 \times 10^{2}$, $b = +0,8005 \times 10^{2}$, $c = +0,9009 \times 10^{2}$;
En opérant sans arrondi, on obtient :

$a + b = +1,0005 \times 10^{2}$	, tronqué à	$+ 0,1000 \times 10^{3}$
$(a+b)+ c = +0,19009 \times 10^{3}$	, tronqué à	$+ 0,1900 \times 10^{3}$
$b + c = +1,7014 \times 10^{2}$	, tronqué à	$+ 0,1701 \times 10^{3}$
$a +(b+c) = +0,19010 \times 10^{3}$	, tronqué à	$+ 0,1901 \times 10^{3}$

En opérant avec arrondi, on obtient :

$a + b = +1,0005 \times 10^2$, arrondi à $+ 0,1001 \times 10^3$

$(a+b)+c = +0,19019 \times 10^3$, arrondi à $+ 0,1902 \times 10^3$

$b + c = +1,7014 \times 10^2$, arrondi à $+ 0,1701 \times 10^3$

$a+(b+c) = +0,19010 \times 10^3$, arrondi à $+ 0,1901 \times 10^3$

On trouve sans peine des exemples où $(a \times b) \times c$ n'est pas égal à $a \times (b \times c)$.

Remarquons encore que $(a+a)+a$ n'est pas toujours identique à $a \times 3$:

Soit en effet $a = +0,5008 \times 10^1$, $b = +0,3000 \times 10^1$;

En opérant sans arrondi, on obtient :

$a + a = +1,0016 \times 10^1$, tronqué à $+0,1001 \times 10^2$

$(a+a)+a = +0,15018 \times 10^2$, tronqué à $+0,1501 \times 10^2$

$a \times b = +0,15024 \times 10^2$, tronqué à $+0,1502 \times 10^2$

En opérant avec arrondi, on obtient :

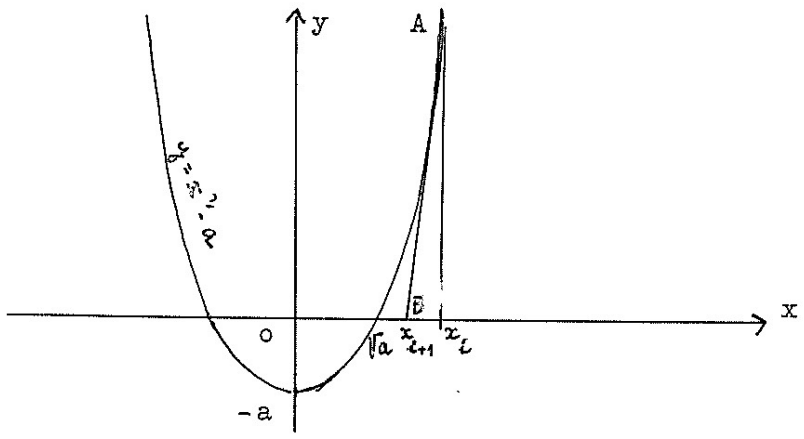
$a + a = +1,0016 \times 10^1$, arrondi à $+0,1002 \times 10^2$

$(a+a)+a = +0,15028 \times 10^2$, arrondi à $+0,1503 \times 10^2$

$a \times b = +0,15024000 \times 10^2$, arrondi à $+0,1502 \times 10^2$

Si j'ai insisté sur ces anomalies, ce n'est pas pour vous faire perdre confiance dans la légitimité du calcul automatique, mais pour vous mettre en garde contre certaines erreurs trop faciles à commettre, erreurs que l'exemple suivant va illustrer.

Considérons la figure



où A B est la tangente en A à la parabole. On a

$$(x_i - x_{i+1})\ 2x_i = x_i^2 - a$$

d'où

$$x_{i+1} = ((x_i^2 + a)/x_i)/2 \qquad (1)$$

ou

$$x_{i+1} = (x_i + (a/x_i))/2 \qquad (2)$$

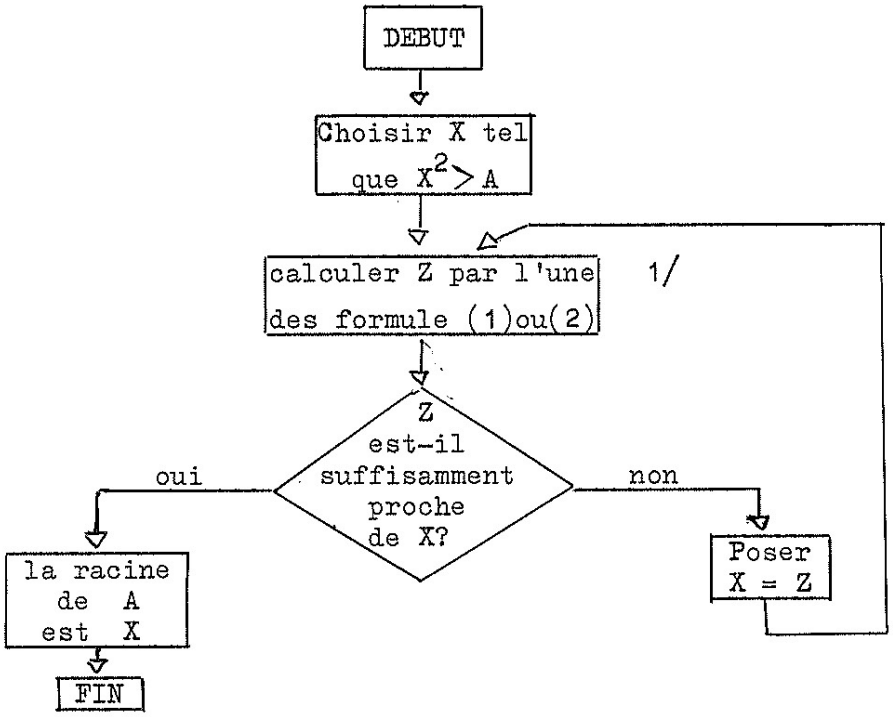
On démontre facilement que, si x_i est $> \sqrt{a}$, on a

$$\sqrt{a} < x_{i+1} < (x_i + \sqrt{a})/2$$

d'où l'on déduit que si x_0 est $> \sqrt{a}$ et si x_{i+1} et x_i sont liés par l'une des relations (1) ou (2) , la suite

