ISSN 0291 - 5413

Informatique fondamentale et Applications

Editeur
L. I. T. A. M.

Depositaire

G. Ambard

Sommaire

- Editorial: Informatique et théocratie. p. 1
- Une preuve directe de l'équivalence entre machine de Turing et machine à cases adressables. p. 7
- Autojection et compilateurs. Procédure formelle symbolique. p.50
- Le GRAFCET: Etudes et reflexions. p.66
- Aide à la construction de programmes BASIC sur systèmes Apple. p.78
- Vouzzavedibisar. p.90

J.-M. KNIPPEL

décembre 1983

ADRESSE POSTALE : FACULTÉ DES SCIENCES DE LUMINY
MATHÉMATIQUE-INFORMATIQUE - LITAM - BAT. TPR2 - 9ème ETAGE
CASE 901 - 70, ROUTE LÉON LACHAMP - 13288 MARSEILLE CEDEX 9 - ☎ (91) 41.01.40 POSTE 3294 - 31

Editorial

j. fr. Lehmann

Informatique & théocratie

L'informatique est le théâtre d'une prolifération de débats, tous qualifiés par leurs protagonistes d'essentiels, voire de cruciaux.

On ne peut s'empêcher de ressentir un certain malaise devant la multiplication de ces "problèmes de l'heure", ainsi nommés pour leur urgence, et dont on finit par se demander si en réalité, ils ne sont pas de l'instant.
Distinguer dans le foisonnement des questions posées, celles dont la portée est générale, de celles qui n'ont qu'un intérêt secondaire ou épisodique, constitue un exercice de plus en plus délicat. Et pourtant, si stérile que cela puisse paraître quelquefois, il n'est pas raisonnable de se tenir à l'écart. Notre espoir est de ne pas ajouter à la confusion générale.

Le domaine fluctuant par excellence, est celui des langages. Sa nature même lui permet d'être le siège d'une espèce de querelle des anciens et des modernes qui semble pouvoir, à la fois, être permanente, et constamment renouvelée dans ses termes.

Nous n'embrasserons pas un tel sujet dans toute son ampleur: ce serait le meilleur moyen d'arriver à n'en rien dire. Un aspect particulier nous semble digne de la plus grande attention, celui du climat quasi-sectaire dans lequel se développent ces querelles que nous évoquions plus haut. Il est clair que la passion excessive qui anime tous ces amoureux de La Langue, contribue, par son caractère désagréable et lassant, à décourager un grand nombre, et en même temps à frayer le chemin à ceux qui souhaitent mettre enfin un peu d'ordre dans cette anarchie insupportable.

Ce dernier point nous parait le plus intéressant; à nos yeux, il y a là, une véritable lame de fond. Non que nous considérions pour notre part qu'un problème réel soit posé, mais au contraire, parce que notre conviction se précise, qu'une volonté puissante existe, visant à nous persuader, que, quelque part, l'intolérable a été atteint.

Ainsi va l'Histoire, et son accélération est telle, que le bébé, à peine sorti de ses langes, voit déjà son espace futur décidé, codifié, et bien entendu, ses fréquentations selectionnées.

Pour les chevaliers de la rigueur, le pire n'est pas d'évoluer dans une tour de Babel. La multiplicité des sectes n'empêche pas l'avènement des églises, au contraire. Et puis, le danger ne vient ni de ces classiques, psalmodiant leurs extatiques formules dont l'une pourrait être, "Ach, Vordran zera doujours Vordran"; ils feront de très bons orthodoxes. Ni de ces modernes non plus, fébriles épileptiques du nouveau "polyglottisme". Somme toute, les uns et les autres ne sont que des tenants de l'inexprimable, des esthètes, et leur mysticisme, bien que peu raffiné, pourra, peut-être, leur permettre d'entendre l'appel de la foi véritable.

Le mal est ailleurs: parler petit-nègre, là est la corruption suprême, le vice rédhibitoire. Partout, le bruit se répand, que même selon l'aveu d'éminents universitaires américains(tous,combien?), celui qui, par malheur, a connu comme premier langage Basic, a acquis de telles habitudes, que le Nirvana informatique lui sera, à tout jamais inaccessible. Peut-on ignorer de telles homélies, lorsque c'est de la Jerusalem même de l'informatique qu'elles nous parviennent?

Ici de distingués théologues soufflent dans la même trompette, et organisent un concert de lamentations en récitant des litanies, qui, dans l'art sacré, tiennent lieu de démonstration.

De toutes les déviances, la plus détestable, est évidemment celle dont l'audience est la plus large. En outre, y a-t-il meilleure stratégie unificatrice que celle de la Guerre Sainte? C'est un devoir, pour asseoir le nouveau règne, que de désigner l'hérétique, en lui livrant un combat sans merci, pour son bien et le salut de tous.

On voit d'ici le magister, considérer ce peuple d'amateurs, livrés à la débauche sur leurs claviers, faisant "n'importe quoi", selon l'expression consacrée, se vautrant dans la turpitude du GOTO, et de l'absence de structuration; et on l'entend soupirer:"Pardonnez leur, car ils ne savent pas ce qu'ils font".

Dira-t-on assez l'état de dégradation effrayante, de ces centaines de milliers de personnes, qui, en quelque sorte, forment une sorte de continent profane là où l'avènement du sacré est imminent.
On ne sauve pas Sodome et Gomorrhe: il faut é-li-mi-ner; ce sera le nouveau tube.

Mais trêve de balivernes; l'art qui consiste à présenter des problèmes fictifs, puis ensuite à prendre des mesures qui sont dirigées vers d'autres fins, est vieux comme le monde. Cette technique, dite du bouc émissaire, est fréquemment fondée sur une subtile permutation des buts et des moyens.

Qui conteste qu'une place de choix dans l'enseignement et plus généralement la formation, soit réservée à l'esprit de rigueur et de méthode?

C'est une bien curieuse démarche que de situer le mal dans un soi-disant désordre, dans tout ce qui relève de processus hésitants et non structurés a priori. Comme si la pensée dont le mode est celui des approximations successives, était une piètre pensée. Comme si le bien ne pouvait surgir qu'à travers un filtrage rigide des modes d'apprentissage ne laissant passer que ce qui, au préalable, aura été décla-

ré correct. Ceux qui préconisent de telles médecines, quelle compétence particulière ont-ils dans ce domaine si délicat qu'est la transmission des connaissances?

Est-il souhaitable ou non que l'accès aux machines soit rendu aisé, quasi-immédiat, grâce à l'existence de langages imparfaits à maints égards, mais puissamment interactifs, très largement diffusés, et d'un très faible coût?

Et surtout, faut-il accorder du crédit aux allégations selon lesquelles, les utilisateurs auraient été, à ce point abrutis, que les limites de Basic ne pourraient leur apparaître, alors qu'au contraire, l'impossibilité de passer des paramètres, la difficulté de lecture des programmes, sont à elles seules, la source de nombreuses questions extrêmement formatrices?

De même qu'on veut nous expliquer si brillamment que la pratique de Basic rend bête, on nous exposera que la pratique de Pascal, ou de tout autre langage structuré, rend intelligent. Ce n'est pourtant pas à des informaticiens qu'il faudrait expliquer que parler dans une langue, c'est autre chose que de parler de cette langue. Si dans une certaine mesure, l'informatique est la science de la commodité, il serait autant souhaitable d'indiquer comment elles se construisent, combien elles peuvent être diversifiées, comment leur compatibilité n'est pas toujours assurée, plutôt que de rechercher des solutions-miracles bien improbables.

Dans un autre domaine, en mathématiques, l'académisme a voulu également administrer ses bienfaits au vulgum pecus. Là aussi, tout n'était que gabegie: les définitions incorrectes, les schèmes déductifs trop flous, les représentations trop concrètes, et de ce fait, lamentablement particulières; on faisait n'importe quoi. Aujourd'hui, la lecture du programme de terminale C, montre à chacun l'étendue du chemin parcouru; on était déjà intelligent en faisant des mathématiques; depuis on est devenu génial; tous les étudiants de DEUG A vous le diront.

