ETH zürich Research Collection

Doctoral Thesis

Calculatrices digitales du déchiffrage de formules logico-mathématiques par la machine même dans la conception du programme

Author(s):
Böhm, Corrado

Publication Date:
1954

Permanent Link:
https://doi.org/10.3929/ethz-a-000090226 →

Rights / License:
In Copyright - Non-Commercial Use Permitted →

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.

ETH Library

Prom. Nr. 2186

CALCULATRICES DIGITALES
DU DÉCHIFFRAGE DE FORMULES LOGICO-MATHÉMATIQUES PAR LA MACHINE MÊME DANS LA CONCEPTION DU PROGRAMME

THÈSE

PRÉSENTÉE

À L'ÉCOLE POLYTECHNIQUE FÉDÉRALE, ZURICH,

POUR L'OBTENTION DU

GRADE DE DOCTEUR ÈS SCIENCES MATHÉMATIQUES

PAR

CORRADO BÖHM, ing. électr. dipl. EPUL
de Milan (Italie)

Rapporteur: Prof. Dr. E. STIEFEL
Co-rapporteur: Prof. Dr. P. BERNAYS

BOLOGNA
COOPERATIVA TIPOGRAFICA AZZOGUIDI
1954

Estratto dagli *Annali di Matematica pura ed applicata*
Serie IV, Tomo XXXVII (1954)

A MA FEMME

Leer - Vide - Empty

0. - Introduction.

0.1. Utilité d'une codification automatique.

De nos jours on tend de plus en plus à employer les grandes calculatrices digitales en raison des caractéristiques suivantes :

0.11. Possibilité d'exécuter une succession de calculs suivant un *programme* fixé à l'avance.

0.12. *Vitesse* de calcul remarquablement supérieure (de quelques centaines de fois) à celle des précédents types de calculatrices électromécaniques ou à la main.

Ces deux propriétés permettent d'établir une analogie, au point de vue de l'organisation et de l'économie, entre une de ces machines et un bureau de calcul. De ce même point de vue nous pourrions ébaucher une classification en trois classes des travaux à soumettre à une calculatrice digitale à programme de la manière suivante :

A) Calculs ayant un caractère d'*extrême urgence* et comportant un très grand nombre de données : p. ex. prévisions météréologiques ou bien dépouillements de scrutins électoraux.

B) Calculs de grande série comportant un très grand nombre d'opérations qui doivent être répétées plusieurs fois de suite, comme p. ex. dans l'inversion de matrices d'ordre élevé.

C) Tous les autres calculs ne pouvant se classifier ni sous *A*) ni sous *B*) : p. ex. l'intégration d'une équation différentielle donnée, la résolution d'une équation trascendente, etc.

Pour les catégories *A*) et *B*) la difficulté du programme et, en dernière analyse, la *durée* de sa préparation ne jouent pas un grand rôle, puisque la qualité requise pour le programme est celle de permettre de profiter au maximum de la vitesse de calcul. Par contre, pour la catégorie *C*) le temps dépensé par les personnes chargées de la préparation et du contrôle du programme peut être beacoup plus grand, d'un autre ordre de grandeur

même que le temps employé par la machine pour l'exécution des calculs relatifs au même programme (p. ex. il peut y avoir un rapport jours/minutes). Pour la catégorie *C*) c'est la propriété 0.11 qui est essentielle, c. à d. la facilité et la flexibilité de la programmation, telles qu'elles furent conçues lors du projet de la machine.

Dans ce qui suit nous supposons que le lecteur connaît le principe de fonctionnement d'une calculatrice digitale et plus spécialement celui des calculatrices de construction récente [1], [2] (1). Le travail de programmation peut même avoir une influence directe sur la durée globale des calculs: si, par hasard, lorsqu'on soumet un programme à la machine pour la première fois, les résultats du calcul ne concordent pas avec ceux qu'on avait prévus par un calcul de contrôle préalable, et si l'on peut exclure un défaut de fonctionnement de la machine, on est obligé d'admettre une au moins des causes perturbatrices suivantes:

a) le programme est fautif du point de vue logico-mathématique.

b) le programme n'a pas été correctement enregistré sur le support matériel qui doit être placé à l'entrée de la calculatrice.

Pour localiser ces fautes, des épreuves supplémentaires de calcul, souvent même à vitesse réduite, deviennent nécessaires avec une perte relativement grande du temps utile de la calculatrice.

Il vaut mieux chercher à éviter la production de fautes de toute espèce. Une méthode intéressante a été celle qui consiste à construire et à utiliser les automatismes auxiliaires qu'on groupe sous le nom de « machines à codifier » (2). Une autre méthode qui a donné de très bons résultats est celle adoptée par M. Wilkes [3], appelée « méthode de la library of sub-routines »; suivant cette méthode on cherche à composer chaque nouveau programme par des liaisons et combinaisons opportunes de programmes partiels établis à l'avance, dont les duplicatas se conservent dans une espèce de bibliothèque. Le contrôle d'un nouveau programme est ainsi réduit à celui des seules instructions de liaison entre les différentes *sub-routines*.

Au sujet de la mise en programme d'un calcul mécanique, il faut considérer que le travail humain repose sur une double connaissance: d'une part de la méthode pour résoudre un problème donné, d'autre part de l'organisation générale (fonctionnement et structure) de la machine employée. L'homme doit donc transformer chaque phase du calcul en une succession d'opérations pouvant être exécutées par la machine.

Or, dans quelle mesure la machine peut-elle même calculer elle-même cette succession à partir de certaines formules interprétant la méthode de résolution du problème proposé? La réponse est le but principal du présent

(1) Les chiffres entre crochets se réfèrent à la bibliographie placée à l'issue de cette étude.

(2) De telles machines ont été construites par M. Aiken et M. Zuse.

exposé. Dans la suite, pour abréger nous nommerons *codification automatique* cette question.

Le problème de la codification automatiques a été posé avec une certaine latitude pour ne pas devoir tomber de prime abord sur les difficultés attachées à la notion de calculatrice à programme. Comme le remarque aussi M. Von Neumann [4] nous ne possédons pas encore, aujourd'hui, une théorie satisfaisante des automates et en particulier des machines calculatrices à programme. Pour pouvoir formuler la question de la codification automatique plus rigoureusement que nous ne l'avons fait ici, nous allons utiliser dans le prochain paragraphe certains résultats dus à M. Turing.

Des questions similaires ont d'ailleurs déjà été traitées; voici un bref aperçu à ce sujet:

En 1949 M. Zuse [5] a traité la formulation d'un problème dans lequel il s'agit de reconnaître si une suite de signes comportant des variables et symboles d'operations algébriques (voir les exemples en 4.2) possède ou non un sens (3). Cette question constitue une partie du probléme plus général de la programmation automatique d'une formule contenant des parenthèses, et fait l'objet du chapitre 5 du présent travail. Par coïncidence M. Rutishauser aussi s'est occupé de cette dernière question [6] mais les principes de nos deux solutions sont entièrement différents (v. 5.1); ce genre de problèmes comporte de nombreuses voies pour atteindre le même but.

Avant d'achever ces remarques générales il nous paraît utile de signaler ceux des résultats de notre étude qui paraissent les plus susceptibles d'applications pratiques:

0.13. Possibilité d'écrire tout programme sous forme d'une succession de formules adhérant, le plus étroitement possible, aux conventions de notation suivies par les mathématiciens, tout en excluant les indications qui regardent davantage la calculatrice et ses particularités que le programme à réaliser.

0.14. Possibilité de résoudre le problème de la codification automatique.

La codification se fait par l'emploi exclusif de la calculatrice, pendant deux phases consécutives: durant la première phase, le support matériel sur lequel le programme a été enregistré sous forme de succession de formules est placé à l'entrée de la machine et la calculatrice exécute une série de calculs qui fournissent le même programme, mais enregistré sous forme d'instructions codifiées propres à être ultérieurement interprétées et exécutées automatiquement. Pendant la deuxième phase la calculatrice réalise les calculs, d'après les instructions codifiées produites précédemment.

---

(3) Dans le travail cité, M. Zuse exprime l'opinion qu'on est à même de construire aujourd'hui une nouvelle machine résolvant entre autres ce problème. Nous avons par la suite réussi à résoudre le même problème au moyen d'une tabulatrice à cartes perforées (système BULL). Le succès remporté nous a encouragé à entreprendre le présent essai de synthèse sur la programmation automatique.

Il est évident que les deux possibilités mentionnées facilitent le contrôle d'un programme en le réduisant à un contrôle à vue de formules, et éliminent les fautes de transcription puisque une transcription n'a pas lieu.

En outre, même si l'on fait abstraction de toute valeur pratique de ces recherches, l'existence même de ces possibilités semble posséder, à notre avis, un certain interêt au point de vue de la théorie des machines calculatrices.

0.2. Spécification de la catégorie de machines digitales envisagées.

Pour éviter des définitions circulaires des termes: calculatrices automatiques, programme, instructions codifiées, codification automatique, etc. précisons brièvement à quelles catégories de machines ces locutions se rapportent.

0.21. Parmi les calculatrices digitales automatiques nous rencontrons tout d'abord les machines de bureau (à la main ou électromécaniques) qui possédent quelques *micro-programmes* fixes; elles peuvent en effet exécuter automatiquement les séquences d'opérations pour déterminer p. ex. le produit ou le quotient de deux nombres donnés.

0.22. Les installations mécanographiques à cartes perforées sont costituées essentiellement par des calculatrices digitales susceptibles d'être employées différemment suivant leur programmation préalable. On varie le programme en changeant la disposition des circuits électriques de la machine, obtenant ainsi une variation correspondante dans la séquence des opérations arithmétiques effectuées.

0.23. Les calculatrices les plus évoluées diffèrent des précédentes, du point de vue de la programmation, surtout par le fait que la modification des circuits électriques qui déterminent la suite des opérations n'est pas effectuée à la main par l'opérateur, mais par l'intermédiaire de nombres conventionnels ou *instructions codifiées* introduites préalablement, et une fois pour toutes, dans la machine. D'après cette description le lecteur reconnaîtra aisément que, dans notre étude, il n'est question que de cette dernière catégorie de machines. Toutefois la différence entre 0.21, 0.22 et 0.23, tellement évidente lorsque l'on songe à la construction ou à l'utilisation d'une machine déterminée, risque de s'évanouir lorsqu'on doit utiliser, comme nous devons le faire, des propriétés d'une nature logique très générale du calcul mécanique. D'où la nécessité de faire appel aux notions du prochain paragraphe.

0.3. Utilisation de la théorie de Turing.

Depuis 1937 M. Turing [7] a créé une théorie logique qui lui a permis, en partant d'une analyse profonde de ce qu'est un calcul, de définir ce qu'on appelle aujourd'hui *calcul mécanique*. Sa définition découle de la description d'un type déterminé de machines automatiques, aptes à calculer

des *séquences* de chiffres représentant des nombres connus à l'avance ou bien jouissant de propriétés connues.

Son étude utilise cette définition pour atteindre rapidement certains résultats en logique mathématique. Certaines parmi ses conclusions — que nous rapportons plus bas — ont aussi un grand interêt pour une éventuelle future théorie des calculatrices, et elles ont influencé les progrès relatifs à la programmation des calculatrices dans dans les dix dernières années.

M. TURING a montré comment la notion de calculabilité mécanique d'un nombre est, au fond, équivalente à la notion d'existence de méthodes générales pour déterminer ce même nombre. Il a montré en outre, à partir de l'hypothèse qu'on sache construire des calculatrices particulières pour calculer des séquences particulières de nombres, qu'il existe une calculatrice dite *universelle* qui jouit de la propriété suivante : si l'on pourvoit la calculatrice universelle de la *description* (formalisée conventionnellement) du fonctionnement de *chaque* calculatrice particulière, la première est à même de *simuler* le comportement de la seconde, c. à d. de calculer à sa place.

Nous voulons admettre — ce qui est assez plausible — que les calculatrices les plus évoluées sont universelles, au sens spécifié par M. TURING. Ceci nous permet alors de formuler les deux hypothèses de travail suivantes, auxquelles nous aurons souvent recours par la suite :

0.31. Les calculatrices à programme de la catégorie 0.23 sont, au point de vue logico-mathématique, équivalentes entre elles.

Cette hypothèse nous permet de borner notre étude à un seul type de calculatrice, p. ex. une calculatrice à *trois adresses*, sans crainte de rien perdre en généralité.

0.32. Le « programme » est susceptible, par rapport aux calculatrices universelles, d'une double interprétation. La première est : « Description d'un comportement de la calculatrice ». La deuxième : « Description d'une méthode numérique de calcul ».

Cette hypothèse n'est qu'une nouvelle formulation de l'idée de TURING et justifie la recherche d'un formalisme apte à mettre cette idée pleinement en évidence.

0.4. **Résumé.**

Nous décrivons tout d'abord (CHAP. 1) la structure et l'organisation d'une calculatrice du type de celles déjà construites. Nous pouvons ainsi définir explicitement un programme cyclique fondamental traduisant en termes de cette machine la définition de « programme » (v. 0.32), avant même de spécifier quelles doivent être les instructions codifiées. Ce dernier choix étant fait, nous démontrons l'universalité de cette calculatrice et introduisons quelques instructions supplémentaires utiles dans la suite. Après être passé de la notation habituelle des programmes à une autre notation

(CHAP. 2), nous justifions le symbolisme introduit par la possibilité de pouvoir écrire tout programme dans ce langage formel, en faisant seulement recours à des notions logiques ou algébriques (CHAP. 3). De plus nous démontrons (CHAP. 4) qu'on peut rendre ce formalisme accessible à la calculatrice par l'intermédiaire d'un téléscripteur, en établissant une fois pour toute une correspondance biunivoque entre symboles et nombres entiers. Les formules traduisant algébriquement un programme de calcul peuvent contenir des parenthèses (CHAP. 5) ou des polynômes à plusieurs variables mis sous forme normale (CHAP. 6). Dans les deux cas nous pouvons faire exécuter par la calculatrice même, grâce à un programme fixe indépendant de la nature particulière des formules (CHAP. 7) les calculs nécessaires pour produire les instructions codifiées détaillées à partir des formules interprétant la méthode numérique envisagée. Enfin (CHAP. 8) nous avons mis en rélief la redondance logique de certaines opérations et rappelé des procédés d'arithmétisation du calcul des propositions.