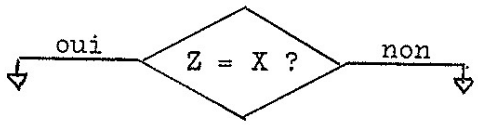
$$x_0 , x_1 , x_2 , \cdots$$

converge vers $\sqrt{a}$ par valeurs supérieures. On tire de là une méthode pour le calcul de la racine carrée, méthode qu'on attribue d'ordinaire à Newton bien qu'elle fût connue d'Archimède, comme je l'ai appris depuis peu :



1/ (1) : Z = ((X² + A)/X)/2
(2) : Z = (X + (A/X))/2

A première vue le succès de l'entreprise est indépendant de la valeur initiale donnée à X , pourvu que X^2 soit supérieur à A , et aucune raison impérieuse n'oblige à préférer la formule (2) à la formule (1) si ce n'est qu'elle comporte une multiplication de moins que cette dernière. Reste le test qu'il faut bien préciser si on veut faire de cette méthode un programme destiné à un calculateur automatique. Comme on opère, en fin de compte, sur des entiers, on peut penser que la différence entre Z et X doit finalement être nulle. Il semble donc naturel d'utiliser le test suivant :



La plupart du temps les choses se passent fort bien. Malheureusement il y a quelques exceptions.

En virgule fixe,

Soit $A = 15$

Partons de $X = \underline{4}$

Utilisons la formule (2) ; il vient :

$$Z = (4 + (15/4))/2 = (4 + 3)/2 = \underline{3} .^{(1)}$$

Comme Z est $\neq X$, posons

$$X = Z = \underline{3}$$

et calculons :

$$Z = (3 + (15/3))/2 = (3 + 5)/2 = \underline{4} .$$

Inutile d'aller plus loin : la condition $X = Z$ ne sera jamais satisfaite.(2)

Mais peut être aurons-nous plus de chances en virgule flottante sans arrondi .

Hélas !

Soit $A = + 0{,}9006 \times 10^1$

Partons de

---

(1) Rappelons que x/y désigne le quotient entier, à une unité près , de x par y .

(2) Euler qui fut, entre autres, un bon calculateur, n'hésitait pas à appeler convergente une suite telle que

$$4 , 3 , 4 , 3 \ldots$$

et lui attribuait pour valeur limite $3{,}5$.

$$X = \underline{+0,3001 \times 10^{1}}$$
$$A/X = +0,30009 \times 10^{1} \text{ , tronqué à } +0,3000 \times 10^{1}$$
$$X+(A/X) = +0,6001 \times 10^{1} \text{ , tronqué à } +0,6001 \times 10^{1}$$
$$Z = +0,30005 \times 10^{1} \text{ , tronqué à } \underline{+0,3000 \times 10^{1}}$$

Comme Z est $\neq$ X , posons

$$X = Z = \underline{+0,3000 \times 10^{1}}$$

et calculons

$$A/X = +0,30020 \times 10^{1} \text{ , tronqué à } +0,3002 \times 10^{1}$$
$$Y+(A/X) = +0,6002 \times 10^{1} \text{ , tronqué à } +0,60002 \times 10^{1}$$
$$Z = \underline{+0,3001 \times 10^{1}}$$

Nous n'avons pas été plus heureux.

Que va-t-il se passer

en virgule flottante avec arrondi

si on choisit - pourquoi pas - la formule (1)

Soit encore $A = +0,9006 \times 10^{1}$

Partons de

$$X = \underline{+0,3001 \times 10^{1}}$$

et calculons

$$X^2 = +0,09006001 \times 10^{2}, \text{ arrondi à } +0,9006 \times 10^{1}$$
$$X^2+A = +1,8012 \times 10^{1} \text{ , arrondi à } +0,1801 \times 10^{2}$$
$$(X^2+A)/X = 6,0013 \times 10^{0} \text{ , arrondi à } +0,6001 \times 10^{1}$$
$$Z = 0,30005 \times 10^{1} \text{ , arrondi à } \underline{+0,3001 \times 10^{1}}$$

Voilà qui est plus satisfaisant, certes, mais si nous partons de

$$\underline{X = +0,3002 \times 10^{1}}$$

nous avons successivement

$$X^2 = +0,09012004 \times 10^{2}, \text{ arrondi à } +0,9012 \times 10^{1}$$
$$X^2+A = +1,8018 \times 10^{1} \text{ , arrondi à } +0,1802 \times 10^{2}$$
$$(X^2+A)/X = +6,0026 \times 10^{0} \text{ , arrondi à } +0,6003 \times 10^{1}$$
$$Z = +0,30015 \times 10^{1} \text{ , arrondi à } \underline{+0,3002 \times 10^{1}}$$

Voici donc deux valeurs différentes qui satisfont séparément à la condition imposée ,

$$+0,3001 \times 10^{1} \quad \text{et} \quad +0,3002 \times 10^{2}$$

Ainsi la valeur approchée de la racine carrée de A dépend de la première valeur donnée à X !