Saura-t-on faire entendre raison à l'académie?
Je propose la démarche suivante: étant donné que le but essentiel de l'académie est avant tout d'exister, que certes, l'académisme qu'elle prone n'est pas indifférent, mais qu'il ne constitue qu'un moyen, même s'il est présenté comme un but; que d'autre part s'il est possible d'envisager des modifications des moyens, il ne faut par contre nullement douter de l'intangibilité du but,
SUGGERONS QUE, il n'est pas sage de s'opposer à l'instauration de l'académie d'informatique, vu sa puissance et notre extrême faiblesse,

il est bon de contribuer à précipiter vers ces hautes sphères, ceux qui sont assoifés d'honneurs, car il n'est pas rare de les voir entendre ce que repu veut dire,

il est envisageable de permettre à des enseignants et des chercheurs compétents de telles ascensions, car il peut quelquefois se produire des contradictions interessantes et profitables,

il est mauvais de favoriser d'une quelconque façon l'accès à ces lieux,des ayatollahs, tout le monde sait pourquoi .

(Ce dernier point est le plus délicat, mais c'est aussi le plus nécessaire. Si vous n'y prenez garde, ce sera comme au conservatoire de musique: vous ne jouerez du pipo, qu'après votre inscription en bonne et due forme à l'institution, et notamment au cours de solfège.)

J.Ph.Lehmann

UNE PREUVE DIRECTE DE L'EQUIVALENCE ENTRE MACHINE DE TURING ET MACHINE A CASES ADRESSABLES.

REMARQUES SUR LA NOTION DE COMMUTATION

(J.Ph.Lehmann)

C.R.:Subject classification informatics
4.20 5.20 5.26 5.29 6.30 6.32

Résumé:

A l'époque de Turing, l'intuition du calculable se manifestait de diverses façons. La préoccupation essentielle était de construire une ou plusieurs bonnes définitions d'une notion imprécise.
Aujourd'hui l'intuition se développe à travers une pratique, celle de l'utilisation des ordinateurs, de plus en plus répandue.
Dans cet article, on établit un lien direct, entre un ordinateur très simple, mais qui mérite néammoins son nom, et la machine de Turing. On ne passe par l'intermédiaire d'aucune autre définition du calculable.

Au passage, on examine dans le détail, les caractéristiques d'un langage de programmation et de sa sémantique, du point de vue structurel et logiciel.

La notion d'état dans une machine de Turing, est éclairée à travers la notion de commutation intimement liée aux langages de programmation.

Plan de l'exposé

PRELIMINAIRES p.10à11

Iere PARTIE p.12à49

-A- LE LANGAGE M.A.C. p.12à17

-La forme F_o du langage p.13
-Structure de la machine: mémoire centrale, mémoires externes, registre spécial u p.13
-Commutation implicite et explicite p.14
-Sémantique du langage: les formes et les lieux, les transferts, les passages de formes à formes,etc... p.14à17

-B- REDUCTION STRUCTURELLE ET LOGICIELLE DE LA M.A.C. p.18à39

I SUPPRESSION DU CARACTERE "" p.18à19

II REDUCTION DU NOMBRE DES FILES A 1 p.19à23

- Initialisation p.19à21
1) Simulation de $Ca_i := Cf_N$ p.21
2) Simulation de Av f_N p.21à23
3) Simulation de $Cf_N := Ca_i$ p.23

III REDUCTION DU NOMBRE DES OPERATIONS MEMOIRE CENTRALE p.24à27

1) Suppression de Vers e_i p.24
2) Suppression de Si Cu=1 Vers e_i p.24
3) Suppression de $Ca_i := \bar{0}$ p.25
4) Suppression de $Ca_i := Ca_j$ p.26
- La forme F_1 du langage p.27

IV REDUCTION DU NOMBRE DES FORMES A DEUX p.27à32

- Qu'entend-on par"machines équivalentes"? p.27à29
1) Réalisation du schéma $Ca_i := Ca_i + 1$ p.30
2) Réalisation du schéma $Ca_i := Cf$ p.30à31
3) Réalisation du schéma $Cf := Ca_i$ p.31
4) Réalisation des schémas Av et Ar p.31

V REDUCTION DU NOMBRE DES CASES DE LA M.C. A UN p.32à36

- Problèmes de codage p.32à33
1) Réalisation du schéma $Ca_i := Ca_i + 1$ p.33à34
2) Réalisation du schéma Si $Ca_i = \bar{0}$ Vers e_i p.34
3) Réalisation du schéma $Ca_i := Cf$ p.35
4) réalisation des schémas Av et Ar p.35
- La forme F_2 du langage p.36

VI SUPPRESSION DE L'OPERATION $\bar{a}$ p.36à38
1) Réalisation de $\bar{a}$ p.37
2) Réalisation de SiCa=$\bar{O}$Vers e_i, Ca:=Cf,Cf:=Ca p.37
3) Réalisation de Av et Ar p.37
- La forme F_3 du langage p.38

-C- FAITS DE LIMITATION p.39à48

I LA QUESTION DU BORNAGE DU NOMBRE DES ETIQUETTES p.39à44
- Remarques p.39
1) Un problème particulier p.39à42
2) Remarque sur la machine universelle p.43à44
II CARACTERE MINIMAL DE LA FORME F_3 p.44à48
1) Précisions p.44à45
2) caractère minimal p.45à48

2eme PARTIE

La démonstration d'équivalence entre la forme réduite du langage F_3 et la machine de Turing, sera exposée dans le prochain numéro de ce bulletin, ainsi que quelques remarques au sujet de la commutation.

PRELIMINAIRES

On sait, qu'à partir d'intuitions distinctes, plusieurs définitions de la calculabilité ont vu le jour: fonctions récursives, machines de Turing, systèmes de Post....
L'unité de ces différents points de vue s'exprime à travers les démonstrations d'équivalence de ces définitions, et renforce le crédit accordé à la thèse de Church.
Par ailleurs, à cause de l'utilisation croissante des ordinateurs, il nous semble que l'intuition de ce qu'est le calculable, provient, pour le plus grand nombre, de la pratique qui s'acquiert sur les machines. Qu'un lien clair et direct soit établi entre ces deux domaines intuitifs, l'un se développant sur le plan abstrait, l'autre sur un terrain pratique, n'est peut-être pas sans intérêt.
Certes, désormais, l'étudiant préoccupé des questions de l'informatique fondamentale, se convainc rapidement que par exemple, une machine de Turing, permet de passer d'une base de représentation des nombres à une autre, de réaliser l'addition, la multiplication, etc... . Bien entendu, ces points de suspension sont censés traduire le fait, qu'ainsi tout le domaine du calculable est engendré. Or il n'y a pas là, la moindre déduction, même informelle, mais en fait une définition d'un terme jusque là demeuré imprécis. De ce fait l'unité totale qui relie un ordinateur quelconque à la machine de Turing, forme une question subrepticement éludée.
Plusieurs auteurs se sont préoccupés de cette question, notamment Minsky(1). Dans l'ouvrage cité en référence, un chapitre est consacré à exhiber des modèles d'ordinateur, et à montrer leur équivalence avec les autres formulations de la calculabilité. Néammoins, ces modèles restent idéals en grande partie, quoique leur structure soit très voisine de celle d'un ordinateur : par exemple, les registres des machines présentées sont de capacité illimitée. En outre la démonstration de l'équivalence, Turing-calculable/"ordinateur"-calculable, éclatée dans l'ouvrage, impose au lecteur, en fait, un cheminement long et assez complexe.

Pour notre démonstration, nous tirerons notre modèle d'ordinateur, de l'ouvrage cité en référence de Nolin(2). Il s'agit de la machine à cases adressables, que nous noterons dans ce qui suivra M.A.C.. L'intérêt de ce modèle est de représenter un ordinateur, certes rudimentaire, mais qui, d'une part, ne présente aucun caractère idéal, en ce sens qu'il serait tout à fait possible de le construire, et d'autre part s'impose à l'esprit du praticien, à l'évidence comme un ordinateur.
La démarche suivie par L.Nolin est la suivante: après avoir introduit les fonctions récursives, on présente un très simple calculateur, en fait très voisin d'une machine de Turing binaire, et on établit l'équivalence de puissance. Puis on accroit progréssivement la puissance du calculateur, en lui adjoignant plusieurs files, en enrichissant son alphabet, enfin en aboutissant à la M.A.C., véritable ordinateur avec mémoire centrale. On établit alors l'équivalence de puissance de ce dernier dispositif avec les fonctions récursives, donc avec la machine de Turing.
La pratique de l'enseignement nous a convaincus, que l'intuition du calculable à travers le formalisme des fonctions récursives, ne se développe pas toujours harmonieusement, sauf à s'adresser au public logicien. Les algorithmes "à plat" représentés par les fonctions récursives, dévoilent difficilement leurs liens, pourtant bien réels, avec les algorithmes décrits dans les langages de programmation les plus courants. En outre un des concepts centraux de l'algorithmique réside dans la notion de succéssion d'actions, nous utiliserons le terme de commutation, et le formalisme des F.R. se prête mal à le mettre en évidence. La démonstration qui suit suggère qu'il n'en est pas de même avec les machines de Turing. Si, bien évidemment, elles ne présentent pas les commodités des ordinateurs, il semble qu'on puisse les rapprocher assez aisément de ces derniers.
Nous procèderons de la façon suivante:

Partant d'une M.A.C. évoluée, nous ramènerons l'ensemble des instructions disponibles à un corps minimal, et établirons alors l'équivalence avec la M.T.(machine de Turing). Ceci nous offrira l'occasion de développer quelques remarques sur la notion de commutation.