1. - Description d'une calculatrice à programme à trois adresses.

1.0. La machine que nous allons décrire dans le grandes lignes est une calculatrice à programme de principe. En conséquence nous laissons de côté toute question regardant le système de numération (binaire ou décimale ou mixte) employé dans la calculatrice et de même toute considération regardant la représentation des fractions (emplacement de la virgule) et de nombres négatifs.

Sans perte de généralité nous pouvons supposer dans la suite que tout nombre qui entre en relation avec la calculatrice est un nombre entier non négatif d'une *longueur fixe* c. à d. composé d'un nombre fixe (p. ex. 14) de chiffres décimaux (4).

Double signification des nombres employés.

Tout nombre entier ainsi *quantifié* possède en général une deuxième signification: celle d'*instruction codifiée* (coded order) fixée une fois pour toute lors de la construction de la calculatrice par une table d'opérations codifiées ou *code* (voir 1.3). P. ex. le nombre zéro (4) interprété comme instruction cause l'arrêt du calcul (STOP).

Le nombres-nombres et le nombres-instructions tout en ne possédant aucun signe distinctif, subissent un traitement différent grâce à la structure de ce que nous appelons le *programme cyclique fondamental* (voir 1.2) de la calculatrice.

(4) P. ex. les nombres 00000000000000 et 00000000000001 représentent le zéro et l'unité.

1.1 **Organes de la calculatrice** (v. la fig. 1).

I. Une *unité arithmétique UA* où le nombres provenant d'autres organes de la calculatrice (fig. 1 connexions 3 et 7) sont soumis à des opérations arithmétiques (addition, soustraction, multiplication, etc.). De notre point

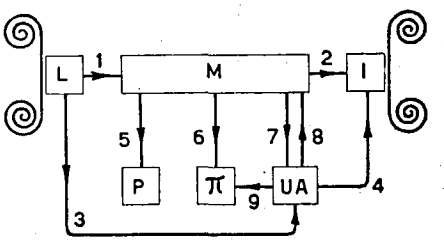

Figure 1.

de vue, il convient de concevoir toute opération s'effectuant dans l'*UA* comme *binaire*, c. à d. comme caractérisée par le fait de faire correspondre univoquement à tout couple ordonné de nombres — appelés respect. *1er terme* et *2ème terme* de l'opération — un 3ème nombre ou *résultat*.

II. Une *mémoire interne M* composée d'environ 1000 cellules dont chacune peut conserver un nombre ou une instruction (à 14 chiffres décimaux) pendant un laps indéterminé de temps. À chaque cellule est attaché un numéro d'ordre ou *adresse*. La mémoire interne rend possible à tout moment du calcul deux catégories d'opérations :

1.11. La *substitution* ou bien le *transfert à une adresse déterminée*, c. à d. l'action de transférer un nombre d'un organe de la calculatrice dans la cellule ayant une adresse quelconque mais déterminée, le nombre qui se trouvait dans cette cellule étant en même temps *oublié*, c. à d. remplacé par le nouveau (voir dans la fig. 1 les connexions 1 et 8).

1.12. L'*exploration* ou bien le *transfert depuis une adresse déterminée*, c. à d. l'action de transférer le *duplicata* du nombre se trouvant dans une cellule déterminée (et d'ailleurs quelconque), à une autre organe de la calculatrice (voir dans la fig. 1 les connexions 2, 5, 6, 7).

III. Un organe de *lecture* ou d'*entrée L* permettant d'introduire dans *M* (connexion 1) ou dans *UA* (connexion 3) les nombres (ou instructions) enregistrés sur un ruban en papier perforé ou magnétique.

IV. Un organe d'*impression* ou *sortie I* permettant de perforer ou enregistrer magnétiquement sur un ruban en papier, placé à la sortie de la machine, les nombres provenant de *M* (connexion 2) ou de *UA* (connexion 4).

V. Un *registre* π avec une capacité de quatre chiffres décimaux. Les connexions 6 et 9 (fig. 1) représentent respect. l'*exploration* des quatre derniers chiffres de droite d'un nombre contenu à une adresse quelconque de *M*, ou contenu dans *UA*. La fonction de ce registre sera expliquée en 1.2.

VI. Un organe de *commande automatique* ou *pilote* (control) P qui a pour but de coordonner l'action de tous les autres organes mentionnés, de façon que les instructions relatives à chaque programme de calcul soient exécutées correctement et dans l'ordre juste. En particulier toutes les connexions de la fig. 1 représentant la circulation des nombres à l'intérieur de la calculatrice, sont mises en activité par le *pilote*, chacune à l'instant convenable. Nous pouvons décrire globalement l'action du pilote en attribuant tout simplement à celui-ci la faculté de pouvoir premièrement interpréter, grâce à une *exploration* (connexion 5 de la fig. 1), le nombre contenu dans une certaine adresse comme une instruction codifiée, et de pouvoir ensuite en rendre effective l'exécution.

1.2. **Fonctionnement stationnaire.**

Après avoir décrit séparément les organes de la calculatrice, nous cherchons maintenant à décrire comment se déroule dans le temps un calcul quelconque, en nous bornant à la phase stationnaire, c. à d. en excluant les instants où la machine commence à calculer ou s'arrête (5). Il s'agit de caractériser le degré d'automatisme de la calculatrice et d'expliquer le mécanisme suivant lequel la présence de certains nombres ou instructions codifiées dans une région de la mémoire interne M entraîne le déroulement correspondant des calculs. Nous supposons donc qu'à un certain instant de la phase stationnaire les instructions codifiées soient inscrites en M de telle façon que la majorité de celles qui doivent être exécutées consécutivement se trouvent à des adresses consécutives.

L'action de calcul a lieu, indépendamment de la nature spécifique des calculs à effectuer, suivant un schéma fixe à structure périodique, dans lequel jouent un rôle essentiel le registre π et le pilote P.

Ce schéma comporte le trois phases successives suivantes :

1.21. Le pilote explore l'adresse de M spécifiée par le nombre contenu dans le registre π.

1.22. Le nombre contenu dans π est remplacé par celui qu'on obtient en augmentant le premier d'une unité.

1.23. Le pilote P, après avoir interprété comme instruction le nombre exploré pendant la phase 1.21, pourvoit à son exécution.

La phase suivante de calcul est formellement identique à la phase 1.21, celle qui lui succède à la phase 1.22, et ainsi de suite. L'ensemble 1.21,

(5) L'instant du commencement des calculs est choisi par l'opérateur, qui agit sur le pilote par l'intermédiaire d'organes que nous n'avons pas mentionnés pour réduire notre description à l'essentiel. De même l'instant où la machine cesse de calculer peut dépendre — sauf dans le cas où il résulte du programme — de plusieurs circonstances étrangères au fonctionnément habituel de la calculatrice.

1.22, 1.23 constitue le *programme cyclique fondamental.* Ce schéma de calcul est formel en ce sens qu'il ne décrit pas seulement *une* calculatrice à programme, mais plutôt une classe de calculatrices. En effet nous n'avons pas encore spécifié, outre à ce schéma, la nature des opérations arithmétiques ou logiques décrites par les instructions codifiées. Ce n'est qu'après que l'on a choisi une classe d'instructions codifiées et qu'on y a joint le programme cyclique fondamental que la description de principe de notre calculatrice est univoque.

Notre intention est de démontrer que, si l'on ne codifie que les opérations que nous avons mentionnées au cours de la description du paragraphe 1.1., cela définit déjà une calculatrice ayant un degré de flexibilité comparable à celle des calculatrices de la catégorie 0.23 c. à d. une calculatrice universelle.

Les opérations du paragraphe étaient, en résumant, les suivantes:

1.10) Opérations arithmétiques (addition, soustraction, etc).

1.11) et 1.12) Opérations de *substitution* et d'*exploration*, c. à d. des opérations de *transfert* comprenant les échanges avec le *monde extérieur* (connexions 1, 2, 3, 4 de la fig. 1), et les interconnexions 6, 7, 8, 9 [6] et l'opération STOP.

Pour conduire cette démonstration nous devons introduire au préalable ler notations habituelles relatives à tout programme, c. à d. définir un code.

1.3. Code à trois adresses.

Toute instruction codifiée est représentée par un nombre N ainsi défini:

$$N = 10^{10} \cdot c_1 + 10^8 \cdot c_{op} + 10^4 \cdot c_2 + c_r.$$

où c_1, c_2, c_r entiers < 1000, et c_{op} entier < 100; c. à d. la structure décimale du nombre N est donc la suivante:

$$\underbrace{\cdot}_{0}\ \underbrace{\cdot\cdot\cdot}_{c_1}\ \underbrace{\cdot\cdot}_{c_{op}}\ \underbrace{\cdot}_{0}\ \underbrace{\cdot\cdot\cdot\cdot}_{c_2}\ \underbrace{\cdot}_{0}\ \underbrace{\cdot\cdot\cdot\cdot}_{c_r}$$ (en tout 14 chiffres décimaux).

La signification de c_1, c_{op}, c_2 et c_r est fixée par les règles suivantes:

c_1 est l'adresse du 1er terme de l'opération.

c_{op} est le nombre de code caractéristique de l'opération.

c_2 est l'adresse du 2ème terme de l'opération.

c_r est l'adresse à laquelle on doit transférer le résultat de l'opération.

---

[6] La connexion 5 n'est pas mentionnée car elle appartient au programme cyclique fondamental.

Les c_{op} sont définis par la table suivante :

Symbole	c_{op}	*Résultat de l'opération correspondante*
+	01	Somme (modulo 10^{14}) du 1er et du 2ème terme.
$\dot{-}$	03	Différence entre le 1er et le 2ème terme, si 1er terme $>$ $>$ 2ème terme ; sinon zéro (7).
·	02	Produit (modulo 10^{14}) du 1er et du 2ème terme.
:	04	Partie entière du quotient de la division du 1er par le 2ème terme (7).
$\div$	06	Valeur absolue de la différence entre le 1er et le 2ème terme.
mod	07	Reste de la division du 1er terme par le 2ème.
$\cup$	08	Le plus grand de deux termes.
$\cap$	09	Le plus petit de deux termes.

Ainsi p. ex. l'instruction

$$N' = 0\,123\,04\,0567\,0890$$

acquiert la signification suivante :

« Le nombre contenu à l'adresse 123 est à diviser par le nombre contenu à l'adresse 567 ; la partie entière du quotient est à transférer à l'adresse 890 ».

Nous reprenons la notation générale avec des lettres à la place des adresses numérotées pour représenter toutes les opérations de transfert par le nombre indéfini unique ayant la structure décimale suivante :

$$T = 0c_1\,05\,0c_2\,0c_r.$$

où 05 est le numéro de code attaché à l'opération du transfert et c_1, c_2, et c_r ont la même signification qu'auparavant ; l'instruction codifiée T possède donc par convention la signification suivante : « Le nombre exploré à l'adresse c_2 est à transférer à l'adresse c_r ». On remarquera qu'on peut considérer le transfert comme une opération à deux termes, à condition de considérer comme négligeable l'adresse c_1 du premier terme ; il s'agit en réalité d'une opération à un terme seulement.

Convenons maintenant d'attribuer une fois pour toute l'adresse 000 au registre π et à chaque organe L et I (voir 1.1) la *même* adresse 001 de façon

(7) Ces modifications des opérations habituelles de soustraction et de division sont dictées par la nécessité de ne pas sortir de la classe de nombres entiers non négatifs (voir 1.0).

à pouvoir interpréter les trois nombres suivants :

$$J = 0c_1\ 05\ 0c_2\ 0000$$
$$L = 0c_1\ 05\ 0001\ 0c_r$$
$$I = 0c_1\ 05\ 0c_2\ 0001$$

respectivement de la manière suivante :

J) « Les quatre derniers chiffres à droite du nombre dont l'adresse est c_2 sont à transférer dans le registre π ».

L) « Le nombre inscrit sur le ruban d'entrée L est à transférer à l'adresse c_r et le ruban est à avancer jusqu'au prochain nombre inscrit ».

I) « Le nombre exploré à l'adresse c_2 est à enregistrer sur le ruban de sortie I et ce ruban est à avancer d'une place ».

On reconnaît aisément dans les trois instructions J, L, I respectivement une préparation à un *saut inconditionné* (v. [2] à page 8), un ordre de *lecture* et un ordre d'*impression* ou de *sortie*.

Convenons enfin d'interpréter le nombre

$$S = 0000\ 00\ 0000\ 0000$$

comme l'ordre de STOP. Dans la suite nous admettons aussi une deuxième cause d'arrêt : lorsque à un instant donné le contenu du registre π est 0000 (8).

En résumé nous avons représenté toute instruction codifiée de la machine décrite par un nombre entier N ayant une structure décimale fixe : en spécialisant la valeur numérique de certaines positions décimales nous avons défini plusieurs sousclasses d'opérations que nous rappelons ici par les lettres introduites le long de ce paragraphe : N' (come exemple d'instruction ayant un caractère arithmétique), T, J, L, I et S (instructions ayant un caractère logique).

1.4 Universalité de la calculatrice décrite.

Pour démontrer que la machine décrite a une flexibilité comparable à celle des calculatrices de construction récente, ils suffit évidemment de démontrer que toute instruction codifiée relative à ces dernières peut être exprimée au moyen d'instructions codifiées de la première. Nous nous bornons au cas le plus intéressant c. à d. à celui des *commandes conditionnées* (9) (C. C.), en nous réservant de discuter le question des instructions relatives aux dispositifs appelés respectivement « *i*-register » et « *B*-tube » dans un paragraphe ultérieur.

---

(8) Si dans la calculatrice le chiffre zéro est représenté par l'absence d'impulsions, cette deuxième cause est une conséquence de la première, comme on peut vérifier facilement en suivant le mécanisme du programme cyclique fondamental.

(9) En anglais : conditional call (v. aussi [2], à page 45).