Des gens dont on ne saurait soupçonner la bonne foi vous diront peut-être un jour que parmi les artisans du succès d'un calcul numérique il faut soigneusement distinguer mathématiciens et programmeurs. De telles classifications me plongent dans une grande perplexité. Car, d'une part, si le mathématicien s'estime satisfait lorsqu'il a fourni l'organigramme de la méthode de NEWTON (ARCHIMEDE ?) tel que nous l'avons tracé plus haut, il faut bien avouer qu'il reste encore au programmeur à faire toute une étude qui, si elle relève d'un domaine bien humble des mathématiques, n'en est pas moins une activité de mathématicien. Et si le mathématicien veut bien chercher des réponses à toutes les questions que nous venons de poser, alors que reste-t-il à faire au programmeur, en calcul numérique tout au moins ?

EXERCICE V.1

Trouver d'autres exemples des phénomènes précédents dans le cas d'un calcul en virgule flottante avec une mantisse de 4 chiffres.
Même question avec une mantisse de 5 chiffres.

EXERCICE V.2

Montrer que si le calcul approché de $\sqrt{A}$ est effectué en virgule fixe, en utilisant la formule (2) , avec une valeur initiale de X telle que X^2 soit $> A$, on peut s'arrêter lorsque Z est $\geqslant X$.
X est alors une valeur approchée de $\sqrt{A}$ à une unité près par défaut.

EXERCICE V.3

Traiter quelques exemples d'opérations sur des nombres exprimés en virgule flottante lorsque la base de numération est 2 ou 8 .

VI - PROGRAMMES ET LANGAGES DE PROGRAMMATION

Qu'un programme soit l'expression d'un algorithme destinée à une machine; que, pour une machine déterminée, il ne puisse mentionner qu'un nombre limité d'opérations élémentaires sur des représentations conventionnelles des nombres: cela concerne sa matière, son contenu, ou si l'on veut, sa signification.

Qu'il se présente comme un organigramme ou comme une suite d'instructions, cela concerne son aspect ou sa forme.

C'est ce dernier point que nous allons examiner de plus près pour clore cette introduction à la programmation.

L'organigramme met en évidence la structure d'un algorithme. Cela fait bien l'affaire d'un spectateur intelligent qui veut avant tout comprendre, et accessoirement exécuter. Mais ce qu'on attend d'une machine, c'est simplement qu'elle exécute pas à pas et à la lettre les indications qu'on lui fournit. C'est pourquoi un programme - au sens strict du terme - a l'apparence d'une suite linéaire d'instructions; et une instruction est une phrase, donc un élément d'un langage.

Un tel langage doit être à la fois assez riche pour exprimer tous les algorithmes d'un certain type, et assez pauvre pour être facilement interprété par une machine.

Ces deux exigences écartent d'emblée tous les langages "usuels". En effet on trouve ici, tout d'abord, un vocabulaire beaucoup trop riche. Qu'on puisse dire, indifférement, "ajouter b à a", "additionner a et b", "calculer la somme de a et de b", "calculer a+ b", "cumuler a et b", que sais-je encore, cela ne peut que compliquer la tâche d'interprétation de la machine; et ces difficultés ne sont rien à côté de celles que crée une syntaxe

libérale qui permet des suppressions, des contractions, des inversions et tous ces tours subtils qui font le beau langage: le langage usuel est donc trop complexe pour aspirer aux fonctions de langage de programmation. Et, ce qui peut paraître surprenant, il est aussi trop pauvre, à certains égards. Voyons cela.

Dans une lettre à Frénicle datée de 1640[1], Fermat communique à son correspondant la proposition que voici:

"Si un nombre est mesuré[2] par un autre, et que le nombre divisé soit encore divisé par un autre nombre moindre que le premier diviseur, en ce cas, si vous ôtez du quotient de la seconde division, multiplié par la différence des deux diviseurs, le reste de la seconde division, ce qui restera sera mesuré par le premier diviseur."

Exprimons cela dans un langage plus familier:

"Si un nombre x est mesuré par un autre nombre y, et que x soit encore divisé par un autre nombre z, moindre que y, en ce cas, si vous ôtez du quotient q de x par z, multiplié par la différence de y et de z, le reste r de la division de x par z, ce qui restera sera divisé exactement par y."

De façon plus concise encore:

"Si x est divisible par y,
si z est $< y$,
si $q = \left[\frac{x}{z}\right]$ [3],
si $r = x - q \times z$, $(x < z)$,
alors $q \times (y - z) - r$ est divisible par y".

Ce dernier langage n'est pas seulement "plus moderne" que celui de Fermat, il est d'une toute autre nature.

Son originalité se manifeste déjà au niveau du vocabulaire par l'emploi de ces pronoms à l'état pur que sont les variables "x", "y", "z" ... Mais surtout, ce langage est un langage écrit. Qu'il

---

1/ Oeuvres de Fermat, publiées par P.Tannery et Ch.Henry, Paris, Gauthier-Villars, 1894, p 206-212.

2/ C'est-à-dire, divisible.

3/ C'est-à-dire le quotient entier de x par z, à une unité près, par défaut.

permette donc d'employer des signes tels que " < " ou " [-] " pour "moindre que" ou "quotient", voilà qui est déjà avantageux. Mais l'essentiel n'est pas là; il est dans la syntaxe qui, affranchie des restrictions qu'impose le langage parlé, permet de donner aux phrases une structure simple et de les combiner suivant des règles logiques bien définies. C'est pourquoi ce langage est particulièrement apte à exprimer des démonstrations.

Celle de l'énoncé de Fermat tient en quatre lignes:

$$q \times (y - z) - r = q \times y - (q \times z + r) ,$$

donc

$$q \times (y - z) - r = q \times y - x ,$$

et $q \times y - x$ est divisible par y .