Ière PARTIE

- A -

LE LANGAGE DE LA M.A.C.

- B -

REDUCTION STRUCTURELLE ET LOGICIELLE DE LA M.A.C.

- C -

FAITS DE LIMITATION

- A -

LE LANGAGE DE LA M.A.C

$Ca_i := Ca_j$	(1)
$Ca_i := \overline{0}$	(2)
$Ca_i := Ca_i + 1$ (k)	(3)
Si $Ca_i = \overline{0}$ vers e_i	(4)
Si Cu $= \overline{1}$ vers e_i	(5)
vers e_i	(6)
$Ca_i := Cf_n$	(7)
$Cf_n := Ca_i$	(8)
$Cf_n :=$ ""	(9)
Av f_n	(10)
Ar f_n	(11)
Si $Cf_n =$ "" vers e_i	(12)

La forme F_o

Précisons la sémantique attachée à ce langage:

La mémoire centrale

On dispose d'un ensemble fini de K lieux. Ces lieux sont décrits par leurs noms, a_i, a_j, etc....
Dans ces lieux, peuvent être déposées k formes distinctes, et uniquement celles-là. Lorsqu'on considère un lieu, avant d'exercer une action, ou après qu'elle ait été exercée, il s'y trouve exactement une de ces k formes. Ces formes sont décrites par leurs noms, $\overline{0}$, $\overline{1}$, $\overline{2}$,....., $\overline{k-1}$.

Les mémoires externes

On dispose de N files externes, qui constituent un autre type d'ensemble de lieux. Chacun de ces ensembles est une suite de lieux, le nombre de ceux-ci étant fini, mais non fixé à l'avance. Chacun de ces lieux peut, dans les mêmes conditions que précédemment, détenir k+1 formes.
Les files sont décrites par leurs noms, f_1, f_2,...., f_N.
Les lieux dont la suite forme une file n'ont pas de nom.
Les formes déposées sont décrites par leurs noms, $\overline{0}'$, $\overline{1}'$,..... $\overline{k-1}'$, "" .

Le registre u

Il existe un lieu spécial, u, dont le rôle est précisé après.

Maintenant que le support a été décrit, il nous faut indiquer les actions susceptibles de s'exercer sur lui.
Une action élémentaire est décrite par une quelconque des 12 instructions, indiquées en tête de la page précédente. Pour être plus précis, il s'agit en fait de schémas d'instructions. Avant d'examiner la sémantique de chaque instruction, nous nous intéresserons au déroulement de la suite des instructions que constitue un programme:

La commutation

Il s'agit d'indiquer comment les actions élémentaires s'enchainent les unes aux autres; il y a deux types d'enchainement, l'un implicite l'autre explicite: lorsqu'une instruction n'est pas du type (4), (5), (6), (12), on sous-entend qu'après exécution de l'action élémentaire correspondante, suivra l'action décrite par l'instruction située immédiatement après elle. Nous appellerons cette commutation, la commutation-liste, puisqu'elle est inscrite en quelque sorte dans la liste ordonnée des instructions.
Lorsque les instructions appartiennent à l'un des types cités plus haut, la commutation reste implicite(commutation-liste), pour les types (4), (5), (12), quand les tests décrits dans chacun sont reconnus faux. Dans le cas contraire, et aussi pour le type (6), l'action qui doit suivre est celle identifiée en marge par l'étiquette e_i. Il s'ensuit que chaque fois qu'une instruction de l'un des types cités, apparait dans un programme, il doit figurer, quelque part dans le programme, une instruction identifiée par l'étiquette correspondante, et ceci de façon unique.

Sémantique des instructions

(1) $\boxed{Ca_i := Ca_j}$

La forme présente dans a_j, immédiatement avant qu'on ne rencontre cette instruction, est recréée dans a_i, sans être modifiée dans a_j.

(2) $\boxed{Ca_i := \bar{0}}$

La forme $\bar{0}$ est déposée dans a_i.

Remarque: on peut observer que tout ce qui est demandé aux formes manipulées, c'est d'être différentes les unes des autres. Ainsi il est indifférent qu'on ait choisi ici celle de nom $\bar{0}$. Toute autre aurait également convenu.

Ce qui est important, c'est que k soit supérieur ou égal à 2: il ne faut pas perdre de vue que, dans ces opérations, ces formes auxquelles nous donnons des noms, peuvent précisément être elles-mêmes des noms lorsqu'on souhaite les interpréter; par exemple des noms de nombres, de la façon la plus courante, ou de toutes autres sortes de choses. Et ceci serait hors de portée si l'on ne disposait pas, au moins de deux formes.

(3) $\boxed{Ca_i := Ca_i + 1\ (k)}$

Considérons l'application suivante, de l'ensemble des formes dans lui-même:

$$\bar{0}\ \bar{1}\ \bar{2}\ \ldots\ldots\ldots\ \overline{k-1}$$

$$\bar{1}\ \bar{2}\ \bar{3}\ \ldots\ldots\ldots\ \bar{0}$$

L'opération décrite en(3), consiste à remplacer la forme présente dans le lieu a_i par son image dans l'application ci-dessus. De plus, le dispositif est muni d'un lieu spécial, u, qui reçoit la forme $\bar{1}$ ou $\bar{0}$, selon que l'image est $\bar{0}$ ou non.

Remarque: Il nous parait indispensable de noter que la forme du schéma d'instructions (3), présente, à coté d'une grande commodité, un risque de confusion: il n'est pas douteux que la notation choisie évoque le calcul, et plus précisément l'arithmétique modulo k. Cette impression se trouve encore renforcée par le choix des noms des formes qui sont quasiment ceux des nombres, et par le rôle attribué à u, celui d'un indicateur de débordement.
Ainsi s'il s'agit de décrire un calcul sur les nombres on comprend la commodité présentée par ce genre d'instructions. Mais en même temps, cette interprétation est beaucoup trop restrictive: l'action élémentaire qui est ici en cause, a une portée beaucoup plus large qui consiste à pouvoir substituer à toute forme, toute autre, quitte à opérer plusieurs fois.
Dans ces conditions, la sémantique du schéma (3), nous parait bien plus correctement décrite, par le procédé indiqué plus haut, savoir, une permutation de l'ensemble des formes; il est possible d'ajouter que la nature des permutations convenables n'est pas indifférente, et que leur caractéristique doit être de n'avoir aucun point fixe; le nombre de celles-ci est $(k-1)!$.

(4), (5), (6), (12).

la sémantique de ces schémas est décrite plus haut, dans le paragraphe relatif à la commutation.

(7) $\boxed{Ca_i := Cf_n}$

Comme on l'a vu plus haut, les lieux dont la suite constitue la file f_n ne sont pas nommés; il faut donc sous-entendre, pour attribuer un sens au schéma (7), que dans la structure même de la machine décrite, il existe un moyen de pointer la file de

quelque façon; ce moyen est l'index. Il y en a N, chacun attaché à une file.

Considérons alors la bijection suivante:

$$\overline{0}\cdot\ \overline{1}\cdot\ \overline{2}\cdot\ \ldots\ldots\ldots\ldots\ \overline{k-1}\cdot$$

$$\overline{0}\quad\overline{1}\quad\overline{2}\quad\ldots\ldots\ldots\ldots\quad\overline{k-1}$$

L'opération décrite en (7), consiste à remplacer la forme présente dans le lieu a_i par l'image de celle contenue dans le lieu pointé de la file, dans l'application décrite ci-dessus.