La C. C. dont on fait le plus souvent usage est ainsi conçue — si l'on suppose qu'elle soit placée à l'adresse n — :

C. C. 1. « Si le nombre inscrit à l'adresse a est positif, exécuter l'instruction dont l'adresse est le nombre x, sinon exécuter l'instruction placée à l'adresse $n+1$ ». Les nombres a, x, n, étant connus à l'avance, le but de cette instruction est notoirement celui de pouvoir influencer automatiquement la succession des opérations, ou en général tout le déroulement du calcul à partir d'un certain instant, en fonction des résultats obtenus jusqu'à ce même instant. Nous allons montrer la possibilité de codifier une C. C. semblable, qui est une legère généralisation de C. C. 1, à savoir

C. C. 2. « Si le nombre contenu à l'adresse a est positif exécuter l'instruction dont l'adresse est x, sinon celle dont l'adresse est y ».

Démonstration. - Supposons tout d'abord que l'on puisse transférer à l'adresse c_2 le nombre u ainsi défini

$$u = \begin{cases} x & \text{si } (a) > 0 \\ y & \text{si } (a) = 0 \end{cases} \qquad \begin{array}{l}(a) \text{ représentant le nombre} \\ \text{contenu à l'adresse } a.\end{array}$$

À ce moment-là, l'exécution de l'instruction J nous conduirait immédiatement au but envisagé. Comme par ailleurs nous pouvons calculer u à l'aide de la relation suivante :

$$u = [1 \cap (a)] \cdot x + [1 \dot{-} (a)] \cdot y \qquad \text{(Pour } \cap \text{ et } \dot{-}\text{, v. 1.3)} \tag{1.41}$$

la démonstration est déjà achevée. Comme illustration écrivons la succession complète des instructions codifiées dans un cas particulier. Soit, p. ex.

$$x = 301, \qquad y = 302, \qquad a = 002, \qquad c_2 = 300$$

et le nombre 1 soit enregistré p. ex. à l'adresse 007, tandis que x et y le sont respectivement aux adresses 025 et 026. Alors la succession suivante d'instructions codifiées

$$\begin{array}{ll} n_1 = 0\,007\,09\,0002\,0028 & \text{Calcul de } 1 \cap (a) \\ n_2 = 0\,007\,03\,0002\,0029 & \text{» de } 1 \dot{-} (a) \\ n_3 = 0\,028\,02\,0025\,0030 & \text{» de } [1 \cap (a)] \cdot x \\ n_4 = 0\,029\,02\,0026\,0031 & \text{» de } [1 \dot{-} (a)] \cdot y \\ n_5 = 0\,030\,01\,0031\,0300 & \text{» de } u \\ J = 0\,000\,05\,0300\,0000 & \text{Saut inconditionné,} \end{array} \tag{1.42}$$

est équivalente à la C. C. 2.

1.5 Rôle du *B*-tube et du *i*-register.

Dans [8] on montre par des exemples comment les dispositifs appelés *B*-tube ou *i*-register facilitent le « calcul des adresses » grâce aux instructions codifiées supplémentaires qui ont leur origine en eux. Leur inutilité du point

de vue purement logique a été également remarquée ([8] page 51); d'ailleurs il est presque évident que nul n'empêche (même dans la calculatrice que nous venons de décrire) de modifier par le jeu du programme les instructions codifiées en ajoutant, p. ex. à une adresse qui fait partie d'une instruction un nombre donné entier, (mod 1000), avant que l'instruction même ne soit exécutée. Tel est, en gros, l'effet du *B*-tube.

1.6 **Opérations de « substitution et exploration itérées ».**

Si l'on refléchit pourtant à l'utilité pratique de ces instructions auxiliaires pour le calcul d'adresses, on doit reconnaître qu'elles remplissent un certain rôle dans la résolution du problème de la codification automatique par la méthode de la bibliothèque de sub-routines. En effet, à l'aide de ces instructions auxiliaires les instructions composant la sub-routine ne subissent que de très faibles modifications suivant la zone de la mémoire interne (en définitive: les adresses) où elles sont placées.

Comme notre intention est en dernière analyse d'exprimer les instructions codifiées par des formules évidemment *invariantes* par rapport aux adresses de la mémoire interne, nous allons introduire dans la machine décrite des dispositifs qui rendent possible d'effectuer ce que nous appelons brièvement:

1.61 « la substitution itérée » et
1.62 « l'exploration itérée », c. à d.

(v. de nouveau 1.11 et 1.12) des opérations ainsi definies:

1.61. Transfert d'un nombre (placé dans un certain organe de la calculatrice) à l'adresse indiquée par les derniers trois chiffres à droite du nombre contenu à une adresse donnée.

1.62. Transfert à un certain organe de la calculatrice du duplicata du nombre se trouvant à l'adresse indiquée par les trois chiffres à droite du nombre contenu à une adresse donnée.

La description des dispositifs qui effectuent ces opérations sortirait du cadre de cette étude; par contre il est utile de montrer comment on peut incorporer les opérations « itérées » dans le code numérique à trois adresses décrit précédemment.

Rappelons que la structure décimale d'une quelconque des instructions décrites était:

$$N = 0c_1\, c_{op}\, 0c_2\, 0c_r$$

et remarquons:

1° Que l'exécution de cette instruction implique, en général, deux explorations (aux adresses c_1 et c_2 respectivement) et une substitution (à l'adresse c_r).

2° Que le chiffre immédiatement à gauche du premier chiffre de c_1, c_2, c_r est zéro.

Convenons maintenant de distinguer l'exploration *itérée* de c_1 ou c_2 ou bien la substitution *itérée* de c_r des correspondantes opérations *simples* en remplaçant respectivement avec l'unité le 1ère, 7ème, 11ème chiffre du nombre N depuis la gauche, ces remplacements pouvant être, suivant le cas, simultanés ou pas.

Exemple. Le nombre-instruction

$$n' = 1997\ 04\ 1996\ 1995$$

acquiert ainsi la signification suivante :

« Le nombre contenu à l'adresse égale au nombre (mod 1000) contenu à l'adresse 997 est à diviser par le nombre contenu à l'adresse égale au nombre (mod 1000) contenu à l'adresse 996; la partie entiére du quotient est à transférer à l'adresse égale au nombre (mod 1000) contenu à l'adresse 995 ».

Le nombre n' est évidemment invariant par rapport à n'importe quel changement d'adresse des trois termes de la division, tandis que le nombre N' (v. 1.3) ne l'est pas; on comprend donc bien l'interêt que des instructions du type de n' peuvent avoir pour la codification automatique.

2. - Passage à un autre système de notation.

Notre but final est de montrer la possibilité de résoudre les problèmes de la codification automatique à l'aide de la calculatrice de principe décrite dans le chapitre précédent. Pour simplifier cetta tâche nous proposons un changement de notation purement formel pour la représentation des nombres-instructions à trois adresses.

Considérons une instruction absolument quelconque parmi celles que nous avons introduites précédemment. Elle aura la structure décimale suivante :

$$\varepsilon_1 c_1\ c_{op}\ \varepsilon_2 c_2\ \varepsilon_r c_r$$

où c_1, c_2, c_{op}, c_r ont la signification bien connue et chaque ε_i ($i = 1, 2, r$) représente à volonté un des chiffres 0 ou 1. Convenons dans la suite :

2.1. D'effacer purement et simplement ε_i si $\varepsilon_i = 0$, et de remplacer chaque $\varepsilon_i = 1$ par le symbole $\downarrow$.

2.2. D'interposer toujours entre c_2 et c_r (ou $\downarrow c_r$) le *symbole de transfert* $\rightarrow$.

2.3. De remplacer c_{op} par le symbole d'opération correspondant (v. la table en 1.3) en laissant toutefois de côté le groupe inessentiel

$$\varepsilon_1\ c_1\ c_{op} \quad \text{lorsque} \quad c_{op} = 05.$$

2.4. De remplacer systématiquement les adresses 000 et 001 par les symboles π et ? respectivement, et les 52 adresses de 002 à 053 par les lettres $a, b, \ldots, z, A, B, \ldots, Z$ en ordre alphabétique.

2.5. D'employer des lettres avec indice ou des caractères non latins pour indiquer des adresses numériques > 053; en particulier Ω pour une adresse qui contient toujours le nombre 0 (STOP), p. ex l'adresse 998.

2.6. D'interpréter un nombre écrit en italique comme l'indication d'une *adresse qui a pour contenu ce même nombre,* p. ex. *123* représente toute adresse contenant le nombre 123 [10].

Utilisons ces conventions pour transcrire dans la nouvelle notation les nombres-instructions rencontrés dans les pages précédentes :

N'	$123 : 567 \rightarrow 890$	(convention 2.2, 2.3)	
T	$c_2 \rightarrow c_r$	» » »	
J	$c_2 \rightarrow \pi$	» » » 2.4)	
L	$? \rightarrow c_r$	» » » »	
I	$c_2 \rightarrow ?$	» » » »	
Instruc. équival. à S	$\Omega \rightarrow \pi$ ou bien	» » » » 2.5)	
	$0 \rightarrow \pi$	» » » » » 2.6)	
n'	$\lfloor 997 : \lfloor 996 \rightarrow \lfloor 995$	» 2.1, 2.2, 2.3)	
N [11]	$c_1\ op\ c_2 \rightarrow c_r$	» » » »	

Le bref programme codifié équivalent à une C. C. 2 (v. (1.42)) prend l'aspect suivant :

(2.7)
$$\begin{array}{ll} n_1 & 1 \cap a \rightarrow A \\ n_2 & 1 \div a \rightarrow B \\ n_3 & A \cdot x \rightarrow C \\ n_4 & B \cdot y \rightarrow D \\ n_5 & C + D \rightarrow u \\ J & u \rightarrow \pi. \end{array}$$

1ÈRE REMARQUE. – La correspondance entre le système habituel de notation et celui que nous venons de proposer est biunivoque. Le passage inverse s'effectue en effaçant le symbole $\rightarrow$ (en ajoutant $c_{op} = 05$ seulement si le premier terme et le symbole d'opération manquent) et en remplaçant chaque

---

[10] La notation traditionnelle correspondante est $\langle 123 \rangle$ (v. p. ex. [6]. page 4).

[11] « *op* » remplace ici et pour la suite le symbole d'une opération choisie arbitrairement parmi les opérations codifiées.

lettre ou symbole d'opération par son numéro de code suivant les conventions établies : on aura soin d'écrire devant chaque adresse 1 ou 0 suivant que le symbole ↓ est présent ou absent. Nous avons appélé → symbole de transfert car il remplace l'opération 05 ; cependant la fonction véritable de ce symbole est de mettre en évidence que dans l'exécution de $c_1\ op\ c_2 \rightarrow c_r$ les adresses c_1 et c_2 subissent un traitement différent de celui que subit c_r. En effet les deux premières sont l'objet d'une *exploration* (pour laquelle nous n'avons prévu aucun symbole) tandis que la dernière est l'objet d'une *substitution*.

2ÈME REMARQUE. – L'introduction des opérations « substitution et exploration itérées » dans le code à trois adresses pourrait laisser croire que la double itération de ces opérations constitue un problème nouveau et rend nécessaire une opération codifiée nouvelle ou un symbole nouveau. Or il n'en est rien comme nous allons le montrer au moyen d'une application relativement simple du symbolisme proposé. En effet la « substitution itérée deux fois » suivante : « Transférer un nombre de l'adresse x à l'adresse contenue dans l'adresse contenue dans l'adresse A » s'exprime en deux lignes par

$$\downarrow A \rightarrow P$$
$$x \rightarrow \downarrow P.$$

De même pour l'opération inverse qui est une sorte d' « exploration deux fois itérée » :

$$\downarrow A \rightarrow P$$
$$\downarrow P \rightarrow x.$$

3. – Justification du symbolisme introduit.

3.1. Possibilité d'une formulation abstraite d'un programme.

Au chap. 1 nous avons démontré l'universalité de la calculatrice décrite. Cela équivaut à l'affirmation que toute description de méthodes de calcul peut être exprimée à l'aide d'une suite convenable des symboles $a, b, \ldots, z, A, B, \ldots, Z, \Omega, \pi, ?, \rightarrow, \downarrow, +, \div, \ldots, \cup, \cap$, la signification de ces symboles étant celle qui fut définie dans le chapitre précédent.

Or, notre intention est de montrer que pour se servir correctement de ces symboles il n'est pas nécessaire de remonter chaque fois à leur signification opératoire par rapport à la calculatrice. Les mêmes symboles — et c'est en quoi consiste l'avantage pratique de la notation — sont susceptibles d'une deuxième interprétation de nature surtout logique et algébrique [12] ; cette

[12] On peut déjà entrevoir ce résultat en comparant la formule (1.41) avec celle qu'on obtient en remplaçant partout dans le premières cinq lignes des formules (2.7) le signe → par =, et en éliminant des cinq équations ainsi obtenues, A, B, C, et D considérées comme des grandeurs algébriques.

interprétation n'est donc directement reliée à aucune notion relative à une machine, elle se rattache plutôt à celle de « description d'une méthode numérique de résolution d'un problème donné », locution que par la suite nous remplacerons brièvement par le mot *description.*

Ce qui suit a évidemment un caractère heuristique et pourrait aussi bien valoir de définition pour ce que nous entendons, aujourd'hui, par *description.*

Toute *description* implique qu'on soit pourvu de *données* numériques et de *renseignements exhaustifs* sur les opérations arithmétiques à accomplir sur les données, afin d'obtenir de proche en proche les valeurs intermédiaires et les résultats définitifs du calcul.

Toute *description* est formée d'une alternance de formules et de phrases; les premières indiquent d'habitude la succession des opérations et les deuxièmes indiquent les variantes, les répétitions, ou en général les conditions auxiliaires ayant une influence sur le déroulement ultérieur du calcul. Par exemple, il existe des *descriptions* utilisant des méthodes itératives, le nombre des itérations étant inconnu *a priori* ou des descriptions telles que certains résultats intermédiaires nécessaires pour la poursuite des calculs doivent satisfaire à des conditions auxiliaires (positivité, réalité des nombres, tirages au sort, etc).

Nous admettons donc que pour toute *description,* l'on peut indiquer *a priori* la succession définitive des opérations arithmétiques, ou du moins le critère de choix, c. à d. les conditions qu'il faudra vérifier à un instant ultérieur donné, pendant les calculs, pour déterminer univoquement la succession des calculs.

Le fait de renoncer à un ordre de complexité plus élevé que celui-ci a pour conséquence la possibilité de décomposer chaque *description* en une succession de formules — de type logico-mathématique — au moyen de l'application d'un système de conventions regardant surtout:

3.2. La décomposition de toute *description* en un certain nombre de groupements semblables, chacun composé de trois parties.