Il ne faut pas chercher ailleurs les raisons de "la fécondité de la méthode algébrique" , sujet de tant de dissertations philosophico - mathématiques. Fermat a énoncé sa proposition tout de suite après un de ses théorèmes les plus importants[1/] et, de son aveu même, il avait trouvé là une occasion d'embarrasser son correspondant; il est clair, maintenant, que la difficulté de cette proposition n'était pas d'ordre mathématique mais seulement linguistique.

Les langages destinés à exprimer des programmes - les langages de programmation- sont inspirés du langage de l'algèbre. En particulier ils font usage, en même temps que de constantes qui sont les noms d'objets bien définis , de variables qui permettent de faire allusion à des objets qu'on appelle les valeurs de ces variables et dont on sait seulement qu'ils sont les éléments d'un certain ensemble.

Mais ces deux emplois des variables diffèrent notablement. Dans une même phrase, voire un même paragraphe, si ce n'est un chapitre entier d'un ouvrage d'algèbre, une variable conserve la même valeur car elle fait allusion à un objet, toujours le même, bien que non précisé. Dans un programme, au contraire, la valeur d'une variable

---

1/ Tout nombre premier p divise soit a soit $a^{p-1} - 1$.

peut changer au cours de l'exécution de ce programme ! C'est donc plutôt aux "points mobiles" de la mécanique qu'il faut songer, à ces points dont la position varie avec le temps.

Cette différence apparaît bien dans les deux programmes suivants

Début	Début
Poser i = 0;	
Calculer $x_0 = (1 + a)/2$	Calculer y = (1 + a)/2 ;
1: Faire i = i + 1 ;	1: Poser x = y
Calculer $x_i = (x_i + a/x_i)/2$	Calculer y = (x + a/x)/2
Si $x_i < x_{i-1}$, aller à 1 ;	si y < x , aller à 1 ;
(sinon) Poser rac = x_{i-1} ;	(sinon) Poser rac = x ;
Fin	Fin

Ces deux programmes sont rigoureusement équivalents en ce sens qu'à partir d'une même valeur de a ils permettent d'obtenir, avec la même machine, la même valeur de rac. Mais le premier est rédigé à la manière d'une page d'algèbre où l'on donne des désignations distinctes à des quantités qui risquent d'être différentes. Il en résulte qu'à tout moment la machine garde "en mémoire" la trace de résultats intermédiaires qui ne présentent guère d'intérêt.

En effet, à chaque variable utilisée dans un programme - "x" par exemple - la machine associe une mémoire. Le contenu de cette mémoire représente la valeur de la variable "x" , en général un nombre; et cette valeur change dès que la machine exécute une instruction telle que

Poser x = 3,14

ou

Faire x = x + 1 [1/]

. . .

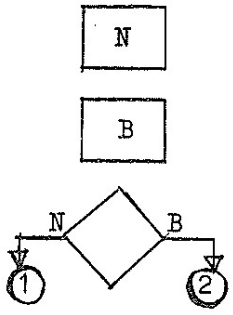
Nous avons vu [2/] comment l'emploi d'une variable permet de simplifier l'exposé d'un algorithme destiné à une machine de Turing. Une seule variable "e" suffit d'ailleurs dans ce cas, variable dont

---

1/ C'est-à-dire, rappelons-le, ajouter 1 à la valeur actuelle de x et attribuer à x cette nouvelle valeur; autrement dit: augmenter de 1 la valeur de x .

2/ Page 30

la valeur est le contenu de la case explorée au moment où l'on en parle. Ainsi les instructions correspondant à certaines des opérations élémentaires peuvent s'énoncer comme il suit:



Poser e = N [1]

Poser e = B

Si e = N aller à 1 , sinon aller à 2

ou: Si e = B aller à 2 , sinon aller à 1

Mais, si nous voulons pouvoir décrire une configuration [2] , ce langage est trop pauvre: il nous faut recouvrir à l'emploi de variables indicées

$$a_{-3} \ , \ a_0 \ , \ a_1 \ , \ a_{17} \ , \ a_i \ , \ a_{i+4} \ , \cdots$$

artifice bien connu pour noter le résultat de l'application de l'ensemble des entiers < 0 , $= 0$ ou > 0 dans un certain ensemble, ici $\{$ "▒" ; "□" $\}$.

Une configuration est alors déterminée par la valeur actuelle de l'indice i de la variable "a_i" qui désigne le contenu de la case explorée et par les valeurs des variables

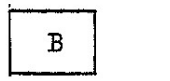
$$\cdots, \ a_{i-2} \ , \ a_{i-1} \ , \ a_i \ , \ a_{i+1} \ , \ a_{i+2} \ , \cdots$$

Les instructions correspondant aux opérations élémentaires peuvent alors s'exprimer comme il suit:

1/ N et B sont des constantes, dans ce langage, c'est-à-dire les noms d'objet bien définis, "▒" et "□".

2/ Cela n'est pas indispensable, on vient de le voir, lorsqu'on se propose seulement de faire exécuter à la machine certaines instructions; il en est autrement si nous voulons "parler" de cette exécution: il faut alors un langage de niveau supérieur ("méta-langage")

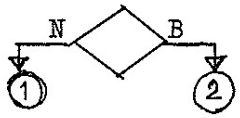
Poser $a_i = N$



Poser $a_i = B$

Faire $i = i+1$

Faire $i = i-1$



Si $a_i = N$ aller à 1 , sinon aller à 2

ou: Si $a_i = B$ aller à 2 , sinon aller à 1

En ajoutant au vocabulaire de chacun de ces deux langages les mots suivants, on pourra donc exprimer, sous forme de programmes, les algorithmes exposés au chapitre IV et tous les algorithmes analogues [1]:

;	pour séparer deux instructions consécutives.
les suites de chiffres	pour désigner les instructions auxquelles renvoient des instructions telles que "Si $a_i = N$ aller à m , sinon aller à n ".
:	pour séparer ces désignations, ces étiquettes, de l'instruction qu'elles désignent.
DEBUT FIN	pour encadrer, en quelque sorte, les programmes.