Remarque: Le problème qu'on rencontre ici, est celui de l'interfaçage. Il faut bien que les formes soumises au calcul, puissent être transmises au mécanisme central, et de même les formes résultantes doivent pouvoir transiter en sens inverse. Dans la réalité les différents supports qui interviennent, n'ont aucune raison d'être semblables entre eux; et même s'il en était ainsi, les interprétations attachées aux mêmes formes sur des supports distincts peuvent être distinctes.
Dans ces conditions, la forme du schéma (7), bien que très voisine dans son allure de celle du schéma (1), gagne plus, sur le plan sémantique à être interprétée en termes de réalisation d'une application, comme cela a été le cas pour le schéma (3). Les processus qui sont ici en jeu, sont ceux du calcul symbolique, et dépassent le cadre limité de l'opération courante de lecture-écriture en mémoire centrale.
De même que le schéma (3) fait référence à une opération de transformation des symboles, le schéma (7) doit être considéré de manière identique; on voit ainsi, tout naturellement se déssiner, la notion de calculateur d'échanges.
Notons pour terminer cette remarque, le caractère inessentiel de la bijection choisie, qui n'a été retenue que pour sa forme canonique.

(8) $\boxed{Cf_n := Ca_i}$

Les commentaires précédents, se transposent immédiatement au présent schéma, sans particularité nouvelle.

(9) $\boxed{Cf_n := ""}$

Tout processus de calcul demande à être initialisé et à être clôturé. L'absence de signes est une notion qui est dénuée de sens dans les mécanismes que nous étudions, aussi est-il bien commode de disposer de marques spéciales, pour indiquer ces choses bien spéciales que sont les informations sur les informations, par exemple pour signifier que le calcul va commencer, qu'il est terminé, que toutes les données ont été fournies, ou qu'on va précisément les transmettre, etc.....

Le schéma (9) joue ce rôle: il existe une marque spéciale "", hors alphabet, qu'on peut à tout moment configurer dans une file quelconque, là où elle est pointée.

(10), (11) | Av f_n, Ar f_n |

Les actions élémentaires ici décrites sont celles qui consistent à déplacer l'index pointant sur la file f_n en un lieu donné vers le suivant immédiat (Av f_n) ou vers le précédent immédiat (Ar f_n). Il n'y a aucune ambiguité, dans la mesure où il a été indiqué que la structure de chaque file est celle d'une suite ordonnée.

- B -

REDUCTION STRUCTURELLE ET LOGICIELLE DE LA M.A.C.

L'exposé qui précède a présenté une machine qui possède l'essentiel des caractéristiques attachées à un ordinateur; si cette machine n'est pas munie de tous les perfectionnements connus, quiconque a l'habitude de programmer se convainc aisément que l'élaboration de logiciels adaptés permettrait de pallier ces insuffisances.
Nous entreprenons maintenant un processus de reduction de cette machine, pour l'amener à une forme la plus compacte possible. Tantôt, nous montrerons que certaines instructions, peuvent s'interpréter comme des programmes décrits au moyen d'autres instructions en nombre et en genre de plus en plus réduit, tantôt, nous interviendrons sur la structure de la machine en la simplifiant, sans qu'elle perde rien de sa puissance. Il arrivera que ces questions soient mêlées.

I SUPPRESSION DU CARACTERE SPECIAL ""

Considérons une M.A.C. dont les lieux peuvent détenir k+1 formes. Nous faisons jouer à la dernière de ces formes le rôle de la marque "".
Le schéma (9) s'obtiendra alors ainsi:
a_{i_o} est un lieu quelconque de la mémoire centrale.

$Ca_{i_o} := \overline{0}$
k fois [$Ca_{i_o} := Ca_{i_o} + 1 \quad (k+1)$; ..... ; ..... ; $Ca_{i_o} := Ca_{i_o} + 1 \quad (k+1)$]
$Cf_n := Ca_{i_o}$

$Cf_n :=$ ""

Voyons maintenant comment obtenir le schéma d'instructions(12).

Ca_{i_o} := $\bar{0}$
Ca_{i_o} := Ca_{i_o} + 1 (k+1)
Ca_{i_o} := Cf_n
Ca_{i_o} := Ca_{i_o} + 1 (k+1)
Si Cu=1 vers e_i

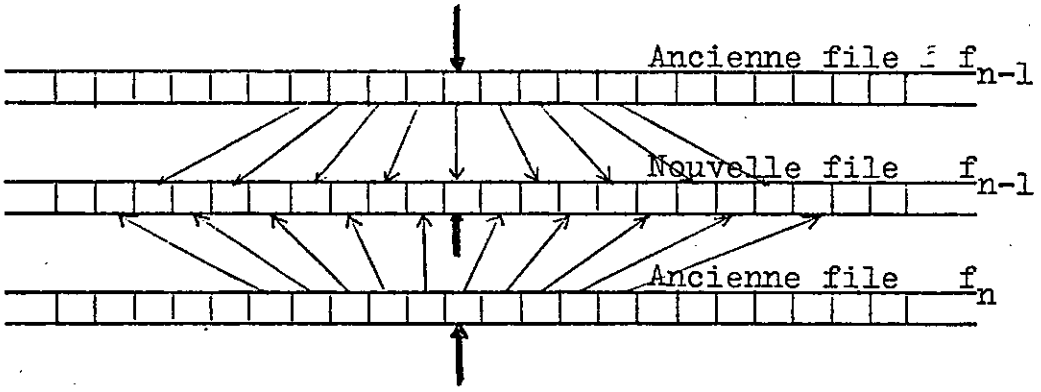
Si Cf_n="" vers e_i

Les deux premières instructions ont pour effet de mettre $\bar{0}$ dans u. Après le transfert depuis la file, u ne sera remis à $\bar{I}$ que dans le seul cas où c'est la forme $\bar{k}$ qui se sera retrouvée dans a_{i_o}.

Ainsi les schémas (9) et (12) ont été supprimés du langage, sans perte de puissance, puisque leurs effets peuvent être obtenus par le moyen des programmes indiqués.
Notons cependant qu'il a fallu accroître le nombre des formes, qui est passé de k à k+1.

II REDUCTION DU NOMBRE DES FILES A UN

Considérons une M.A.C. à N files. Nous allons construire une M.A.C. de même puissance comportant N-1 files.
Le procédé va consister à identifier les N-2 premières files de la M.A.C. à simuler aux N-2 premières files de la M.A.C. réduite. La dernière file de la M.A.C. réduite va servir à simuler les deux dernières files de l'autre machine.
La situation initiale qui affecte ces deux files, sera retranscrite dans la dernière file de la M.A.C. réduite ainsi:



Comme on ne dispose désormais que d'un seul index pour traiter deux files, maintenant copiées sur une seule, il faut trouver le moyen d'indiquer à chaque instant, où l'on pointe, sur la première et la seconde, et laquelle est en cours de pointage.

Comme l'indique le dessin plus haut, la file N-1 de l'ancienne machine, a été recopiée sur la dernière file de la nouvelle machine, selon l'application suivante:

$\cdots\cdots\ v'_{-3}\quad v'_{-2}\quad v'_{-1}\quad v'_{0}\quad v'_{1}\quad v'_{2}\quad v'_{3}\ \cdots\cdots\cdots$

$\cdots\cdots\ v_{-6}\quad v_{-4}\quad v_{-2}\quad v_{0}\quad v_{2}\quad v_{4}\quad v_{6}\ \cdots\cdots\cdots$

De même que pour cette application on associe à v'_n, v_{2n}, pour la file N de l'ancienne machine, on associera à v''_n, v''_{2n+1}.

Dans cette opération préliminaire, les termes indicés par 0, représentent les contenus des lieux pointés par les index correspondants, au début du processus.

D'autre part nous utiliserons une mémoire a_1, qui indiquera par son contenu, 0 ou 1, quelle est la file ancienne, N-1 ou N, dont le pointage est en train d'être simulé sur la nouvelle; nous adjoignons à l'alphabet un signe supplémentaire, /, qui permettra de simuler l'index de la file qui n'est pas en cours de pointage; de ce fait, il sera nécéssaire de conserver en une mémoire a_2, le contenu du lieu de la file dans lequel aura été déposée la marque /; une mémoire a_3, servira de registre intermédiaire, et enfin une mémoire a_4 servira à indiquer par son contenu 0 ou 1, que l'index de la file qui n'est pas en cours de pointage, est situé en avant ou en arrière, de l'index de celle qui est en cours de pointage.