3.3 Une décomposition ultérieure de trois parties de chaque groupement en formules comportant des opérations tout au plus binaires.

3.2. Décomposition en groupements semblables. - Interprétation graphique.

Chaque *description* est décomposable en groupements différents ayant une structure semblable. Appelons $A, B, C, \ldots, K, \ldots$ ces groupements. Un groupement quelconque K comportera les trois parties suivantes:

·) Indication que ce qui suit appartient au groupement K.

·) Indication chronologique des opérations arithmétiques et des grandeurs sur lesquelles on opère.

•) Indication sur le prochain groupement à choisir (le choix étant en général subordonné à certaines conditions).

Nous convenons d'appeler le premier groupement à choisir A; de même appelons Ω le groupement caractérisé par le fait de ne posséder aucune opération ni aucune indication sur le prochain groupement à choisir; il s'agit évidemment dans ce cas de l'indication que la *description* est achevée. La décomposition que nous venons de suggérer possède l'avantage de rendre inessentiel l'ordre de succession des groupements, qui peuvent être arbitrairement permutés sans nuire à l'intelligibilité de la *description*.

Nous ferons aussi usage, dans la suite, d'une interprétation graphique d'une telle décomposition au moyen d'un *graphe* d'un type très général [9]. Il s'agit d'une généralisation du « diagramme du structure » [10] que nous obtenons aisément de la façon suivante: à tout *groupement* on fait correspondre un POINT dénommé par la même lettre; soient ensuite K et L deux points ou groupements arbitraires. On convient de les joindre par le SEGMENT ORIENTÉ $\vec{KL}$ *si et seulement si*, dans la troisième partie du groupement K, L *est inclus parmi les prochains groupements à choisir*. Le graphe est dessiné complètement lorsque toutes les liaisons que la *description* comporte ont été exécutées. Chaque processus itératif est représenté dans le graphe par un *cercle* ou par une *boucle* (v. fig. 2, page 27). À chaque groupement comprenant un critère de choix parmi n alternatives correspond un point à au moins n branches dans le graphe (v. fig. 5, page 44, point D où $n = 11$).

3.3. Conventions regardant l'écriture des formules.

Rappelons tout d'abord que dès le début du chap. 1 nous nous sommes bornés à ne considérer que des nombres entiers non négatifs.

Les grandeurs données et les résultats intermédiaires ou définitifs seront désignés par des lettres choisies à volonté, mais si possible avec des caractères minuscules; par contre, les groupements seront designés par des majuscules. Les constantes numériques seront écrites en italique et les opérations arithmétiques seront représentées par leurs symboles habituels (v. tableau en 1.3).

Le grandeurs pourvues d'indices peuvent aussi être directement représentées comme nous allons le montrer, si l'on renonce (en général) à faire parcourir aux indices le même ensemble de valeurs qu'ils parcourent dans la *description*. Toutefois cette réserve n'est pas trop grave car, dans la pire des hypothèses, il s'agit de transformer l'ensemble des valeurs prises par chaque indice par l'addition d'un nombre constant, connu *a priori*. Les conventions ultérieures pour la traduction des phrases en formules peuvent être

condensées en l'emploi du « dictionnaire » suivant:

Locution	*Symbole*
« Devient » ou bien « est posé ègal à ... »	$\rightarrow$ [13]
« Un nombre connu »	?
« Le groupement à choisir prochainement »	π
« La grandeur dont i est l'indice »	$\downarrow i$
« Le groupement qui commence ici »	π'
« Aucun groupement »	Ω ou bien 0.

Les applications plus importantes deviennent ainsi:

Phrase ou formule	*Formule*	*Traduction littérale de la formule*
Les nombres a, b soient donnés.	$? \rightarrow a$ $? \rightarrow b$	Un nombre connu est posé égal à a. » » » » » » » b.
x est un résultat.	$x \rightarrow ?$	x devient un nombre connu.
Le prochain groupement à choisir est C.	$C \rightarrow \pi$	C devient le prochain groupement à choisir.
Il ne faut choisir aucun groupement suivant.	$\Omega \rightarrow \pi$ ou bien $0 \rightarrow \pi$	Aucun groupement ne devient le prochain groupement à choisir.
Soit r le produit de a et b.	$a \cdot b \rightarrow r$	$a \cdot b$ est posé égal à r.
Si $m = 1$ le prochain groupement à choisir est S; si $m = 0$ il n'y a pas de prochain groupement.	$m \cdot S \rightarrow \pi$	$m \cdot S$ devient le prochain groupement à choisir.
Introduisons le nombre h et posons $h = 1$.	$1 \rightarrow h$	1 devient h.
Appelons dorénavant m ce qui était $m + 1$. Ou bien: augmentons m d'une unité.	$m + 1 \rightarrow m$	$m + 1$ devient m.
Ici commence le groupement K [14].	$\pi' \rightarrow K$	Le groupement qui commence ici est posé égal à K.

[13] Notre définition du signe $\rightarrow$ par rapport à la calculatrice (v. (2.2)), correspond à celle de M. Von Neumann [10]. D'autre part la définition logique ci-dessus embrasse soit la signification donnée par M. Zuse [5] au signe $\rangle =$ « ergibt » que celle donnée par M. Rutishauser [6] au signe $\subset\!\!\rightarrow$. Plutôt que d'employer trois symboles différents [6] pour décrire ce qui dans la machine s'effectue par une seule opération (c. à d. une substitution), nous avons préféré élargir quelque peu la définition logique du signe $\rightarrow$.

[14] Nous traduisons systématiquement ainsi la première partie de chaque groupement.

Le dictionnaire et l'échantillon de formules que nous avons présentés doit suffire pour traduire en une succession de formules toute description, si l'on tient en outre compte des trois règles suivantes de « syntaxe » :

3.31. *Règle d'explicitation.* – Dans une succession de formules « bien écrites » chaque lettre ([15]) apparaissant à la gauche du signe $\rightarrow$ a déjà paru précédemment ([16]) au moins une fois à la droite du même signe ; à droite du signe $\rightarrow$ il n'y a jamais plus d'une lettre.

3.32. *Règle de transformation des indices.* – Si dans la description figure une grandeur avec un indice i, i variant p. ex. de 0 à n, il faut remplacer i par l'indice $i + z$ où $z \geq 100$ est un nombre fixe. S'il y figure plus d'une grandeur dépendant du même indice i; la notation $\mid i$ (v. page préc.) ne peut plus suffire : ceci est un avertissement que l'on doit alors introduire en général autant d'indices (différant les uns des autres par des constantes) que de grandeurs ; enfin les ensembles de nombres parcourus par des indices différents ne doivent avoir aucun élément en commun et tout indice doit être un nombre entier à trois chiffres (ce qui implique en tout cas $n < 900$).

3.33. *Règle pour l'arithmétisation du critère de choix du groupement suivant.* – La nécessité d'un développement univoque dans le temps nous oblige à admettre que le critère de choix parmi les groupements $A, B, \ldots, K, \ldots$ est toujours formulable ainsi : « Si la condition C_A est satisfaite il faut choisir A comme prochain groupement, si la condition C_B est satisfaite il faut choisir $B, \ldots$ si la condition C_K est satisfaite il faut choisir $K, \ldots$ », les conditions C_A, $C_B, \ldots$, $C_K, \ldots$ constituant une alternative stricte en ce sens qu'une et une seule d'entre elles doi être satisfaite simultanément.

Supposons maintenant que les groupements A, B, .., $K, \ldots$ soient des nombres entiers positifs tous différents entre eux et qu'on puisse faire correspondre à toute condition C_K un nombre entier non négatif e_K de façon à obtenir pour chaque groupement :

« $e_K = 0$ si et seulement si C_K est vérifié ».

Si l'on appelle S le groupement à choisir, il est aisé de voir que la règle cherchée s'exprime alors par la formule

$$S = (1 \dot{-} e_A) \cdot A + (1 \dot{-} e_B) \cdot B + \ldots + (1 \dot{-} e_K) \cdot K + \ldots .$$

On peut considérer S comme le produit scalaire de deux vecteurs dans l'espace à n dimensions (n étant le nombre de termes de l'alternative) vecteurs

([15]) Certaines lettres peuvent faire exception à cette règle : à savoir Ω (trivialement), π', et les lettres qui représentent les groupements de la description. Ces exceptions feront l'objet d'une discussion particulière au chap. 4.

([16]) « Précédemment » signifie « dans une formule précédente du même groupement ou bien dans une formule appartenant à un groupement tel qu'un point qui parcourt le graphe en partant du grpupement A dans la direction des flèches rencontre ce groupement en premier lieu ».

ayant pour composantes respectivement $(1 \dot{-} e_A,\ 1 \dot{-} e_B, \ldots,\ 1 \dot{-} e_K, \ldots)$ et $(A, B, \ldots, K \ldots)$; toutefois comme $1 \dot{-} e$ est 0 ou 1 suivant que $e > 0$ ou $e = 0$, il suit que le premier vecteur est vecteur-unité et appartient à la base: on a donc bien $S = A,\ B, \ldots,\ K, \ldots$ suivant respectivement que $0 = e_A$, $e_B, \ldots, e_K, \ldots$; c. q. f. d. [17].

3.4. Deux exemples de description.

1) Algorithme d'Euclide: recherche du plus grand diviseur commun m de deux nombres donnés (v. fig. 2).

	$\pi' \rightarrow A$
A. Soient a, b deux nombres donnés. Appelons M le plus grand et m le plus petit.	$? \rightarrow a$
	$? \rightarrow b$
	$a \cup b \rightarrow M$
	$a \cap b \rightarrow m$
	$B \rightarrow \pi$
	$\pi' \rightarrow B$
B. Le reste de la division de M par m soit r. Si $r = 0$ poursuivons sous C, autrement sous D.	$M \bmod m \rightarrow r$
	$\{[(1 \dot{-} r) \cdot C] + [(1 \cap r) \cdot D]\} \rightarrow \pi$ [18]
	$\pi' \rightarrow C$
C. m est le résultat. Fin.	$m \rightarrow ?$
	$\Omega \rightarrow \pi$
	$\pi' \rightarrow D$
D. Appelons M le nombre m, et appelons m le nombre r; poursuivons de nouveau sous B.	$m \rightarrow M$
	$r \rightarrow m$
	$B \rightarrow \pi$

2) Généralisation à n nombres; calcul du plus grand diviseur commun de n nombres donnés (v. fig. 3).

La description est beaucoup plus compliquée que la précédente, pas seulement du fait que $n > 2$ mais surtout car elle doit être valable pour n'importe quelle valeur de n (en pratique nous supposons $n \leq 800$). La méthode envisagée peut se résumer en peu de mots: n étant connu, soit $(a_n, a_{n-1}, \ldots, a_i, \ldots, a_2, a_1)$ l'ensemble de nombres donné. Soit ensuite m le plus petit parmi les a_i et r_i pourtant le reste de la division de chaque a_i par m. Considérons l'ensemble de $h \leq n$ nombres $(r_1, r_2, \ldots, r_k, \ldots, r_h = m)$ obtenu à

---

[17] Dans le cas d'une alternatives à deux termes on retrouve une formule analogue à la formule (1.41) par l'utilisation de l'identité $1 \dot{-} (1 \dot{-} e) \equiv 1 \cap e$ (e entier ≥ 0).

[18] L'expression à la gauche du signe $\rightarrow$, tout en étant compréhensible, n'obéit pas à la convention 3.3 à page 21. (L'on devrait écrire cinq lignes comme en (2.7)). Comme toutefois l'entier chapitre 5 est dédié à la démonstration qu'une expression avec parenthèses peut être directement confiée à la calculatrice, que le lecteur nous pardonne ces anticipations au bénéfice de la clarté.

partir de l'ensemble des r_i en lui ajoutant m et en écartant les restes nuls. Continuons les calculs en traitant les r_k comme auparavant les a_i, et ainsi de suite, jusqu'au moment où $h = 1$. Alors m est le nombre cherché.

Nous allons maintenant écrire la description détaillée pour le calcul mécanique. On remarquera combien la description qu'on vient d'esquisser contient de sous-entendus, même du point de vue logique ou algébrique; on notera en outre que les ensembles de nombres parcourus par les indices i et k ne peuvent pas, dans ce cas particulier, être disjoints car à un certain instant les r_k prennent littéralement la place des a_i. Le nombre z (v. convention 3.32) vaut dans la suite 100.

A. Soit donné le nombre n, et posons $i = n + 1$.

$$\pi' \rightarrow A$$
$$? \rightarrow n$$
$$n + 101 \rightarrow i$$
$$B \rightarrow \pi$$

B. Diminuons i d'une unité; a_i est donné. Si $i = 1$ la suite sous C, autrement recommençons sous B.

$$\pi' \rightarrow B$$
$$i - 1 \rightarrow i$$
$$? \rightarrow \downarrow i$$
$$[(1 \dot{-} (i \dot{-} 101)) \cdot C] + [(1 \cap (i \dot{-} 101)) \cdot B] \rightarrow \pi$$

C. h est au début égal à n. Appelons m le nombre a_1.

$$\pi' \rightarrow C$$
$$100 + n \rightarrow h$$
$$\downarrow i \rightarrow m$$
$$D \rightarrow \pi$$

D. Augmentons i d'une unité. Appelons m le plus petit des deux nombres a_i et m. Si $i = h$ la suite sous E, autrement recommençons sous D.

$$\pi' \rightarrow D$$
$$i + 1 \rightarrow i$$
$$\downarrow i \cap m \rightarrow m$$
$$\{[(1 \dot{-} (h \dot{-} i)) \cdot E] + [(1 \cap (h \dot{-} i)) \cdot D]\} \rightarrow \pi$$

E. Posons $i = 0$ et $k = 1$.

$$\pi' \rightarrow E$$
$$100 \rightarrow i$$
$$101 \rightarrow k$$

F. Augmentons i d'une unité. Calculons le reste r de la division de a_i par m. Si $r = 0$ la suite sous G, autrement sous H.

$$\pi' \rightarrow E$$
$$i + 1 \rightarrow i$$
$$\downarrow i \bmod m \rightarrow r$$
$$\{[(1 \dot{-} r) \cdot G] + [(1 \cap r) \cdot H]\} \rightarrow \pi$$

G. Si $i = h$ la suite sous I, autrement recommençons sous F.

$$\pi' \rightarrow G$$
$$\{[(1 \dot{-} (h \dot{-} i)) \cdot I] + [(1 \cap (h \dot{-} i)) \cdot F]\} \rightarrow \pi$$

H. Attribuons à la valeur actuelle de r l'indice k. Augmentons k d'une unité. La suite de nouveau sous G.	$\pi' \rightarrow H$ $r \rightarrow \lfloor \overline{k}$ $k + 1 \rightarrow k$ $G \rightarrow \pi$
I. Appelons h la valeur actuelle de k. Posons $m = r_h$. Posons $i = 1$ et appelons m le nombre r_i. Si $h = 1$ la suite sous L; autrement recommençons sous D en convenant d'appeler $(a_1, a_2, \ldots, a_i, \ldots, a_h)$ les nombres $(r_1, r_2, \ldots, r_k, \ldots, r_h)$.	$\pi' \rightarrow I$ $k \rightarrow h$ $m \rightarrow \lfloor h$ $101 \rightarrow i$ $\lfloor i \rightarrow m$ $\{[(1 \div (h \div 101)) \cdot L] + [[1 \cap (h \div 101)] \cdot D]\} \rightarrow \pi$
L. m est le résultat. Fin.	$\pi' \rightarrow L$ $m \rightarrow ?$ $\Omega \rightarrow \pi$

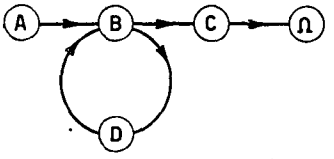

Figure 2.