Ceci nous donne un aperçu de ce qu'est un langage de programmation. Et nous voyons dès maintenant que sa description comporte deux parties logiquement distinctes.

D'une part, la syntaxe, qui décrit avec précision les mots et les phrases du langage;

D'autre part, la sémantique, qui indique la signification des mots et des phrases dont l'emploi est permis.

---

[1] Voir pages 28 à 31

C'est là une distinction fondamentale qui commence à peine à entrer dans l'usage: pendant longtemps la description des langages de programmation n'a été qu'un mélange affreux de syntaxe et de sémantique qui eut été risible s'il n'avait compliqué, comme à plaisir, la tâche des programmeurs. L'étude de quelques langages particuliers vous convaincra, je l'espère.

L'écriture d'un programme destiné à une machine déterminée est donc soumise à deux contraintes: la première est la nécessité de respecter strictement la syntaxe du langage de programmation qu'on emploie; la seconde est l'obligation de tenir compte de la manière dont les objets - en particulier des nombres - sont représentés dans la machine et la nature des opérations que celle-ci peut effectuer sur ces représentations.

On peut se demander à quel degré ces contrainte sont liées, c'est-à-dire quelle est la nature de la relation qui existe - à n'en pas douter - entre le langage de programmation et les machines.

Une machine déterminée ne peut comprendre, directement, qu'un seul langage, celui qui est décrit dans son mode d'emploi. Un tel "langage-machine" est, en substance, tout à fait analogue aux langages dont nous avons parlé; il n'en diffère que par son aspect extérieur, car tous les mots, toutes les phrases d'un tel langage, ne sont rien d'autre que des suites de chiffres [1]. Mais une machine peut parfaitement déchiffrer d'autres langages et le traduire dans le sien propre: il lui suffit pour cela d'exécuter un programme approprié. Comme c'est là un sujet d'étonnement perpétuel et aussi un bon exemple de traitement d'un problème scientifique "non numérique" [2] nous allons en donner un exemple.

---

[1] C'est ce qui permet d'enregistrer les programmes écrits dans ce langage de la même manière qu'on enregistre des nombres.

[2] Nous entendons par là un problème où l'information à traiter est, par nature, non numérique; bien sûr, on peut "arithmétiser" à peu près n'importe quoi et cela rend parfois de grands services. De même lorsqu'on étudie les propriétés du cône, il est avantageux de le considérer comme la limite d'une pyramide; mais quand on veut faire un pied de table tronconique on se sert d'un tour à bois et non d'une machine à chanfreiner.

Considérons deux langages L_1 et L_2 .

Syn Dans chacun d'eux les variables sont

$$A \quad B \quad . \; . \; . \quad Z \quad a_1 \quad a_2 \quad a_3 \quad . \; . \; .$$

et les constantes fondamentales sont les suites finies non vides de chiffres décimaux séparées éventuellement en deux sous-suites non vides par une virgule (exemples: 40 3,14) 1/.

Sém Une constante fondamentale est le nom propre d'un nombre décimal $\geqslant 0$ (un même nombre peut avoir plusieurs noms, comme par exemple 013 et 13); une variable fait allusion à un nombre décimal < 0, $= 0$, ou > 0 , non précisé.

Pour désigner des nombres ou les résultats des opérations effectuées sur des nombres, on dispose, dans L_1 et dans L_2, d'expressions dont la signification est immédiate. Elles sont définies comme suit :

Syn Dans L_1

- les variables et les constantes fondamentales sont des expressions;
- si x est une variable ou une constante fondamentale, alors +x 2/ et -x sont des expressions (exemples: -B +3,14);
- si x et y sont des variables alors

$$x + y \quad x - y \quad x \times y \quad x \,/\, y$$

sont des expressions.

Il n'y a pas d'autres expressions dans L_1 , mais le vocabulaire de L_2 est beaucoup plus riche.

- les variables et les constantes fondamentales sont des expressions simples;
- si x est une expression, (x) 3/ est une expression simple;
- il n'y a pas d'autres expressions simples;
- si x et y sont des expressions simples alors

1/ On laisse de côté les questions de limitation du nombre des variables ou de la "longueur" des constantes.

2/ C'est-à-dire "+" suivi de x .

3/ C'est-à-dire "(" suivi de x suivi de ")" .

x +x -x

x + y x - y x × y x / y

dont des expressions;

- il n'y a pas, dans L_2 , d'autres expressions.[2/]

Exemples

les expressions de L_1 , (A + B) - (3,14 × (C - D)), sont des expressions; A + B - (3,14 × (C - D)) n'en est pas une.

Syn Dans chacun des langages L_1 et L_2 , si x est une variable et y une expression (de ce langage), alors x := y est une instruction d'attribution.

Sém x := y signifie: poser x = y ,
c'est-à-dire: calculer (éventuellement) la valeur de y et attribuer cette valeur à x .

Syn Un programme, dans L_1 et dans L_2 , est une suite finie d'instructions séparées par le symbole ";" .

Bien entendu, L_1 et L_2 contiennent d'autres instructions qui n'ont pas d'intérêt pour notre problème. Car ce que nous nous proposons, c'est de traduire les instructions d'attribution de L_2 dans le langage L_1 , qui est très proche d'un langage-machine; et nous prendrons comme exemple l'instruction

D := (A + B) - (3,14 × (C - D))

Pour ce faire, il nous suffit de disposer d'une machine de Turing "généralisée". Une telle machine est tout-à-fait analogue à celle que nous avons décrite au chapitre IV.