Voyons maintenant comment se présentent les programmes.

En premier lieu, il est indispensable d'initialiser eorrectement le déroulement des opérations:

Au départ, on sait qu'on pointe sur la cas indexée de l'ancienne file N-1.

	Av f_{N-1}	On pointe donc sur la case indexée de la file f_N simulée
	$Ca_2 := Cf_{N-1}$	On met en réserve l'information s'y trouvant
(i)	$Cf_{N-1} := /$	On indique que l'index de la file f_N simulée est là
	Ar f_{N-1}	On se remet en position standard
	$Ca_1 := \bar{0}$	La file simulée en cours de pointage est N-1
	$Ca_4 := \bar{0}$	L'index de la file simulée, non actuellement pointée(N), est situé en avant par rapport à l'index de la file en cours de pointage.

(i): il ne s'agit pas là, en fait d'une instruction disponible, puisqu'elle a été supprimée du langage, mais on a vu que ses effets pouvaient être obtenus au moyen d'un programme convenable qui a été précédemment décrit. Chaque fois qu'une situation similaire se présentera, nous la signalerons par (i), et indiquerons, si ce n'est déjà fait, le programme correspondant.

Maintenant que l'initialisation est faite, il nous faut montrer que toutes les instructions faisant intervenir l'ancienne file N, peuvent être simulées dans leurs effets avec la machine réduite; les schémas concernés sont ceux du type (7), (8), (10), (11).

1) SIMULATION DE $Ca_i := Cf_N$

Si $Ca_1=\bar{0}$ vers e_1 Quelle est la file en cours de pointage?

(C'est bien N) $Ca_i := Cf_{N-1}$

vers la suite.....

(C'est N-1) e_1: $Ca_i := Ca_2$

vers la suite.....

2) SIMULATION DE Av f_N

Si $Ca_1=\bar{0}$ vers e_1 Quelle est la file en cours de pointage?

Si c'est bien N, le branchement vers e_1 ne sera pas pris, et il faudra alors franchir deux cases, en examinant si on ne franchit pas également l'index de l'autre file, car il est essentiel de savoir le positionner par rapport à celui de la file en cours de pointage.

Donc Av f_{N-1}

Si Cf_{N-1}=/ vers e_2

Av f_{N-1} puisque l'index n'a pas été rencontré.

Vers suite.....

e_2: Av f_{N-1}

$Ca_4 := \bar{1}$ puisque l'index a été franchi.

Vers suite.....

S'il s'agit au contraire de N-1, il faut changer de file pour positionner l'index correctement; de ce fait, quittant la file N-1, toutes les informations concernant la case qui était pointée, son contenu, devront être mémo-

risées, et il faudra de plus pouvoir situer la file quittée par rapport à celle qu'il faut atteindre.

e_1:Ca_3 := Cf_{N-1}

On met en réserve l'information sous l'index de la file qu'on va quitter, dans a_3, temporairement, puisque a_2 contient en ce moment l'information correspondante dans la file à atteindre. La restitution se fera ultérieurement.

(i) Cf_{N-1}:= / Quittant la file, il faut la marquer pour pouvoir la retrouver plus tard.

Si Ca_4=$\bar{0}$ vers e_3 Quelle direction prendre?

Arrière e_4:Ar f_{N-1} On se positionne sur la bonne file.

(i) Si Cf_{N-1} ≠ / vers e_4

On cherche la marque /, et on ne sort de la boucle que lorsqu'on l'a trouvée, ce qui se produit à coup sûr.

Cf_{N-1} := Ca_2

On restitue à sa place, l'information qui était contenue dans a_2, puisque l'index ne sera plus situé à cet endroit.

Ca_2 := Ca_3 Le rôle intermédiaire de a_3 est terminé.

Il reste alors à réaliser l'action requise en s'assurant qu'on ne franchit pas l'index de la file quittée.

Av f_{N-1}

(i) Si Cf_{N-1} = / vers e_5

Av f_{N-1}

Ca_4 := $\bar{0}$ L'index de l'autre file est maintenant en avant par rapport à celui de la file effectivement pointée.

Vers suite.....

e_5:Av f_{N-1}

Vers suite.....

Il faut maintenant examiner le cas où la direction prise est en avant.

Avant e_3:Av f_{N-1} On se positionne sur la bonne file.

Si Cf_{N-1} ≠ / vers e_3

On cherche la marque / comme précédemment.

Cf_{N-1} := Ca_2

Ca_2 := Ca_3

Dans ce cas de figure il est exclu que les

positions relatives des deux index soient changées.

Av f_{N-1}

Av f_{N-1}

$Ca_4 := \bar{I}$ L'index de l'autre file est maintenant derrière.

Vers suite.....

3) SIMULATION DE $Cf_N := Ca_i$

Si $Ca_1 = \bar{0}$ vers e_1 Quelle file est pointée?

C'est N' $Cf_{N-1} := Ca_i$

Vers suite.....

C'est N-1 $Ca_2 := Ca_i$

Vers suite.....

Il est clair que la réalisation de l'instruction Ar f_N, s'obtiendrait selon des voies similaires à celles mises en oeuvre pour réaliser Av f_N.

D'autre part on a pu noter que certaines actions décrites dans ce qui précède, étaient référencées en marge par le signe (i); comme on l'a annoncé, il s'agit en fait de programmes qui remplacent des instructions qui ont été supprimées, ou qui peuvent être obtenus simplement.

Par exemple Si $Cf_{N-1} \neq /$ vers e_i peut s'obtenir ainsi:
(s'il y a k+1 symboles, le dernier étant /)

Si $Cf_{N-1} = \bar{0}$ vers e_i

Si $Cf_{N-1} = \bar{1}$ vers e_i

.....

.....

Si $Cf_{N-1} = \bar{k}$ vers e_i

Au total on a donc construit une machine à N-1 files réalisant la totalité des opérations de la M.A.C. à N files. De proche en proche, en un nombre fini d'opérations, on se ramène à une M.A.C. ne possédant plus qu'une seule file. Notons qu'il a fallu étendre l'alphabet, et spécialiser certaines mémoires, donc assurer à la mémoire centrale une taille minimum. Cet aspect des choses est traité plus loin.

III REDUCTION DU NOMBRE DES OPERATIONS MEMOIRE-CENTRALE

1) SUPPRESSION DE L'INSTRUCTION Vers e_i

Le procédé est très simple:
Il suffit d'écrire $Ca_{i_o} := \bar{0}$

Si $Ca_{i_o} = \bar{0}$ vers e_i

Le branchement est alors automatiquement pris.

2) SUPPRESSION DE L'INSTRUCTION Si Cu=1 vers e_i

Rappelons que le contenu de cette case spéciale u, est automatiquement affecté par l'exécution de l'instruction, $Ca_i := Ca_i + 1$ (k). Si le résultat est $\bar{0}$, u est mis à $\bar{1}$, et dans tous les autres cas, u est mis à $\bar{0}$. En quelque sorte, l'évolution de u est cablée.
Nous allons le remplacer en utilisant une case de la M.C., (mémoire centrale) qui servira à le simuler. Notons la a_u.
Les nouveaux programmes seront alors ainsi rédigés:

A l'initialisation, en tête de tout programme il y aura

$Ca_u := \bar{0}$

(i) A chaque intervention d'une instruction de schéma (3) dans les anciens programmes on trouvera désormais la séquence:

$Ca_i := Ca_i + 1$ (k)
Si $Ca_i = \bar{0}$ vers e_1
$Ca_u := \bar{0}$
Vers suite.....
e_1: $Ca_u := \bar{1}$
Vers suite.....

De plus, l'instruction Si Cu=1 vers e_i, sera remplacée par:

Si $Ca_u = \bar{1}$ vers e_i

Notons que là encore, il a été nécéssaire pour arriver au résultat, d'ajouter à la M.C. une case supplémentaire.

(i):voir toutefois une restriction dans leparagraphe suivant.