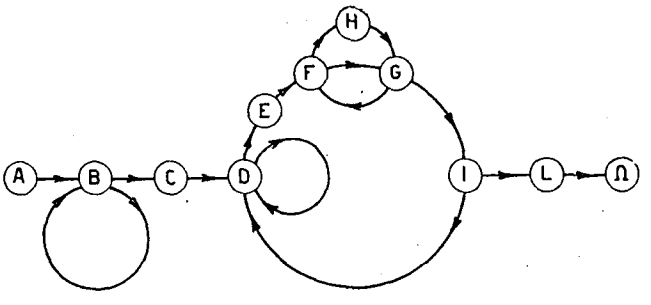

Figure 3.

4. - Principe de la codification automatique.

4.1. Programme d'entrée.

Proposons-nous de répondre à la question suivante :

Après avoir exprimé la description d'une méthode numérique relative à la solution d'un problème donné dans le symbolisme dont les règles formelles ont été établies au chap. précédent, exécutons la transformation de notation inverse à celle indiquée au chap. 2. Nous obtenons ainsi une suite de nombres-instructions.

Par quel procédé cette suite de nombres peut-elle être placée dans la mémoire interne de la calculatrice de façon à constituer le programme de calcul définitif pour la solution du problème proposé ?

Remarquons que ce n'est que si ce procédé existe et s'il est général que nous pouvons affirmer avoir résolu, (au moins en principe), le problème de la codification automatique. Dans ce cas, en effet, tout programme serait exprimable par des formules, qui ne tiennent évidemment pas compte des adresses de la mémoire où les nombres-instructions doivent prendre place.

De plus, comme les formules peuvent s'établir sans faire appel à des propriétés particulières de la calculatrice employée, à part celle qu'elle est à trois adresses ([19]) et universelle, l'on démontrerait en même temps l'indépendence du programme par rapport à la calculatrice particulière qui doit exécuter les calculs.

Or, la réponse à notre question est relativement simple, si l'on fixe une fois pour toute, comme nous allons le faire, le rôle joué par l'adresse π' et par les adresses $A, B, \ldots, K, \ldots$ etc. par rapport aux nombres-instructions du type $\pi' \rightarrow A$, $\pi' \rightarrow B, \ldots, \pi' \rightarrow K, \ldots$ etc., que nous allons appeler « instructions immédiates ». Nous représentons par π' une adresse déterminée, p. ex. l'adresse 999.

Supposons que sur le ruban d'entrée de la calculatrice soient placés, l'un après l'autre, N nombres-instructions relatifs à un programme quelconque. Soit $100 \leq \varphi \leq 970$ une adresse de la mémoire telle qu'aucune des N adresses suivantes, par ordre croissant, ne coïncide avec une des valeurs que les indices du programme proposé peuvent prendre. Au début le contenu de l'adresse 999 doit être $\varphi + 1$.

Nous imaginons qu'aux adresses 970 et suivantes soit inscrit le bref programme d'entrée décrit plus loin, dont le rôle est le suivant :

Chaque nombre-instruction qui entre dans la machine est analysé: s'il s'agit d'une « instruction immédiate » elle est exécutée sur l'instant; autrement les nombres-instructions sont placés à des adresses consécutives à partir de $\varphi + 1$.

α. Le nombre-instruction qui suit sur le ruban est appéle a; s'il est nul, fin; autrement la suite sous β.	$\pi'' \rightarrow \alpha$ ([20]) $? \rightarrow a$ $(1 \dot{-} a) \cdot \Omega + (1 \cap a) \cdot \beta \rightarrow \pi$
β. Soit b le nombre formé par les 4 derniers chiffres de a. Si les chiffres 5-10ème de a sont 050999 la suite sous γ, autrement sous δ.	$\pi'' \rightarrow \beta$ $a \bmod 10000 \rightarrow b$ $a \dot{-} b \rightarrow c$ $c + 94999010000 \rightarrow f$ $f \bmod 10^{10} \rightarrow d$ $(1 \dot{-} d) \cdot \gamma + (1 \cap d) \cdot \delta \rightarrow \pi$
γ. Le nombre contenu en π' est à tranférer à l'adresse désignée per b; la suite sous α.	$\pi'' \rightarrow \gamma$ $\pi' \rightarrow \downarrow b$ $\alpha \rightarrow \pi$
δ. Le nombre a est à transférer à l'adresse désignée par π'. Augmentons de 1 le nombre contenu en π'. Recommençons sous α.	$\pi'' \rightarrow \delta$ $a \rightarrow \downarrow \pi'$ $\pi' + 1 \rightarrow \pi'$ $\alpha \rightarrow \pi$

([19]) Cette restriction n'est cependant pas indispensable pour les développements qui suivent.

([20]) Dans cette description pour des raisons de clarté, nous avons écrit π'' à la place de π' au début de tous les groupements (qui ont été indiqués par des lettres grecques).

Au moyen de l'artifice décrit, les nombres inscrits aux adresses $A, B, \ldots, K, \ldots$, avant le début du calcul, deviennent précisément les adresses où la première instruction du groupement correspondant a été inscrite: ce fait garantit le déroulement correct du programme lors des *sauts inconditionnés* (p. ex. $K \rightarrow \pi$) ou même *conditionnés.*

4.2. **Réalisation de principe de la codification automatique.**

Comme nos objectifs pour la programmation «en gros» ont été raisonnablement atteints, concentrons-nous sur le problème plus restreint de rendre directement accessible le formalisme à la calculatrice. Il s'agit de construire une espèce de machine à codifier. Il nous suffit, en tout cas, d'un téléscripteur pourvu d'un clavier à touches, dont chacune porte une des indications suivantes:

Variables: $a, b, \ldots, z, A, B, \ldots, Z, \downarrow a$, (21) $\downarrow b, \ldots, \downarrow z, \downarrow A, \downarrow B, \ldots, \downarrow Z, \pi, \pi', ?, \Omega$.
Opérations: $+, \cdot, \dot{-}, :, \rightarrow, \div$, mod, $\cup, \cap$.
Parenthèses: $(,)$.

Lorsqu'on abaisse une touche, on imprime le symbole correspondant sur une feuille de papier et en même temps on enregistre un nombre sur un ruban perforé ou magnétique. À chaque touche correspond biunivoquement un nombre entier fixé une fois pour toutes. À tout programme correspond alors une suite de nombres. Nous considérons cette suite comme donnée et nous nous proposons de déterminer une autre suite de nombres, à savoir la suite de nombres-instructions relatifs au programme donné. Il s'agit d'un problème arithmétique susceptible d'être résolu par la calculatrice même, par l'application du programme de la «codification automatique». D'autre part, comme nous devons disposer en tout cas d'un programme même pour codifier la formule du type $a\,op\,b \rightarrow c$, nous cherchons à tirer le plus grand profit de cette circonstance. Nous allons montrer dans les deux chapitres suivants que l'on peut codifier automatiquement des formules du type

$$(((a + b) \cdot c) : d) \rightarrow x$$

c. à d. formées d'une suite quelconque, (mais significative), de symboles comprenant des variables, des signes d'opérations et des parenthèses. De même nous codifierons automatiquement des formules du type

$$a \dot{-} b \cdot c + d : f \cdot g \rightarrow x \qquad \text{(où par } d : f \cdot g \text{ on entend } d \cdot f^{-1} \cdot g\text{)}$$

c. à d. représentant des polynômes à plusieurs variables, mis sous forme normale, chaque monôme étant formé d'un produit de facteurs ayant un exposant égal à $+1$ ou bien -1.

(21) Pour simplifier l'exposé, nous traitons ici et par la suite chaque $\downarrow a$, $\downarrow b$, etc., comme s'il s'agissait d'un seul symbole et non de deux groupés ensemble.

De plus, nous allons englober dans ces programmes le contrôle du fait que la suite des symboles de chaque formule a un sens ([22]), de façon à pouvoir signaler automatiquement si, p. ex., les deux signes d'opération $+$: se suivent immédiatement.

4.3. Correspondance entre symboles et nombres.

Entre le symbole S et le nombre c qui doit être enregistré sur le ruban d'entrée par le téléscripteur, nous choisissons la correspondance suivante:

$$c = 5 \cdot k(S) + r$$

en convenant de poser, comme nous l'avons fait déjà en 2.4:

$k = 0, 1, 2, 3, ..., 26, 27, ..., 52, 53, 998, 999, 1002, 1003, ..., 1026, 1027, ..., 1052, 1053$
si
$S = \pi, ?, a, b, ..., z, A, ..., Y, Z, \Omega, \pi', \downarrow a, \downarrow b, ..., \downarrow z, \downarrow A, ..., \downarrow Y, \downarrow Z.$

En outre

$$k = c_{op} = 1, 2, 3, 4, 5, 6, 7, 8, 9,$$

si respectivement

$$S = +, \cdot, \dot{-}, :, \rightarrow, \div, \text{mod}, \cup, \cap,$$

et

$$k = 6 \quad \text{si} \quad S = (,).$$

Pour obtenir la biunivocité entre c et S nous codifions la *nature* r d'un symbole de la manière suivante:

$$r = 0\ ,\ 1\ ,\ 2\ ,\ \ 3\ ,\ \ 4\ ,$$
$$\text{si}\quad S =)\ ,\ (\ ,\ \rightarrow\ ,\ \text{variable, opération.}$$

4.4. Codification automatique d'une suite d'opérations au plus binaires.

Proposons-nous de codifier automatiquement un programme quelconque, p. ex., la succession suivante de symboles (v. aussi 2.7)

$$h \cap a \rightarrow A \quad h \dot{-} a \rightarrow B \quad A \cdot x \rightarrow C \quad B \cdot y \rightarrow D \quad C + D \rightarrow u \quad u \rightarrow \pi$$

de façon à obtenir la succession de nombres désignés respectivement en (1.42) par

$$n_1, n_2, n_3, n_4, n_5, J.$$

Pour réduire ce premier exemple de codification automatique à l'essentiel, faisons abstraction de la partie du programme relative au contrôle du fait

---

([22]) Pour ce qui regarde le bilan des parenthèses et la succession de deux symboles, voir aussi [5].

que la succession de symboles a un sens. Posons d'abord, une fois pour toute:

$$m_1 = 10^{10},\ m_2 = 10^8,\ m_3 = 10^4.$$

α. Examinons le premier symbole c. Si $c = 0$ fin; autrement poursuivons sous β.

$$\begin{aligned} \pi' &\to \alpha \\ ? &\to c \\ c : 5 &\to C \\ \{(1 \div c) \cdot \Omega + (1 \cap c)\beta &\to \pi \end{aligned}$$

β. Examinons le deuxième symbole d. S'il est $\to$, la suite sous γ; autrement sous δ.

$$\begin{aligned} \pi' &\to \beta \\ ? &\to d \\ d : 5 &\to D \\ [(1 \div (D \div 5)) \cdot \gamma] + [1 \cap (D \div 5) \cdot \delta] &\to \pi \end{aligned}$$

γ. Soit f le troisième symbole. Calculons le nombre-instruction (qui est du type T, J, L ou I) et recommencons sous α.

$$\begin{aligned} \pi' &\to \gamma \\ ? &\to f \\ f : 5 &\to F \\ 5 \cdot m_2 + C \cdot m_3 + F &\to ? \\ \alpha &\to \pi \end{aligned}$$

δ. Examinons le troisième symbole g et le cinquième j.
Calculons le nombre-instruction et recommençons sous α.

$$\begin{aligned} \pi' &\to \delta \\ ? &\to g \\ ? &\to j \\ ? &\to j \\ g : 5 &\to G \\ j : 5 &\to J \\ C \cdot m_1 + D \cdot m_2 + F \cdot m_3 + J &\to ? \\ \alpha &\to \pi \end{aligned}$$

5. - Codification automatique des formules comprenant des parenthèses.

5.1. Une première solution du problème. Discussion.

Pour ce problème, M. Rutishauser a déjà développé une méthode cyclique de solution basée sur l'abaissement successif de l'ordre des parenthèses [6]. Dans ses formules on distingue les parenthèses rondes les plus intérieures (d'ordre 1), le crochets (parenthèses d'ordre 2), les accolades (parenthèses d'ordre 3) etc.

On obtient une solution cyclique (au point de vu du programme) du problème si l'on codifie tout d'abord les opérations contenues entre parenthèses rondes, remplaçant ensuite ces parenthèses par les adresses des résultats intermédiaires et abaissant de 1 l'ordre des parenthèses qui restent. On répète ce processus le nombre de fois indiqué par l'ordre maximum des parenthèses.