Elle contient plusieurs bandes dont chacune est explorée par une tête de lecture ou d'écriture. Chaque case de l'une de ces bandes peut contenir à volonté l'un des symboles suivants:

une variable, un chiffre décimal, l'un des signes:

, ; := + - × / ( )

et d'autres signes encore qui sont sans rapport avec notre problème. Signalons pourtant que certaines bandes peuvent contenir une suite de chiffres décimaux quelconque. Nous préciserons cela plus loin.

2/ On voit que la classe des expressions et celle des expressions simples (qui est une sous-classe de la première) sont définies non pas séparément mais simultanément et de manière récurrente. Elles sont pourtant parfaitement définies en ce sens qu'on peut toujours décider, de façon effective, si une suite de variables, de constantes et de signes opératoires est ou n'est pas une expression (ou une expression simple).

e_1 e_2 ..... e_m

désignent le contenu de la case explorée, à un moment donné, dans la 1ère bande, la 2ème bande, ... , la m-ième bande.

$e_i := e_j$ - où i et j sont 1, 2, ..., ou m -

signifie: inscrire dans la case explorée de la i-ième bande le contenu de la case explorée de la j-ième bande.

e_i := "s" - où i est 1, 2, ... ou m, et où s est l'un des symboles énumérés ci-dessus -

signifie: inscrire dans la case explorée de la i-ième bande le symbole s .

1 → 2 → ..... m →

signifient: explorer la case située immédiatement à la droite de l case explorée dans la 1ère, la 2ème, ... , la m-ième bande.

1← 2← ..... m←

signifient: explorer la case située immédiatement à la gauche de l case explorée dans la 1ère, la 2ème, ... , la m-ième bande.

Enfin la machine sait reconnaître si le contenu d'une case est ou n'est pas:

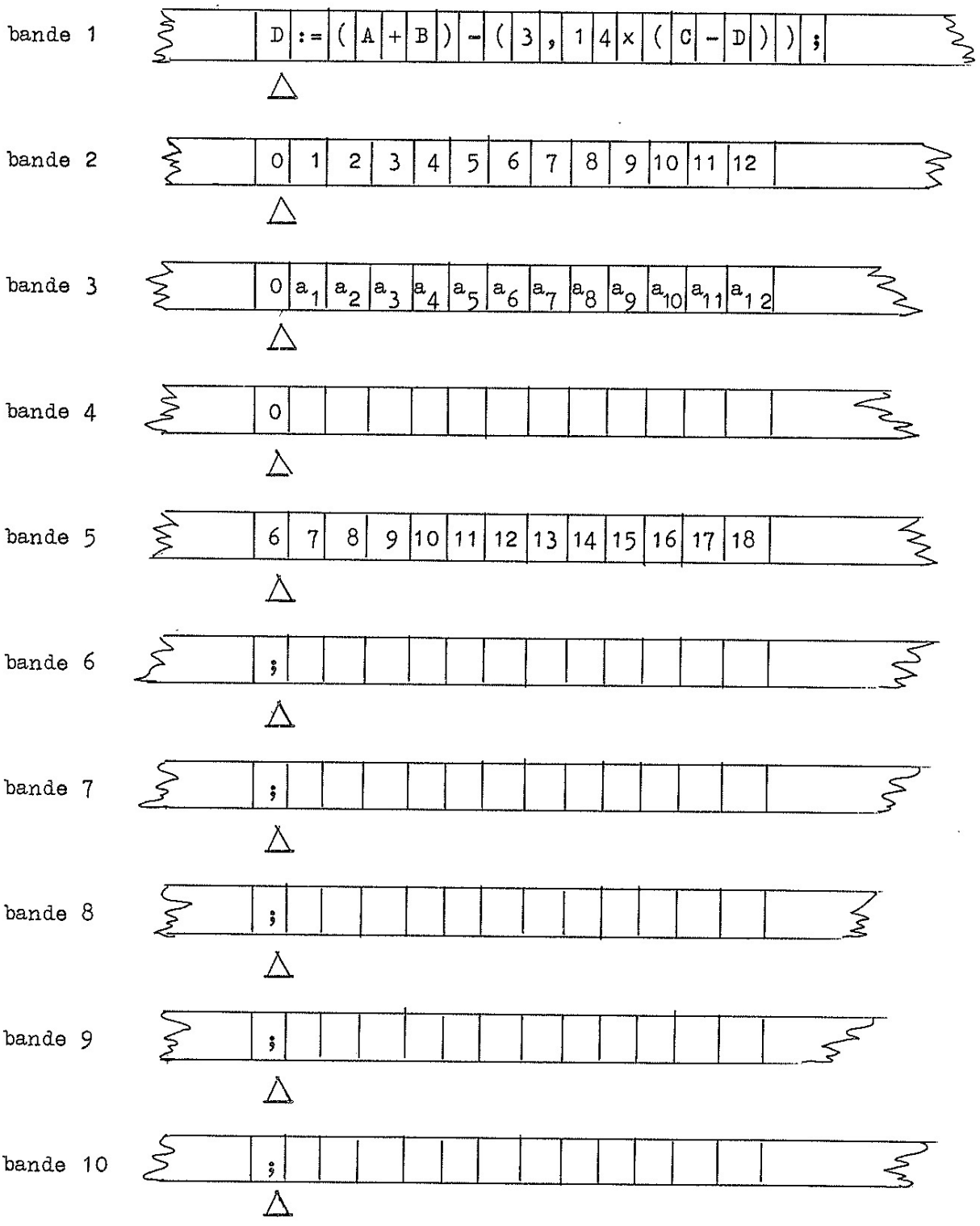
une variable, un chiffre ou une virgule, un signe opératoire (+ , - , × , /) , le signe := , le signe ; la parenthèse ouvrante, la parenthèse fermante, etc ..