3) SUPPRESSION DE L'INSTRUCTION $Ca_i := \bar{0}$

```
                  ⌈ Si Ca_i = 0̄ vers e_1
                  ⌊ Ca_i := Ca_i + 1  (k)
                  ⌈ Si Ca_i = 0̄ vers e_1
                  ⌊ Ca_i := Ca_i + 1  (k)
                    Si Ca_i = 0̄ vers e_1
                    .....................
k-1fois             .....................
                    .....................
                    Ca_i := Ca_i + 1  (k)
                  ⌈ Si Ca_i = 0̄ vers e_1
                  ⌊ Ca_i := Ca_i + 1  (k)
                 e_1 : Vers suite.....
```

On a vu dans le paragraphe précédent que l'intervention de l'instruction $Ca_i := Ca_i + 1$ (k), dans les anciens programmes devait s'accompagner de certaines modifications, visant à réaliser la simulation de l'évolution de la case u. Ici la chose n'est, non seulement pas nécéssaire, mais elle doit être évitée; l'application du schéma d'instructions (3) n'a d'autre fonction que celle de faire évoluer les formes dans a_i, jusqu'à être certain d'avoir obtenu $\bar{0}$.
Dans ces conditions l'indicateur de débordement, n'a pas à être affecté; donc les modifications que nous avons signalées plus haut, ne seront pas appliquées dans le présent cas de figure.
Une autre solution aurait pu consister en la mise en réserve du contenu de u, puis en l'application du procédé ici décrit avec les modifications évoquées, puis enfin en la restitution du contenu de u.

4) SUPPRESSION DE L'INSTRUCTION $Ca_i := Ca_j$

Nous utiliserons une mémoire auxiliaire a.

(i) $Ca := \bar{0}$

e_1: $Ca_j := Ca_j + 1$ (k)

$Ca := Ca + 1$ (k)

(i) Si $Ca_j \neq \bar{0}$ vers e_1

A la sortie de cette boucle, il y a donc dans a, le complément à k du contenu initial de a_j, et dans a_j il y a $\bar{0}$.

On réitère alors le procédé, en mettant à $\bar{0}$ au départ, le contenu de a_i, et en l'incrémentant ainsi que a_j, à partir de la valeur initiale de a.
Il y aura alors dans a_i et a_j, le complément du complément de la valeur initiale, c'est à dire la valeur elle-même.

(i) $Ca_i := \bar{0}$

e_2: $Ca := Ca + 1$ (k)

$Ca_i := Ca_i + 1$ (k)

$Ca_j := Ca_j + 1$ (k)

(i) Si $Ca \neq \bar{0}$ vers e_2

vers suite.....

(i): comme précédemment, il s'agit non pas d'instructions mais de programmes, par exemple Si $Ca \neq \bar{0}$ vers e, s'obtiendrait, par une succession de k-1 groupements de la forme: Ca:=Ca+1(k), SiCa=$\bar{0}$ vers e.

En ce qui concerne l'utilisation du schéma $ca := Ca + 1(k)$, nous reformulons les mêmes remarques qu'au paragraphe précédent: les modifications portant habituellement sur u, n'ont pas ici à intervenir.

Nous disposons maintenant d'un langage réduit, dont voici les schémas d'instruction:

La forme F_1

$Ca_i := Ca_i + 1$ (k)
Si $Ca_i = \bar{0}$ vers e_i
$Ca_i := Cf$
$Cf := Ca_i$
Av
Ar

IV REDUCTION DU NOMBRE DES FORMES A DEUX

La MAC de laquelle nous partons peut produire et reconnaitre k formes. Nous allons construire une machine de puissance équivalente, mais bien entendu comme à chaque étape de la réduction, beaucoup moins commode à manipuler, et qui ne nécessitera que l'usage de k-1 formes.
Ainsi de proche en proche, en un temps fini, on pourra se ramener à 2 formes, et il sera très simple de constater qu'une réduction à 1 est impossible.

Faisons d'abord une remarque sur la notion d'équivalence de puissance: il est clair que la nouvelle machine ne fera pas exactement ce que faisait l'ancienne, puisque précisément cette dernière est dans sa structure même distincte de l'autre. Ce qu'on entend par puissance équivalente demande donc à être précisé; à cette fin des considérations sémantiques nous paraissent tout à fait bienvenues:
Lorsqu'on opère un calcul, dans un cadre algorithmique, on part d'un ensemble de formes sur lesquelles un traitement particulier est appliqué, et on obtient en retour un nouvel ensemble de formes. Dans tous ces processus, un fait essentiel tient à ce que ces formes et les actes qui s'exercent sur elles, ne sont pas regardés pour ce qu'ils sont, mais pour ce qu'ils représentent. Certes aucune forme n'existe sans se manifester physiquement selon un certain mode; de même toute action emprunte les voies matérielles qui lui

sont propres. Mais ici ce n'est pas du tout ce qui nous occupe. Peu importe que les données que nous fournissons à la machine soient représentées physiquement de telle ou telle manière, peu importe que les mécanismes représentatifs du traitement relèvent de telle ou telle technologie. Ce qui compte, c'est que les interprétations attachées à ces formes et à ces actions puissent être unifiées. C'est en ce sens que, par exemple, on peut dire que sont équivalentes en puissance, une calculette électronique, et une machine mécanique du type Pascaline; à l'une et à l'autre, on peut donner des représentations physiquement distinctes, des mêmes nombres. L'une et l'autre peuvent opérer sur ces représentations des actions physiquement très différentes, et pourtant associées aux mêmes processus abstraits, addition, multiplication, etc..... Les mêmes reflexions valent pour les formes résultantes.

Ces considérations nous semblent de nature à éclairer et à justifier une définition plus formelle de l'équivalence de puissance entre machines différentes. On ne saurait viser directement à s'assurer que les sémantiques de deux programmes quelconques sont en fait identiques. Même dans la pratique, il est bien rare qu'à la lecture d'un programme; ce qu'il "fait" apparaisse clairement. De toute façon il faut ici poser le problème dans toute sa généralité. Plutôt que de chercher l'interprétation commune à deux programmes, il vaut mieux considérer qu'ils appartiennent à une même classe d'équivalence, et dégager un critère formel qui les réunit ou les sépare:

Soit L_1 un langagepermettant de décrire un ensemble de machines. Soit M_1 une telle machine, c'est à dire un programme. Soit C_1 l'ensemble des configurations qu'on peut soumettre à M_1, et $M_1(C_1)$ l'ensemble des configurations résultantes.

Soit alors dans un langage L_2 une machine M_2, et son ensemble de configurations C_2,

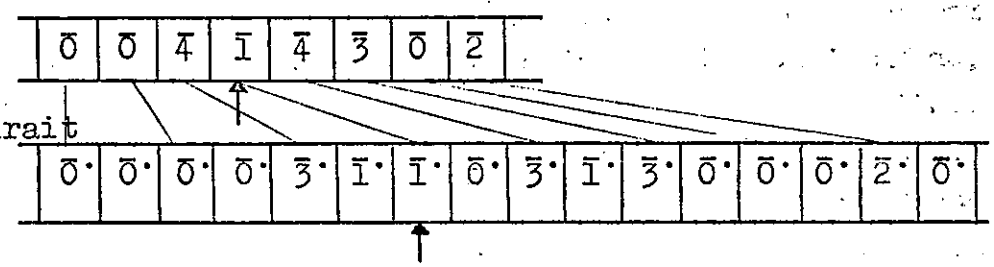
on dira que M_2 est autant puissante que M_1, s'il existe deux injections f et g, l'une de C_1 dans C_2, l'autre de $M_1(C_1)$ dans $M_2(C_2)$, qui vérifient:

Pour tout c appartenant à C_1 $\quad g(M_1(c)) = M_2(f(c))$

Deux machines seront dites aussi puissantes l'une que l'autre, si chacune est autant puissante que l'autre, et deux langages seront dits aussi puissants l'un que l'autre, si à chaque machine de l'un quelconque, on peut associer une machine aussi puissante de l'autre.

Dans le cas qui nous occupe, l'ensemble des formes de la
(i) première machine sera noté $\overline{0}, \overline{1}, \overline{2}, \ldots\ldots\ldots, \overline{k-1}$.
Celui de la seconde machine $\overline{0}; \overline{1}; \overline{2}; \quad , \overline{k-2}$:
L'injection de l'ensemble des configurations C_1 dans C_2 s'obtiendra ainsi:
Pour ce qui concerne la file externe:
L'information contenue dans l'ancienne file est déposée dans la nouvelle selon le procédé $n \longrightarrow 2n$, c'est à dire à raison d'une case sur 2, en changeant $\overline{i}$ par $\overline{i}^{\cdot}$, quand $\overline{i}$ varie de $\overline{0}$ à $\overline{k-3}$. Les deux dernières formes $\overline{k-2}$ et $\overline{k-1}$, sont changées en $\overline{k-2}^{\cdot}$. Dans les cases impaires de la nouvelle file, on inscrit partout $\overline{0}^{\cdot}$, sauf dans celles qui suivent une case contenant $\overline{k-2}^{\cdot}$, et qui correspondent à des cases de l'ancienne file qui contenaient $\overline{k-1}$. Pour celles-ci, on écrit $\overline{1}^{\cdot}$.