La brièveté et la simplicité du programme constituent les principaux avantages de cette méthode, qui offre cépendant quelques inconvénients:

5.11. La nécessité de communiquer à la calculatrice une formule complète avant que le travail de codification ne commence. En d'autres termes toutes les donnée du problème encombrent simultanément la mémoire interne, ce qui correspond à une diminution de la capacité de calcul de la machine digitale.

5.12. La nécessité d'explorer complètement la même formule plusieurs fois de suite pour en extraire les quelques données qui interviennent à chaque cycle conduit à une perte de temps, sensible surtout dans les machines moins rapides.

5.13. La nécessité d'employer des symboles différents pour les parenthèses d'ordres différents. En conséquence, si l'on veut employer un clavier pour communiquer les symboles, celui-ci doit posséder des touches pour des parenthèses ouvertes et fermées de chaque ordre jusqu'à un certain maximum au déla duquel une formule ne peut pas être communiquée à la calculatrice.

Nous avons utilisé, au contraire, une méthode de résolution du problème où les symboles constituant la formule sont communiqués les uns après les autres, dans l'ordre même dans lequel il sont écrits; suivant cette méthode la calculatrice commence déjà à codifier dès que le deuxième symbole a été communiqué.

La capacité de mémoire interne nécessaire pour les données peut être ainsi réduite à deux symboles consécutifs à la place d'une formule entière ([23]). Les données sont utilisées au fur et à mesure et il n'y a pas la perte de temps correspondant à leur recherche.

Enfin nous avons trouvé une méthode à l'aide de laquelle il devient inutile de communiquer à la calculatrice l'ordre d'une parenthèse, le machine pouvant automatiquement déduire une hiérarchie entre les parenthèses en partant de la structure des formules auxquelles elles appartiennent.

5.2. Conventions à observer pour l'écriture des formules.

5.21. Toutes les opérations sont des opération binaires.

5.22. Au point de vue de l'introduction de parenthèses, les opérations sont traitées indistinctement comme des opérations non associatives; p. ex. on écrira:

$$((a + b) + c) \rightarrow x \quad \text{au lieu de} \quad a + b + c \rightarrow x$$
$$((a \cdot b) + c) \rightarrow y \quad » \quad » \quad » \quad a \cdot b + c \rightarrow y$$
$$(((a + b) \cdot (c + d)) \cdot (e \div f)) \rightarrow z$$

([23]) Dans la description qui suit cette possibilité n'a pas été exploitée, car elle entraînerait une modification du programme d'entrée, sur lequel nous ne désirons plus revenir.

au lieu de

$$(a + b) \cdot (c + d) \cdot (e \div f) \rightarrow z.$$

5.23. Chaque formule contenant des parenthèses doit commencer par une parenthèse ouverte ([24]).

Il est visible qu'avec ces conventions la nature des opérations ne joue pas de rôle pour les règles d'emplacement des parenthèses.

5.3. Principe de la solution.

Soit une formule du type

$$(((a\ op_1\ b) op_2 (c\ op_3\ d)) op_4 ((f^0 op_5\ g) op_6\ h)) \rightarrow x \tag{5.31}$$

il s'agit de construire une suite de nombres-instructions correspondant, dans l'ordre indiqué ci-dessous, aux formules

$$\begin{aligned} f^0 op_5\ g &\rightarrow x_6 \\ x_6\ op_6\ h &\rightarrow x_5 \\ c\ op_3\ d &\rightarrow x_4 \\ a\ op_1\ b &\rightarrow x_3 \\ x_3\ op_2\ x_4 &\rightarrow x_2 \\ x_2\ op_4\ x_5 &\rightarrow x_1 \\ x_1 &\rightarrow x \end{aligned} \tag{5.32}$$

formules obtenues en éliminant successivement les parenthèses en commençant par celles qui sont situées le plus à l'intérieur. Les x_f sont les résultats intermédiaires du calcul. Nous remarquons que les indices k, attribués aux opérations pour les distinguer sont choisi dans (5.31) en ordre croissant à partir de la gauche. D'autre part, les indices f, ne servant qu'à distinguer entre eux les résultats intermédiaires, pourraient être choisis arbitrairement. Toutefois, nous les avons choisis de façon qu'ils constituent une progression décroissante, lorsque les formules sont écrites dans leur ordre définitif (5.32). Cette numérotation est justifiée par le fait que les f peuvent être calculés par récurrence à partir de la gauche dans une formule quelconque du type (5.31) en fonction de la disposition des parenthèses, comme nous allons le montrer plus loin.

5.4. Enumération des parenthèses par la fonction $F(n)$.

Soit n le numéro d'ordre des parenthèses et $r(n)$ leur « nature », définie par

$$r(n) = \begin{cases} 0 \text{ si la } n\text{-ième parenthèse est fermée} \\ 1 \text{ » » » » » ouverte.} \end{cases}$$

([24]) L'ensemble de deux conventions (5.23) et (6.11) facilite la fusion des programmes correspondants aux problèmes des chap. 5 et 6 en un seul.

Définissons une fonction $F(n)$ par récurrence par

$$(5.41)\quad F(0)=0 \qquad F(n)=\begin{cases} p+1 \quad \text{où} \quad p=\overset{n-1}{\underset{1}{\Sigma}} r(n), & \text{si} \quad r(n)=1 \\ F\{\operatorname{Min}[F^{-1}(F(n-1))]-1\}, & \text{si} \quad r(n)=0 \end{cases} \text{ (25)}$$

Dans l'exemple choisi (5.31), l'on aurait

	(	(	(	)	(	)	)	(	(	)	)	)
$n=$	1	2	3	4	5	6	7	8	9	10	11	12
$r(n)=$	1	1	1	0	1	0	0	1	1	0	0	0
$F(n)=$	1	2	3	2	4	2	1	5	6	5	1	0.

La fonction $F(n)$ est en outre susceptible d'une représentation graphique conformément à la fig. 4 (relative toujours au même exemple).

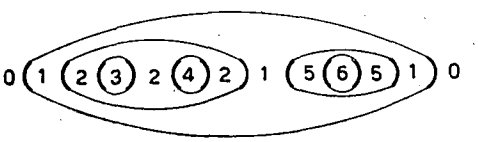

Figure 4.

Cette figure est construite à partir du schéma des parenthèses en réliant par une courbe fermée les parenthéses correspondantes au début et à la fin de la même expression. L'énumération successive à partir de la gauche des régions du plan délimitées par ces courbes est réalisée précisément par la fonction F (0 correspond à la région qui contient le point à l'infini).

Or, si nous désirons codifier automatiquement la formule (5.31) en l'explorant de gauche à droite, nous ne pouvons qu'obtenir successivement les formules :

$$\begin{aligned} a\ op_1\ b &\rightarrow x_3 \\ x_3\ op_2\ x_4 &\rightarrow x_2 \\ c\ op_3\ d &\rightarrow x_4 \\ x_2\ op_4\ x_5 &\rightarrow x_1 \\ f^0\ op_5\ g &\rightarrow x_6 \\ x_6\ op_6\ h &\rightarrow x_5 \\ x_1 &\rightarrow x \end{aligned}$$

(25) Cette formule s'exprime ainsi en mots : si la n-ième parenthèse est fermée, la valeur de $F(n)$ est celle de $F(h-1)$, h étant le plus petit argument tel que la fonction $F(h)$ ait la même valeur que $F(n-1)$.

qui coïncident avec celles de (5.32), sauf pour ce qui est de leur ordre de succession. Les indices f constituent donc nos principales inconnues. Toutefois la relation entre l'indice f et la fonction $F(n)$ est très simple, comme il ressort du tableau suivant.

	premier terme de l'opération	deuxième terme de l'opération	résultat
$)_n\ op\ \cdots$	$f = F(n-1)$	— —	— —
$)\ op\ (_n$	— —	$f = F(n)$	$f = F(n-1)$
$\cdot\cdot\ op\ \cdot\cdot)_n$	— —	— —	$f = F(n-1)$

L'indice n caractérise la dernière parenthèse communiquée à la machine. Le contenu du tableau peut en fait se résumer ainsi:

La valeur de f est le numéro d'ordre de la région (fig. 4) où se trouve l'opération relative à cet indice.

5.5 **Calcul explicite de $F(n)$.**

Comme la deuxième formule (5.41) est écrite sous forme implicite, nous allons la transformer convenablement. Remarquons que si l'on pose, lorsque $r(n) = 1$

$$X(n) = F(n) = p + 1 \quad \text{et} \quad Y_{X(n)} = F(n-1),$$

il résulte simplement de (5.41) pour le cas où $r(n) = 0$

$$X(n) = F(n) = Y_{X(n-1)}.$$

Le calcul explicite de $F(n)$ devient donc:

Si $r(n) = 1$:

On augmente p d'une unité.	$p + 1 \rightarrow p$	
L'ancien $F(n)$ est appelé $F(n-1)$.	$X \rightarrow Y$	
Le nouveau $F(n)$ est égal à p (à une constante m près).	$m + p \rightarrow X$	(v. groupement E au chap. 7)
Nous attribuons à $Y = F(n-1)$ l'indice $X = F(n)$.	$Y \rightarrow \downarrow X$	

Si $r(n) = 0$:

L'ancien $F(n)$ est appelé $F(n-1)$.	$X \rightarrow Y$	
Le nouveau $F(n)$ est la grandeur Y_X dont l'indice est $X = F(n-1)$.	$\downarrow X \rightarrow X$	(v. groupement F au chap. 7).

5.6. La matrice de choix.

Pour permettre de construire automatiquement une suite semblable à la suite (5.32) à partir d'une formule du type (5.31), le programme que nous allons déterminer doit être assez flexible pour pouvoir tenir compte de n'importe quelle structure de la suite de symboles donnée. En principe, la décision prise par la calculatrice, lors de la lecture du i-ème symbole, pourrait dépendre de tous les symboles lus jusqu'à ce moment. Mais en fait, il s'avère qu'à l'exception des décisions relatives aux parenthèses, les décisions à prendre par la calculatrice ne dépendent directement que de la nature des deux derniers symboles lus. Ceci simplifie beacoup le programme qui devient cyclique, nous obligeant toutefois de tenir compte de 25 variantes possibles, chacun des deux symboles pouvant être indifféremment l'un des symboles suivants :), (, →, variable, opération. Plusieurs de ces variantes constituent un contrôle ([26]). Dans la matrice ci-dessous, chaque variante absurde provoque le STOP. Les autres portent l'indication du groupement choisi pour la suite.

(5.61)

Nature s de l'avant-dernier symbole \ *Nature r du dernier symbole*	)	(	→	var.	opér.
)	G → π	Ω → π	J → π	Ω → π	T → π
(	Ω → π	C → π	Ω → π	P → π	Ω → π
→	Ω → π	Ω → π	Ω → π	Q → π	Ω → π
var.	H → π	Ω → π	Ω → π	Ω → π	W → π
opér.	Ω → π	I → π	Ω → π	R → π	Ω → π

5.7. Grandeurs auxiliaires.

La description complète du programme est constituée par les groupements A, B, ..., T, U, W du chapitre 7. $\downarrow t$, $\downarrow t+1$, $\downarrow t+2, \ldots, \downarrow +24$, sont les adresses réservées pour contenir les instructions qui correspondent aux éléments $(r, s) \approx 5 \cdot r + s$ ($r, s = 0, 1, \ldots, 4$) de la matrice (5.61).

([26]) Ce contrôle (v. aussi [5]) porte sur la succession immédiate de deux symboles, le bilan des parenthèses ($F(n) = 0$ seulement à la fin de la formule) et les conventions 5.2. Dans la suite nous avons exécuté tous ces contrôles à l'exception du contrôle 5.22 dont la réalisation aurait compliqué considérablement le programme. Toutefois, ce dernier contrôle peut être exercé automatiquement, grâce à l'interprétation numérique suivante : « La convention 5.22 est respectée si la fonction $F(n)$ assume une même valeur trois fois au maximum ».

Pour l'enregistrement provisoire dans la mémoire de nombres-instructions, avant leur sortie par ordre décroissant (v. groupement N), nous avons prévu les adresses $\downarrow m$, $\downarrow m+1$, $\downarrow m+2, \ldots$, etc., c. à d. les mêmes adresses qui ont servi précédemment à l'enregistrement de $F(n)$.

Les groupements A′, B′, ..., M′, N′, appartiennent par contre au programme pour la codification automatique étudiée au prochain chapitre.

6. - Codification automatique des polynômes mis sous forme normale.

L'interdiction (5.22) d'utiliser la propriété d'associativité du produit et de la somme, lorsqu'on écrit des formules contenant des parenthèses, oblige à l'emploi de plus de parenthèses que d'ordinaire. Pour remédier à cet inconvénient, nous voulons admettre une deuxième façon d'écrire les formules, à côté du modèle (5.31). Bien que nous nous limitions dorénavant aux quatre opérations arithmétiques, rien n'empêche en principe à envisager des types normalisés de formules contenant d'autres opérations élémentaires. Faisons la convention, comme en algèbre élémentaire, d'établir une échelle de priorité entre les quatre opérations prises deux à deux:

$$(+, \div) < (\cdot, :)$$

où le signe $<$ remplace ici l'expression « lie moins que ».

Ainsi une formule telle que

$$a + b \cdot c + d : f \cdot g \rightarrow x$$

acquiert un sens et nous sommes en mesure de réaliser une économie de parenthèses.

6.1. Conventions à observer pour l'écriture des formules.

6.11. Si le premier symbole n'est pas une parenthèse ouverte (dans ce cas on retomberait dans le problème du chap. 5) il ne faut employer aucune parenthèse dans toute la formule.

6.12. Le seules opérations utilisées sont

$$\rightarrow, +, \div, \cdot, : \qquad \text{c. à d. } k(op) \leq 5.$$

6.2. Principe de la solution.

Nous examinons quatre prototypes d'une expression polynômiale et les solutions correspondantes. Il est aisé de voir qu'il suffit d'employer un programme essentiellement cyclique où deux lettres seulement sont appélées à designer les résultats intermédiaires: la lettre X pour les résultats d'une multiplication ou d'une division ($:$) et la lettre S pour les résultats d'une addition ou d'une soustraction ($\pm$).