Comme vous vous en êtes rendu compte, je viens de décrire sommairement un langage de programmation pour une machine de Turing général ce langage, je l'appellerai L_3 .

Notre problème est donc le suivant: traduire dans le langage L_1 à l'aide d'une machine interprétant un programme écrit en L_3 , l'instruction de L_2 :

D := (A + B) - (3,14 × (C - D))

On part de la configuration initiale:



A tout moment:

- la valeur de e_1 est un des symboles qui composent l'instruction à traduire;
- la valeur de e_2 représente le nombre d'instructions complètes de L_1 déjà obtenues;
- la valeur de e_3 est la dernière variable auxiliaire utilisée;
- la valeur de e_4 est "1" depuis le moment où la case explorée dans la bande 1 contient un chiffre, jusqu'au moment où la valeur de e_1 devient un symbole autre qu'un chiffre ou une virgule; dans tous les autres cas, la valeur de e_4 est "0". Ces changements de valeur jouent donc le même rôle que l'ouverture et la fermeture des parenthèses;
- la valeur de e_5 est une suite de chiffres qui représentent un entier, considéré comme la valeur de l'indice m ; cet indice est utilisé comme numéro de bande.

La bande 6 et les bandes suivantes sont utilisées pour composer, au fur et à mesure, les instructions de L_1 qui traduiront l'instruction contenue dans la bande 1 .

Posons, pour abréger, les définitions suivantes:

Pr 1

$e_m := e_1$; m → ; 1 → [1/]

Pr 2

3 → ; $e_m := e_3$; m → ; e_m := "0" ;
1 : 5 → ; si $e_m \neq$ ";" aller à 1 ;
(sinon) m → ; $e_m := e_3$; m → ; e_m := ":=" ; m →

Pr 3

e_m := ";" ; 2 → ; m → ; $e_m := e_2$

Pr 4

2 : 5 ← ; si $e_m \neq$ "0" aller à 2

[1/] c'est-à-dire:
donner à e_m la valeur de e_1 (inscrire dans la case explorée de la bande m le contenu de la case explorée de la bande 1); puis regarder, dans la bande m , le contenu de la case située à droite de la case explorée présentement; puis regarder dans la bande 1 , le contenu de la case située à droite de la case explorée présentement.

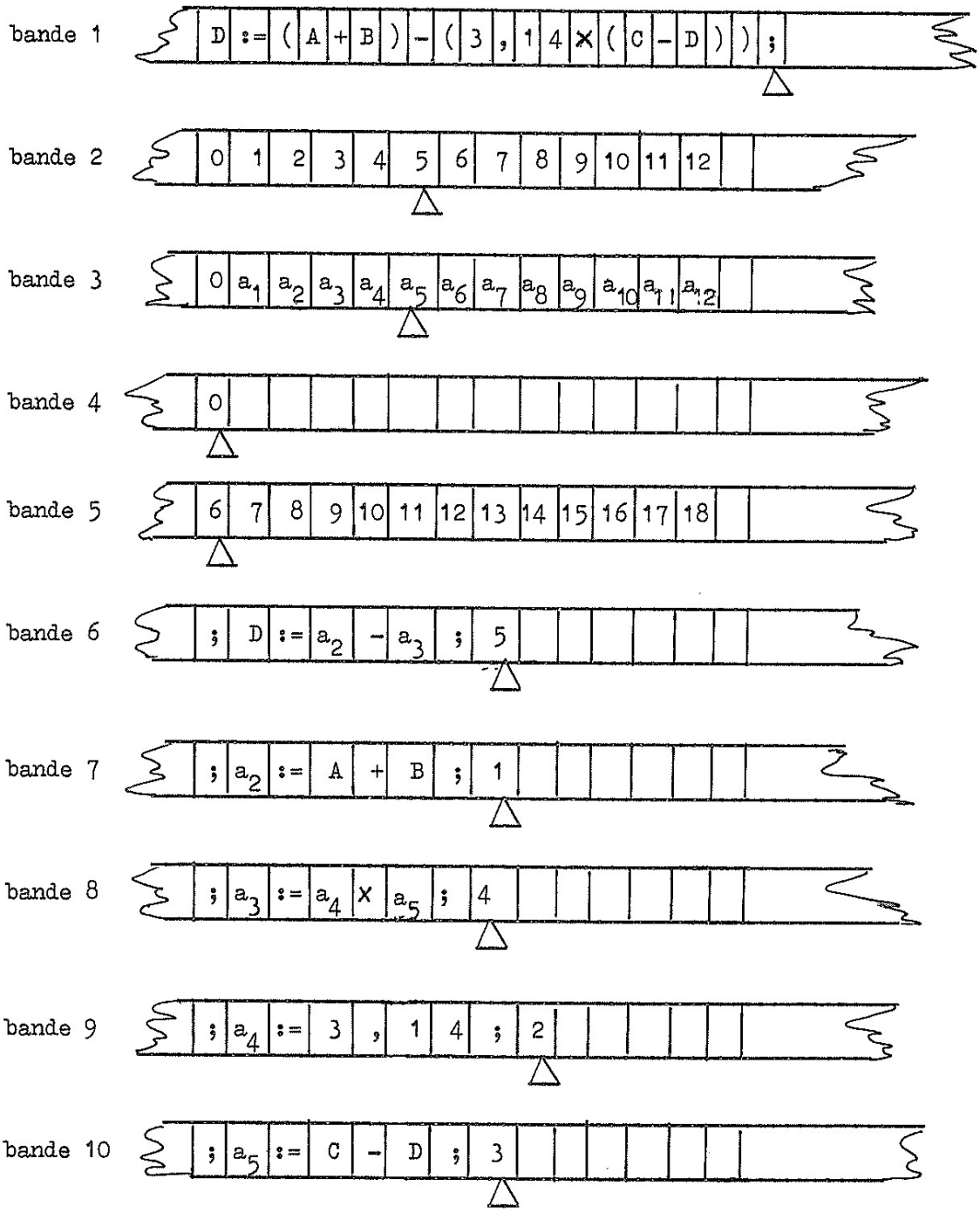
Le programme à exécuter peut être résumé dans le tableau suivant:

valeur de e_1	condition supplémentaire	programme élémentaire
Variable + ou - ou ×/ou / chiffre ou , : =	$e_3 \neq 0$ $e_4 = 0$ $e_4 \neq 0$	Pr 1 ;
variable	$e_3 = 0$	3 → ; m → ; Pr 1 ;
(		Pr 2 ; 1 → ;
chiffre ou ,	$e_4 = 0$	e_4 := 1 ; Pr 2 ; Pr 1 ;
+ ou - ou × ou /	$e_4 \neq 0$	e_4 := 0 ; Pr 3 ; Pr 4 ; Pr 1 ;
)		Pr 3 ; Pr 4 ; 1 → ;
;		Pr 3 ; fin .

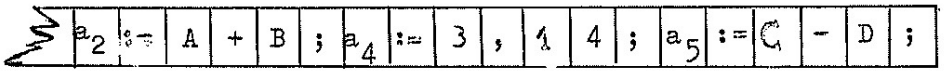
Dès qu'un programme élémentaire a été exécuté (sauf s'il correspond au cas où e_1 = ";"), la machine est prête à explorer la case suivante de la bande 1 : suivant la valeur de e_1, et en tenant compte de la condition supplémentaire, elle exécute alors le programme élémentaire correspondant, et ainsi de suite.

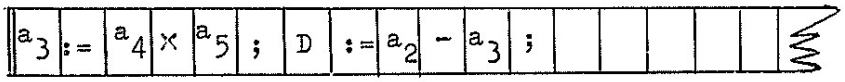
60

Lorsque le programme élémentaire qui correspond au cas où e_1 = ";" a été exécuté, on a la configuration finale:



Il est alors facile de composer la bande 23 - par exemple - qui doit contenir:

bande 23 

(suite) 

C'est-à-dire une suite d'instructions de L1 équivalente à l'instruction de L2 contenue dans la bande 1 ; on revient alors, sauf en ce qui concerne la bande 1 , à la configuration initiale. Notre problème est donc résolu.

A vrai dire, il était très simple, et ceci pour plusieurs raisons. La première c'est que la syntaxe des expressions, dans L2 , est assez rudimentaire en ce qui concerne l'usage des parenthèses. La seconde c'est qu'on a supposé que la suite de symboles à traduire était une instruction correcte de L2 . La troisième raison, enfin, c'est que nous avons supposé que le vocabulaire de L1 était une partie de celui de L2 . Et peut-être pensez-vous que cette dernière restriction nous a empêché d'atteindre vraiment le but dernier de nos efforts qui était de donner d'un programme écrit dans un langage de programmation quelconque, une traduction dans un langage-machine, c'est-à-dire un langage purement numérique. Vous verrez que, malgré tout, nous sommes très près de ce but: il vous suffira pour cela de résoudre l'exercice VI . 5 . Et vous constaterez aussi qu'une même machine peut fort bien exécuter directement des programmes écrits en L1 et des programmes écrits en L3 . Elle peut donc interpréter elle-même les programmes écrits en L2 .

Ainsi, un même langage peut fort bien être interprété par toute une gamme de machines. C'est cette indépendance relative qui justifie l'effort fait actuellement pour définir des langages de programmation bien adaptés au traitement de certaines classes de problèmes, sans avoir égard à la machine qui les interprétera.

Mais ce préambule n'a que trop duré: il est temps, maintenant, d'aborder le problème de la programmation par son côté pratique, en écrivant de vrais programmes, dans un vrai langage de programmation, pour une vraie machine.

Bonne Chance.

Exercice VI . 1

Représenter, sous la forme d'un organigramme, l'algorithme de traduction décrit dans le tableau de la page 59.

Exercice VI . 2

Compléter le programme de traduction pour composer la bande 23, comme il est dit à la page 61 . Ne pourrait-on pas simplifier l'algorithme correspondant en incorporant, dans l'algorithme de traduction, la formation de la suite

7 , 9 , 10 , 8 , 6

qui est celle des numéros des bandes qui contiennent respectivement les instruction de L1 numérotées

1 , 2 , 3 , 4 , 5

Exercice VI . 3

En s'inspirant des méthodes précédentes, écrire un programme - destiné à une machine de Turing généralisée - qui permette d'exprimer dans la notation décimale un nombre entier positif < 2000 exprimé "en chiffres romains" . Traiter de même le problème inverse.

Exercice VI . 4

Ecrire, de même, un programme qui permette d'exprimer en toutes lettres une somme exprimée en chiffres (francs et centimes).

Exercice VI . 5

A première vue, la description du langage L3 que j'ai donnée aux pages 55 et 56 semble satisfaisante. Mais si, en y regardant de plus près, vous découvrez quelque ambiguïté, ne manquez pas de corriger ma rédaction et de me signaler ce manquement impardonnable aux principes que j'ai énoncés quelques pages plus haut.

T A B L E D E S M A T I E R E S