A titre d'exemple la configuration suivante (k=5)



Pour ce qui concerne la M.C., on procède de la même façon en doublant le nombre des cases, et en recopiant l'information initiale selon la méthode qui vient d'être exposée. On constate sans mal, que l'application réalisée est une injection.
Nous montrons maintenant comment réaliser dans la nouvelle machine, les instructions de l'ancienne.

(i): les formes sur la file et celles en M.C. sont dénommées de la même manière. Il n'y a aucune perte de généralité.

Rappelons qu'il n'y a plus que 6 schémas d'instructions:

$Ca_i := Ca_i + 1$ (k)	(1')
Si $Ca_i = \overline{0}$ vers e_i	(2')
$Ca_i := Cf$	(3')
$Cf := Ca_i$	(4')
Av	(5')
Ar	(6')

Le schéma (2') n'a pas dans le cas présent à être modifié.

1) REALISATION DU SCHEMA (1') $Ca_i := Ca_i + 1$ (k)

Dans le doublement des cases qu'on a opéré, nous conserverons dans la nouvelle machine, les mêmes noms, pour les cases images des anciennes a_i. Les cases associées à ces a_i seront nommées a'_i.

$Ca_i := Ca_i + 1$ (k-1)
Si $Ca_i = \overline{0}^{\cdot}$ vers e_1
(i) Vers suite.....
e_1: Si $Ca'_i = \overline{0}^{\cdot}$ vers e_2
(i) Vers suite.....
(i) e_2: $Ca_i := \overline{k-2}^{\cdot}$
(i) $Ca'_i := \overline{1}^{\cdot}$
(i) Vers suite.....

(i): Comme déjà signalé, il s'agit de programmes. On sait comment les construire.

2) REALISATION DU SCHEMA (3') $Ca_i := Cf$

(i) Si $Cf \neq \overline{k-2}^{\cdot}$ vers e_1
Av
(i) Si $Cf = \overline{0}^{\cdot}$ Vers e_2
Ar
(i) $Ca_i := \overline{k-2}^{\cdot}$
(i) $Ca'_i := \overline{1}^{\cdot}$
(i) Vers suite.....
e_2: Ar
(i) $Ca_i := \overline{k-2}^{\cdot}$
(i) $Ca'_i := \overline{0}^{\cdot}$

(i) Vers suite.....

e_1: Ca_i := Cf

(i) Ca_i' := $\bar{0}$

(i) Vers suite.....

(i):Programmes.

3) REALISATION DU SCHEMA (4') Cf := Ca_i

On voit sans grande difficulté que la réalisation de ce schéma s'obtient selon la même technique que celle qui vient d'être exposée avec des adaptations mineures.

4) REALISATION DES SCHEMAS (5') et (6') Av Ar

La transcription sera très simple:

Av ⟶ Av Av
Ar ⟶ Ar Ar

Ainsi de proche en proche, on peut réduire le nombre des formes, jusqu'à ce qu'il n'en subsiste plus que deux.

Il est impossible alors de réduire ce nombre, car le concept même d'information est fondé sur la notion de perturbation. Or cette notion, ne peut être étayée qu'à la condition qu' existent au moins deux formes, la perturbation étant alors la différence que présentent les deux formes, en les opposant.

Un autre moyen de se convaincre de la chose consiste à supposer qu'on ne dispose en fait que d'une seule forme, disons 0 par exemple. Alors partout dans la file, figure ce signe, puisqu'il n'en existe pas d'autre, de même dans la M.C.. Les schémas d'instruction ne permettent alors plus d'opérer aucune véritable action; en effet qu'il s'agisse d'un transfert depuis la file vers la mémoire, ou en sens inverse, d'un mouvement de mémoire à mémoire, d'une application du schéma (1'), la machine reste exactement dans le même état que celui qu'elle avait, avant que de telles actions ne

soient exercées. Il en est de même pour les schémas (5') et (6'); certes on peut avancer ou reculer l'index sur la file, mais il n'est pas raisonnable de prétendre avoir exercé réellement sur la machine une action, puisqu'après coup il sera impossible de déceler d'une quelconque façon que ce mouvement a eu lieu.
Au total aucune action véritable ne peut plus s'exécuter et la puissance de ces machines s'en trouve purement et simplement annulée.

V REDUCTION DU NOMBRE DES CASES DE LA M.C. A UN.

Jusqu'ici il a fallu, pour réaliser notre processus de réduction, accroître la taille de la M.C., dans de très nombreuses circonstances.
Nous allons montrer comment, partant d'une machine à K+1 cases de M.C., on peut lui substituer une machine autant puissante à K cases.

La démonstration est articulée autour des idées suivantes:

- Une partie de la file va servir à remplacer la case manquante, et ce de manière à indiquer également quel est son contenu.
- Comme il faudra alors aller consulter cette partie à l'occasion de certaines opérations faisant intervenir la case manquante, on sera amené à quitter la case en cours de pointage sur la file, et donc il sera nécessaire de fournir un moyen qui indique comment retrouver ce lieu, et un moyen permettant de savoir dans quelle direction se déplacer pour trouver sur la file la"case" a_{K+1}.
- Il est nécessaire d'organiser un codage élaboré sur la nouvelle file des informations qui existaient sur l'ancienne, car il est désormais interdit d'utiliser certains schémas de programme(ceux qui étaient signalés par (i)) lorsque ceux-ci imposaient de façon implicite le recours à des cases supplémentaires.

Par exemple, il ne peut plus être question de faire appel à des schémas du genre: Si Cf=0 (ou 1) vers e_i. Pour opérer toutes sortes de tests sur les configurations de la file, il sera inévitable, au préalable, de transférer en M.C. certains contenus de la file; or de telles opérations ont forcément pour effet de détruire ce qui se trouvait alors dans les cases atteintes de la M.C.. Il faut donc prévoir, pour chaque case de la file susceptible d'être testée, une case adjointe qui servira à recueillir le contenu de la case M.C. utilisée pour le test. Pour simplifier les choses, cette duplication des cases, s'obtiendra par le moyen d'une structuration spéciale de la file.

En procédant ainsi on aura l'avantage de ne pas multiplier le nombre des codes: en effet, le même, selon la place occupée dans la file pourra avoir plusieurs sens.

La structuration sera la suivante:

- La suite des informations de l'ancienne file est appliquée sur la nouvelle, selon l'injection $n \longrightarrow 4n$.
- L'index de la nouvelle file pointe sous la case image de celle qui était pointée dans l'ancienne.
- Deux cases à gauche de la précédente, il y a le code de a_{K+1}, qui est 0 ou 1 selon le cas.
- En ce qui concerne la mémoire centrale, on n'introduit aucune modification, dans la distribution initiale et la structure des données.

1) <u>REALISATION DU SCHEMA (1')</u>

Il s'agit en fait de la seule instruction $Ca_{K+1} := Ca_{K+1} + 1$, puisque l'application de ce schéma aux autres mémoires n'a pas à être modifié.
De plus nous noterons désormais ce type d'instruction $\bar{a}_i$.
En effet, les deux seules formes sont 0 et 1; le schéma (1') échange le 1 en 0 et le 0 en 1.

a_{i_o} est une case donnée de la M.C.

Ar	on pointe sur une case "vide"
Cf := Ca_{i_o}	On préserve le contenu de a_{i_o}
Ar	on pointe sur a_{K+1}
Ca_{i_o} := Cf $\bar{a}_{i_o}$ Cf := Ca_{i_o}	Le changement est opéré
Av	
Ca_{i_o} := Cf	on restitue le contenu initial de a_{i_o}
Av	

2) REALISATION DU SCHEMA(2') Si $Ca_i = \bar{0}$ vers e_i

Là encore, la question ne se pose que si la case concernée est a_{K+1}. Dans les autres cas, aucune modification n'est à introduire.

	Ar
	Cf := Ca_{i_o}
	Ar
	Ca_{i_o} := Cf
	Si $Ca_{i_o} = \bar{0}$ vers e'_i
e'_i:Av	Av
Ca_{i_o}:=Cf	Ca_{i_o} := Cf
Av	Av
e_i:.....	.....
suite avec branchement	..suite du programme(sans branchement)
	.....