(6.2)

Formules données	*Nombres-instructions résultants*	*Désignation des groupements*
$a_1 \vdots_1 b \vdots_2 c_1 \rightarrow d_1$	$a_1 \rightarrow X$	G′
	$X \vdots_1 b_1 \rightarrow X$	H′
	$X \vdots_2 c_1 \rightarrow X$	H′
	$S + X \rightarrow S$	I′
	$S \rightarrow d_1$	N′
$a_2 \pm_1 b_2 \vdots_1 c_2 \rightarrow d_2$	$a_2 \rightarrow S$	F′
	$b_2 \rightarrow X$	M′
	$X_1 \vdots c_2 \rightarrow X$	H′
	$S \pm_1 X \rightarrow S$	I′
	$S \rightarrow d_2$	N′
$a_3 \pm_1 b_3 \pm_2 c_3 \rightarrow d_3$	$a_3 \rightarrow S$	F′
	$S \pm_1 b_3 \rightarrow S$	J′
	$S \pm_2 c_3 \rightarrow S$	J′
	$S \rightarrow d_3$	N′
$a_4 \vdots_1 b_4 \pm_1 c_4 \rightarrow d_4$	$a_4 \rightarrow X$	G′
	$X \vdots_1 b_4 \rightarrow X$	H′
	$S + X \rightarrow S$	I′
	$S \pm_1 c_4 \rightarrow S$	J′
	$S \rightarrow d_4$	N′

Les parties du programme relatives aux groupements

F′, G′, H′, I′, J′, M′, N′

sont cycliques par rapport à la succession de deux symboles d'opérations consécutifs (séparés naturellement par une variable), comme indiqué dans le tableau suivant:

Avant-dernière opération lue: \ *Dernière opération lue:*	$\vdots$	$\pm$	$\rightarrow$
$\vdots$	H′	H′I′	H′I′N′
$\pm$	M′	J′	J′N′

Le premier groupe de deux symboles est traité à part suivant le schéma

Deuxième symbole :

(Le premier symbole est une variable),	$\rightarrow$	$\begin{matrix}+\\ \div\end{matrix}$	$\vdots$
	D′	F′	G′

On remarquera que dans chaque programme (6.2) interviennent deux ou trois instructions qui ne seraient pas nécessaires si la codification était effectuée par un opérateur. Notre méthode devient cependant de plus en plus avantageuse à mesure que la longueur des formules crôit, car il intervient environ une istruction superflue au début et à la fin de chaque polynôme et une à chaque monôme.

6.3. **Contrôle et grandeurs auxiliaires.**

Pour qu'une succession de symboles ait un sens, il faut que les symboles d'opérations alternent avec les symboles de variables.

Nous allons employer dans la suite des grandeurs auxiliaires appelées g_i ($i = 1, 2, \ldots, 6$) qui sont des nombres-instructions incomplets, c. à d. ayant des zéros à la place des symboles en grasset :

$$g_1 \approx X \,\mathbf{op}\, \mathbf{V} \rightarrow X \qquad g_4 \approx S \,\mathbf{op}\, \mathbf{V} \rightarrow S$$
$$g_2 \approx S \,\mathbf{op}\, X \rightarrow S \qquad g_5 \approx S \rightarrow \mathbf{V}$$
$$g_3 \approx \mathbf{V} \rightarrow X \qquad g_6 \approx \mathbf{V} \rightarrow S$$

La partie pas en grasset est constante, indépendante des variables et des opérations de l'expression à codifier. P. ex., l'on a

$$g_1 = m_1 \cdot k(X) + k(X) = 10^{10} \cdot 51 + 51 = 0051\ 00\ 0000\ 0051.$$

Remarque. – Nous n'avons pas besoin d'un symbole spécial pour marquer la fin d'une série de formules. Il suffit d'employer le symbole de parenthèse fermée, grâce au groupement A du programme qui pourvoit automatiquement à l'arrêt de la calculatrice.

7. – Programme détaillé de codification automatique (v. fig. 5).

A. Le premier symbole c d'une formule est analysé. S'il s'agit d'une) ou bien de $\rightarrow$, ou si c'est un symbole d'une opération : STOP. Si c'est une variable la suite sous A' ; si c'est une (la suite sous B.

$$\pi' \rightarrow A$$
$$? \rightarrow c$$
$$c \bmod 5 \rightarrow r$$
$$[(r+1) \bmod 2] \cdot \Omega + [[1 \dot{-} (r \div 3)] \cdot A'] +$$
$$+ [[1 \dot{-} (r \div 1)] \cdot B] \rightarrow \pi$$

B. On pose $Y=0$ (à une constante m près), $p=1$ et l'on calcule $F(1)$ (v. en 5.4). La suite sous *C*.

$$\pi' \rightarrow B$$
$$m \rightarrow Y$$
$$1 \rightarrow p$$
$$m+p \rightarrow X$$
$$Y \rightarrow \downarrow X$$
$$C \rightarrow \pi$$

C. Le symbole suivant est analysé; soit s la nature du précédent et r celle de celui-ci. S'il est une (, la suite sous *E*. S'il est une), la suite sous *F*. Autrement la suite sous *D*.

$$\pi' \rightarrow C$$
$$r \rightarrow s$$
$$? \rightarrow c$$
$$c:5 \rightarrow k$$
$$c \bmod 5 \rightarrow r$$
$$[1 \cap (r \dot{-} 1)] \cdot D + (1 \cap r) \cdot E + (1 \dot{-} r) \cdot F \rightarrow \pi$$

D. Les natures des deux derniers symboles analysés contribuent à déterminer le prochain groupement à choisir (v. matrice (5.61)).

$$\pi' \rightarrow D$$
$$5 \cdot r + s + t \rightarrow \pi$$

E. Détermination de la valeur numérique de la fonction $F(n)$ pour une parenthèse ouverte (v. l'explication en 5.5). La suite sous *D*.

$$\pi' \rightarrow E$$
$$p+1 \rightarrow p$$
$$X \rightarrow Y$$
$$m+p \rightarrow X$$
$$Y \rightarrow \downarrow X$$
$$D \rightarrow \pi$$

F. Détermination de la valeur numérique de la fonction $F(n)$ pour une parenthèse fermée (v. l'explication en 5.5). La suite sous *D*.

$$\pi' \rightarrow F$$
$$X \rightarrow Y$$
$$\downarrow X \rightarrow X$$
$$D \rightarrow \pi$$

G. Si le nombre des parenthèses fermées déjà dénombrées excède celui des parenthèses ouvertes STOP; autrement la suite sous *C*.

$$\pi' \rightarrow G$$
$$[[1 \dot{-} (Y \div m)] \cdot \Omega] + [[1 \cap (Y \div m)] \cdot C] \rightarrow \pi$$

H. Détermination des 4 derniers chiffres (11-14ème) du nombre-instruction et inscription provisoire à une adresses calculée à partir de l'ordre de la dernière parenthèse analysée. La suite sous *C*.

$$\pi' \rightarrow H$$
$$n+Y \rightarrow \downarrow Y$$
$$C \rightarrow \pi$$

I. Détermination des 8 derniers chiffres (7–14ème) du nombre-instruction et inscription comme sous *H*. La suite sous *C*.

$$\pi' \rightarrow I$$
$$m_3 \cdot X + Y + n \rightarrow \downarrow Y$$
$$C \rightarrow \pi$$

J. Si le nombre des parenthèses fermée n'est pas égal à celui des parenthèses ouvertes STOP; autrement la suite sous *M*.

$$\pi' \rightarrow J$$
$$[1 \cap (X \div m)] \cdot \Omega + [1 \div (X \div m)] \cdot M \rightarrow \pi$$

M. Le dernier nombre-instruction de la formule (qui est un transfert) est calculé jusqu'au 11ème chiffre inclus. L'adresse provisoire du dernier nombre-instruction calculé est appelé *L*. La suite sous *N*.

$$\pi' \rightarrow M$$
$$5 \cdot m_2 + m_3 \cdot Y \rightarrow n$$
$$m + p \rightarrow L$$
$$N \rightarrow \pi$$

N. Le nombre placé à l'adresse *L* est enregistré sur le ruban de sortie. Diminuons *L* d'une unité. Recommencer sous *N* jusqu'à ce que tous les nombres calculés soient sortis ($L = 0 = p$ à la constante m près). La suite sous *C*.

$$\pi' \rightarrow N$$
$$\downarrow L \rightarrow ?$$
$$L \div 1 \rightarrow L$$
$$[1 \div (L \div m)] \cdot C + [1 \cap (L \div m)] \cdot N \rightarrow \pi$$

P. Détermination des 4 premiers chiffres du nombre-instruction; la suite sous *C*.

$$\pi' \rightarrow P$$
$$k \cdot m_4 \rightarrow n$$
$$C \rightarrow \pi$$

Q. Le dernier nombre-instruction de la formule avec parenthèses est complété jusqu'au 14ème chiffre et il est enregistré sur le ruban de sortie. La suite sous *A*.

$$\pi' \rightarrow Q$$
$$k + n \rightarrow ?$$
$$A \rightarrow \pi$$

R. Détermination des 7–10èmes chiffres du nombre-instruction. La suite sous *C*.

$$\pi' \rightarrow R$$
$$k \cdot m_3 + n \rightarrow n$$
$$C \rightarrow \pi$$

T. Contrôle du bilan des parenthèses comme sous *G*. Si le bilan est correcte, la suite sous *U*.

$$\pi' \rightarrow T$$
$$[[1 \div (Y \div m)] \cdot \Omega] + [[1 \cap (Y \div m)] \cdot U] \rightarrow \pi$$

U. Détermination des 6 premiers chiffres du nombre-instruction. La suite sous C.

$$\pi' \rightarrow U$$
$$Y \cdot m_1 + k \cdot m_2 \rightarrow n$$
$$C \rightarrow \pi$$

W. Détermination des 5ème et 6ème chiffres du nombre-instruction. La suite sous C.

$$\pi' \rightarrow W$$
$$k \cdot m_2 + n \rightarrow n$$
$$C \rightarrow \pi$$

A'. Le deuxième symbole d'une formule sans parenthèses est analysé. Si c'est une variable, une parenthèse ou bien une opération autre que $+$, $\dot{-}$, $\cdot$, $:$, $\rightarrow$, STOP. Si c'est $\rightarrow$ la suite sous D'; si c'est $+$ ou $\dot{-}$ la suite sous F'; si c'est $\cdot$ ou $:$ la suite sous G'.

$$\pi' \rightarrow A'$$
$$c : 5 \rightarrow h$$
$$? \rightarrow c$$
$$c : 5 \rightarrow k$$
$$c \bmod 5 \rightarrow r$$
$$[[1 \cap (k \dot{-} 5)] + (1 \dot{-} (k \dot{-} 5)) \cdot (1 \dot{-} (r \div 3))] \cdot$$
$$\cdot \Omega + [1 \dot{-} (r \div 2)] \cdot D' + [1 \dot{-} (r \div 4)] \cdot$$
$$\cdot [k \bmod 2] \cdot F' + [1 \dot{-} (r \div 4)] \cdot$$
$$\cdot [(k + 1) \bmod 2] \cdot G' \rightarrow \pi$$

B'. Un autre symbole est analysé. Si ce n'est pas une variable STOP. Si l'avant-dernier symbole analysé est $\rightarrow$ la suite sous N'; autrement sous C'.

$$\pi' \rightarrow B'$$
$$k \rightarrow j$$
$$r \rightarrow q$$
$$? \rightarrow c$$
$$c : 5 \rightarrow k$$
$$c \bmod 5 \rightarrow r$$
$$[1 \cap (r \div 3)] \cdot \Omega + [1 \dot{-} (r \div 3)] \cdot$$
$$\cdot \{[1 \dot{-} (q \div 2)] \cdot N' + [1 \cap (q \div 2)] \cdot C'\} \rightarrow \pi$$

C'. Un autre symbole est analysé. Si c'est une parenthèse ou une variable ou bien une opération autre que $+$, $\dot{-}$, $\cdot$, $:$, $\rightarrow$, STOP. Si l'avant-dernier symbole est $\cdot$ ou $:$, la suite sous H'. S_i l'avant-dernier étant $+$ ou $\dot{-}$, le dernier est $\cdot$ ou $:$, la suite sous M'; autrement la suite sous J'.

$$\pi' \rightarrow C'$$
$$k \rightarrow h$$
$$? \rightarrow c$$
$$c : 5 \rightarrow k$$
$$c \bmod 5 \rightarrow r$$
$$\{[1 \cap (k \dot{-} 5)] + [1 \dot{-} (k \dot{-} 5)] \cdot [1 \dot{-} (r \div 3)]\} \cdot$$
$$\cdot \Omega + [(1 + j) \bmod 2] \cdot H' + (j \bmod 2) \cdot$$
$$\cdot \{[(k + 1) \bmod 2] \cdot M' + (k \bmod 2) \cdot J'\} \rightarrow \pi$$

D'. Le troisième symbole est analysé. Si ce n'est pas une variable STOP; autrement la suite sous E'.

$$\pi' \rightarrow D'$$
$$? \rightarrow c$$
$$c : 5 \rightarrow k$$
$$c \bmod 5 \rightarrow r$$
$$[1 \cap (r \div 3)] \cdot \Omega + [1 \dot{-} (r \div 3)] \cdot E' \rightarrow \pi$$

E′. Calcul du nombre-instruction relatif à la formule explorée, qui est un transfert. La suite sous A.

$$\pi' \rightarrow E'$$
$$h \cdot m_3 + k \rightarrow ?$$
$$A \rightarrow \pi$$

F′. Détermination et sortie du premier nombre-instruction qui est du type g_6. La suite sous B'.

$$\pi' \rightarrow F'$$
$$h \cdot m_3 + g_6 \rightarrow ?$$
$$B' \rightarrow \pi$$

G′. Détermination et sortie du premier nombre-instruction, qui est $0 \rightarrow S$ et du deuxième nombre-instruction qui est du type g_3. La suite sous B'.

$$\pi' \rightarrow G'$$
$$998 \cdot m_3 + g_6 \rightarrow ?$$
$$h \cdot m_3 + g_3 \rightarrow ?$$
$$1 \rightarrow g$$
$$B' \rightarrow \pi$$

H′. Détermination et sortie d'un nombre-instruction du type g_1. Si le dernier symbole analysé est • ou : la suite sous B'; autrement sous I'.

$$\pi' \rightarrow H'$$
$$g_1 + j \cdot m_2 + h \cdot m_3 \rightarrow ?$$
$$(k \bmod 2) \cdot I' + [(k+1) \bmod 2] \cdot B' \rightarrow \pi$$

I′. Détermination et sortie d'un nombre-instruction du type g_2. La suite sous B'.

$$\pi' \rightarrow I'$$
$$g \cdot m_2 + g_2 \rightarrow ?$$
$$B' \rightarrow \pi$$

J′. Détermination et sortie d'un nombre-instruction du type g_4. La suite sous B'.

$$\pi' \rightarrow J'$$
$$j \cdot m_2 + h \cdot m_3 + g_4 \quad ?$$
$$B' \quad \pi$$

M′. Détermination et sortie d'un nombre-instruction du type g_3. La suite sous B'.

$$\pi' \rightarrow M'$$
$$j \rightarrow g$$
$$h \cdot m_3 + g_3 \rightarrow ?$$
$$B' \rightarrow \pi$$

N′. Détermination et sortie du dernier nombre-instruction de la formule sans parenthèses (qui est un transfert du type g_5). La suite sous A.

$$\pi' \rightarrow N'$$
$$k + g_5 \rightarrow ?$$
$$A \rightarrow \pi$$

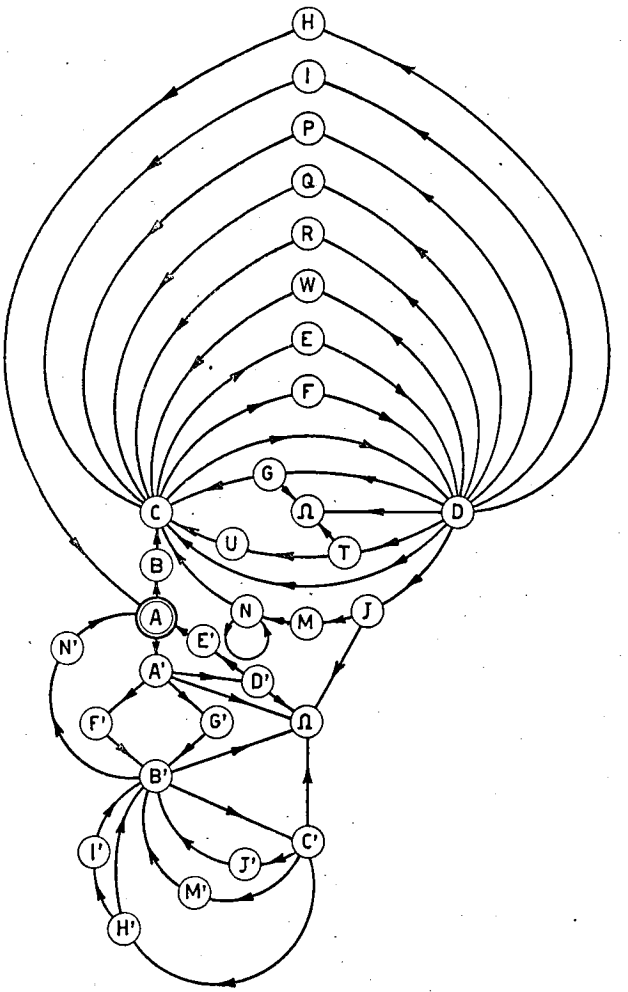

Figure 5.

8. - Relations avec la logique.

Parmi les opérations que nous avons choisies en 1.3, il y a une certaine redondance logique. P. ex. chacune des quatre opérations $\dot{-}$, $\div$, $\cup$, $\cap$ peut être exprimée à l'aide d'une seule d'entre elles et d'additions, comme le montre la réduction suivante:

$$a \div b \equiv (a \dot{-} b) + (b \dot{-} a)$$
$$a \cap b \equiv a \dot{-} (a \dot{-} b)$$
$$a \cup b \equiv b + (a \dot{-} b).$$

De même les opérations $\cdot$ et : pourraient être facilement réalisées par des sub-routines ne comportant que les opérations $+$ et $\dot{-}$. Les nombres

négatifs et fractionnaires d'autre part peuvent aussi être représentés, de plusieurs façons même, par des paires de nombres entiers non négatifs, de sorte que le fait de nous être limités à ces derniers n'entraîne pas une perte en généralité.

Dans le programme traité au chapitre précédent et spécialement à la fin des groupement A', B', et C' les critères de choix sont assez compliqués. Pour faciliter l'écriture des formules nous avons souvent fait appel à certaines correspondances biunivoques que l'on peut établir entre les opérations logiques fondamentales de *negation, conjonction et disjonction* d'une part et les opérations $\dot{-}$, $+$, et $\cdot$ de l'autre.

En effet, si l'on désigne par P et Q des propositions quelconques, par p et q des valeurs numériques entières non négatives et par r' et s des nombres ne pouvant assumer que les valeurs 0 ou 1, nous pouvons établir le tableau qui suit (où les notations logiques sont celles de [11]):

P, Q vraies	$p,\ q = 0$	$r',\ s = 1$
P, Q fausses	$p,\ q \neq 0$	$r',\ s = 0$
$\bar{P}$	$1 \dot{-} p$	$1 \dot{-} r'$
$\bar{\bar{P}}$	$1 \dot{-} (1 \dot{-} p) = p$	$r' = 1 \cap p$
$P \,\&\, Q$	$p \dot{+} q$	$r' \cdot s$
$P \vee Q$	$p \cdot q$	$1 \cap (r' + s)$.

Pour montrer une application de la correspondance entre la première et la troisième colonne du tableau précédent, considérons, p. ex., les critères de choix à la fin du groupement B', chapitre 7.

Appelons ω, n', c' respectivement les multiplicateurs de Ω, N', C'.

Le premier critère exige que $\omega = 1$ si, et seulement si $r \neq 3$; d'où

$$\omega = 1 \cap (r \dot{-} 3).$$

Le deuxième critère exige que $n' = 1$ *si* ce n'est *pas* $r \neq 3$ *et* en même temps *si* ce n'est *pas* $q \neq 2$; d'où

$$n' = [1 \dot{-} (r \dot{-} 3)] \cdot [1 \dot{-} (q \dot{-} 2)].$$

Le troisième critère exige que $c' = 1$ *si* ce n'est *pas* $r \neq 3$ *et* en même temps *si* $q \neq 2$; d'où

$$c' = [1 \dot{-} (r \dot{-} 3)] \cdot [1 \cap (q \dot{-} 2)].$$

BIBLIOGRAPHIE

[1] E. C. BERKELEY, *Giant Brains or Machines that Think*, John Wiley & Sons, New York, (1949).

[2] H. RUTHISHAUSER, A. SPEISER, E. STIEFEL, *Programmgesteuerte digitale Rechengeräte (elektronische Rechenmaschinen)*, Mitt. Nr. 2, « Inst. f. ang. Math. », ETH Zurich, (1951).

[3] M. V. WILKES, *Report on the Preparation of Programmes for EDSAC and the use of library of sub-routines*, « University Math. Laboratory Cambridge », (Sept. 1950).

[4] J. VON NEUMANN, *The general and logical theorie of Automata*, p. 1-41 of: *Cerebral Mechanisms in Behavior.* The HYXON Symposium, John Wiley & Sons, New York, (1951).

[5] K. ZUSE, *Über den allgemeinen Plankalkul als Mittel zur Formulierung schematisch kombinatorischer Aufgaben*, « Arch. Math. », 1, 441-449, (1948-49).

[6] H. RUTISHAUSER, *Über automatische Rechenplanfertigung bei Programmgesteuerten Rechenmaschinen*, Mitt. Nr. 3, « Inst. f. ang. Math. », ETH Zurich, (1952).

[7] A. M. TURING, *Computable numbers, with an application to the Entscheidungsproblem*, « Proc. London Math. Soc. », 2, s. 42, (1937), p. 230-265.

[8] A. P. SPEISER, *Entwurf eines elektronischen Rechengerätes*, Mitt. Nr. 1, « Inst. f. ang. Math. », ETH Zurich, (1950).

[9] D. KOENIG, *Theorie der endlichen und unendlichen Graphen*, Leipzig, (1936).

[10] H. H. GOLDSTINE, J. VON NEUMANN, *Planning and Coding for an Electronic Computing Instrument*, 3 vol., Institute for Advanced Study, Princeton, N. Y., (1947-48).

[11] D. HILBERT, W. ACKERMANN, *Grundzüge der theoretischen Logik*, Springer-V., Berlin, (1949).

RIASSUNTO

Vengono descritte, in primo luogo (Cap. 1), la struttura e l'organizzazione d'una calcolatrice numerica del tipo di quelle già in servizio. Riesce così possibile definire esplicitamente un programma ciclico fondamentale che traduce meccanicamente per mezzo di tale macchina ciò che si intende per « programma » (vedi 0.32) prima ancora di dovere specificare quali siano le istruzioni codificate. Fatta poi quest'ultima scelta, viene dimostrata l'universalità della calcolatrice in questione e vengono aggiunte alcune nuove istruzioni che si riveleranno utili in seguito. Dalla notazione usuale per i programmi si passa ad un'altra notazione (Cap. 2) il cui simbolismo viene poi giustificato dalla possibilità di potere scrivere ogni programma in tale linguaggio formale ricorrendo unicamente a nozioni logiche o algebriche (Cap. 3). Inoltre viene dimostrato (Cap. 4) che questo formalismo può essere reso accessibile alla calcolatrice mediante una telescrivente, stabilendo una volta per tutte una corrispondenza biunivoca fra simboli e numeri intieri. Le formule che traducono algebricamente un programma di calcolo possono contenere parentesi (Cap. 5) oppure doi polinomi scritti in forma normale (Cap. 6). In ambo i casi la calcolatrice stessa, grazie ad un programma fisso, indipendente cioè dalla natura particolare delle formule (Cap. 7) è messa in grado di eseguire i calcoli necessari per produrre le istruzioni codificate finali procedendo dalle formule che interpretano il metodo numerico prescelto. Infine (Cap. 8) viene messa in evidenza la superfluità logica di certe operazioni e vengono applicati alcuni metodi di aritmetizzazione del calcolo delle proposizioni.

SUMMARY

First of all (Chap. 1) the structure and organisation is described of a computer of the type of those already in use. A basic cyclic programme can then be explicitly defined, translating the definition of « programme » in terms of this machine (see 0.32), before specifying what the coded instructions are to be. Having chosen the latter, the universality of the machine is shown and some supplementary instructions useful later on are introduced. After passing from the usual programme notation to another (Chap. 2), the symbolism introduced in order to write the whole programme in formal language is justified, making use only of logical or algebraic concepts (Chap. 3). Furthermore, it is shown (Chap. 4) that this formalism can be made accessible to the machine by means of a teletyper, establishing once for all a one-one correspondence between symbols and integral numbers. The formulae expressing algebraically a computing programme may contain brackets (Chap 5) or polynomials in several variables put in the normal form (Chap. 6). In either case the machine itself, by means of a fixed programme independent of the particular nature of the formulae, can be made to perform the calculations necessary to produce the detailed coded instructions, from the formulae interpreting the numerical method envisaged. Finally (Chap. 8) the logical redundance of certain operations is printed out, and some arithmetisation procedures of propositional calculus is recalled.

ZUSAMMENFASSUNG

Wir beschreiben zunächst (Kap. 1) die Struktur und die Organisation einer Rechenmaschine vom Typ der bereits konstruierten. Wir können so ein zyklisches Grundprogramm explizit definieren (siehe 0.32), noch bevor wir spezifizieren, welches die kodifizierten Befehle sein müssen. Nachdem wir von der gewöhnlichen Bezeichnung der Programme zu einer anderen Bezeichnung (Kap. 2) übergegangen sind, rechtfertigen wir den eingeführten Symbolismus durch die Möglichkeit, jedes Programm in dieser formalen Sprache schreiben zu können, unter blosser Zuhilfenahme logischer oder algebraischer Bezeichnungen (Kap. 3). Weiterhin beweisen wir (Kap. 4), dass man diesen Formalismus der Rechenmaschine zugänglich machen kann mit Hilfe eines Fernschreibers, indem wir ein für alle Mal eine eineindeutige Zuordnung zwischen Symbolen und ganzen Zahlen festlegen. Die Formeln, die ein Rechenprogramm in algebraischer Weise übersetzen, können Klammern enthalten (Kap. 5) oder Polynome in mehreren Variablen in der normalen Form. (Kap. 6). In beiden Fällen können wir durch die Rechenmaschine selbst, dank einem festen Programm, das von der speziellen Natur der Formeln (Kap. 7) unabhängig ist. die für die Herstellung der endgültigen kodifizierten Befehle auf Grund der Formeln notwendigen Rechnungen, ausführen lassen, die die ins Auge gefasste numerische Methode darstellen. Schliesslich (Kap. 8) haben wir die logische Entbehrlichkeit gewisser Operationen hervorgehoben und Verfahren zu Arithmetisierung des Aussagenkalküls in Erinnerung gerufen.

TABLE DES MATIÈRES

CURRICULUM VITAE

CORRADO BÖHM, citoyen italien, né à Milan le 17-1-23. En 1941 il obtint son diplôme d'études secondaires au Licée scientifique « Vittorio Veneto » de Milan.

Entré en 1942 à l'Ecole Polytechnique de l'Université de Lausanne (Suisse) il en sortit fin 1946 avec le diplôme d'ingénieur électricien.

Il fut engagé en 1947 en tant qu'assistant à l'École Polytechnique Fédérale de Zurich de M. R. DUBS (Professeur d'Hydraulique et de Machines hydrauliques) pendant trois semestres et les trois suivants de M. E. STIEFEL (Professeur de Géometrie et Directeur de l'Institut de Mathématiques appliquées).

Pendant cette période, tout en approfondissant ses propres connaissances en mathématiques, il fréquenta un cours de spécialisation chez I. B. M sur les machines à cartes perforées. Il se familiarisa ensuite avec les analogues machines BULL.

En 1949 il fut envoyé a Neukirchen (Allemagne) pour étudier sur place la machine à relais construite par M. ZUSE, machine qui fut adoptée ensuite par l'École Polytechnique même.

Par ces machines il résolut également divers problèmes mathématiques: Tabulation de fonctions, calcul des différences, calcul des résidues, multiplication d'éléments de groupes finis, etc.