3) REALISATION DU SCHEMA (3') Ca_i := Cf (et du schéma (4')

Là encore, aucune modification si la case n'est pas a_{K+1}.

```
Ar                pointe case "vide"
Cf := Ca_io       mise en réserve du contenu de a_io
Av                remise sur Cf
Ca_io := Cf  ┐
Ar           │
Ar           │    réalisation du transfert
Cf := Ca_io  ┘
Av           ┐    récupération du contenu
Ca_io := Cf  ┘    initial de a_io
Av                remise en position normalisée
```

La réalisation du schéma (4') se fait selon des voies quasiment identiques.

4) REALISATION DES SCHEMAS (5') et (6') Av Ar

La structure des programmes précédents indique, qu'il est nécessaire, avant que ne se déroule une instruction, d'avoir disposé, deux cases à gauche de la case en cours de pointage, le code correspondant à la valeur de a_{K+1}.
De ce fait, à l'occasion d'un déplacement du type Av ou Ar, il faudra transporter la valeur de a_{K+1}.

Réalisons Av :

```
Av
Cf := Ca_io       sauve a_io
Ar           ┐
Ar           │    cherche a_K+1
Ar           ┘
Ca_io := Cf  ┐
Av           │
Av           │    réalise le déplacement
Av           │         de a_K+1
Av           │
Cf := Ca_io  ┘
Ar
Ca_io := Cf       restitution de a_io initial
Av           ┐
Av           │    remise en position
Av           ┘         normalisée.
```

La réalisation du schéma (6') s'obtiendrait selon des voies quasiment identiques.

De proche en proche, on diminue ainsi le nombre des cases de la M.C.; il est cependant essentiel, de pouvoir disposer, à chaque cran du processus de réduction, d'une mémoire interne (celle qui a été appelée a_{i_0} dans les démonstrations précédentes).

Au total, le langage sous sa forme réduite apparait maintenant ainsi:

$\bar{a}$ application de la permutation $(\bar{0}\ \bar{1}) \longrightarrow (\bar{1}\ \bar{0})$ au contenu de a, unique case de la M.C.

Si Ca=$\bar{0}$ vers e_i

Ca := Cf

Cf := Ca

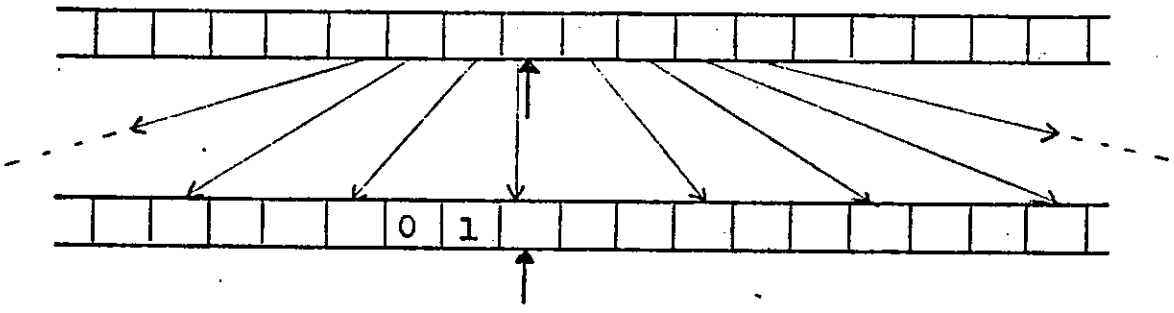
Av

Ar

La forme F_2

VI SUPPRESSION DE L'OPERATION $\bar{a}$

Avant d'examiner certains faits de limitation, qui nous assureront que le procédé de réduction est achevé, nous allons construire une machine aussi puissante que la précédente, qui ne fait pas appel à l'opération $\bar{a}$.
La structuration de la file se présente ainsi que le suggère le dessin suivant, sans que nous en donnions une définition formelle, ceci ayant déjà été vu à plusieurs reprises.



1) REALISATION DE $\bar{a}$

	Si Ca=$\bar{0}$ vers e_1
	Ar
	Ar
e_1:Ar	Ca := Cf
Ca := Cf	Av
Av	Av
.....	.....
suite(avec brancht.)	suite(sans branchement)
.....	.....

2) REALISATION DE Si Ca=$\bar{0}$ vers e_i ; Ca:=Cf ; Cf := Ca

Ces instructions n'ont pas à être modifiées.

3) REALISATION DE Av et Ar

Comme on vient de le voir en 1), il faut disposer des standard $\bar{0}$ et $\bar{1}$, à la gauche de la case en cours de pointage. La méthode consistera donc à transporter ces formes standard à chaque mouvement de l'index, pour les mettre en position normalisée.

Av

```
Av
Av
Av                    ——met en réserve le contenu
Av                                de a
Cf := Ca
Ar
Ar
Ar
Ar
Ar
Ar                    ——pointe alors sur 0̄
Ca := Cf
Av
Av                    ——place le 0̄
Av
Cf := Ca
Ar
Ar                    ——pointe alors sur 1̄
Ca := Cf
Av
Av                    ——place le 1̄
Av
Cf := Ca              ——récupère le contenu ini-
Av                                tial de a
Av
Ca := Cf
Ar                    ——retour à position norma-
                                  lisée
```

La réalisation de l'instruction Ar s'obtiendrait selon des voies identiques.

La forme réduite du langage est maintenant:

Si $\bar{0}$ vers e_i	(1") Test sur la seule case de la M.C.
af	(2") Transfert de la file vers la M.C.
fa	(3") Transfert de la M.C. vers la file
Av	(4") Avant sur la file
Ar	(5") Arrière sur la file

Il n'y a plus que deux formes $\bar{0}$ et $\bar{1}$.

Forme F_3

Cette forme impose que $\bar{0}$ et $\bar{1}$ soient déposés, au préalable sur la file. Au départ la case où pointe l'index est précédée du $\bar{1}$ et du $\bar{0}$.

Dans le chapitre suivant, nous attirons l'attention, sur un ensemble de faits de limitation. La forme réduite à laquelle nous avons abouti, ne semble pas pouvoir être rendue, encore plus compacte. Néammoins, il n'est pas certain, que cette intuition, n'ait pu se développer quelques étapes avant la dernière que nous venons d'atteindre.
Dans ces conditions il nous faut établir dans quel sens, la forme réduite dégagée a bien un caractère minimal.

- C -

FAITS DE LIMITATION

I LA QUESTION DU BORNAGE DU NOMBRE DES ETIQUETTES

En passant succéssivement des formes F_o à F_1, puis F_2 et F_3, on a établi que tous les processus de calcul, descriptibles dans le type F_o, pouvaient être représentés dans les types suivants. Bien entendu, la régréssion en sens inverse est également possible, chaque machine de type F_{i+1} étant obtenue à partir du type F_i, par une technique"d'affaiblissement".

Il est très important d'insister sur cette notion de représentation; les machines F_{i+1} ne font pas ce que font les machines F_i, mais leurs actions peuvent représenter les processus du type antérieur; pour passer d'un type au suivant, on a chaque fois décrit, comment était mise en oeuvre cette représentation, par le moyen d'un codage injectif, non ambigu.

Observons également un fait dont l'évidence demandera à être rappelée plus loin: lorsque ces techniques de codage sont appliquées, la question de la construction des objets sur lesquels elles s'appliquent, ne se pose pas; en effet les configurations initiales de la file du type i sont données, et grâce au codage, on définit automatiquement les configurations correspondantes pour le type i+1.
On verra plus loin l'importance fondamentale de ce fait.

Pour l'instant, nous abordons une question dont les liens se révèleront ultérieurement avec ce qui précède: le langage residuel est muni d'un schéma d'instructions, Si $Ca=\bar{0}$ vers e_i. Ce schéma permet d'engendrer un nombre non borné d'instructions, puisqu'un choix illimité d'étiquettes e_i est offert. Que se passe-t-il, si ce choix est restreint, et qu'on impose un nombre maximum fixé d'étiquettes?

1) UN PROBLEME PARTICULIER

Soit N le nombre maximum d'étiquettes distinctes disponibles. On demande alors de construire la machine suivante:
Par souci de simplicité, nous utiliserons la forme F_2 du langage, plus commode.

n est un nombre tel que 2^n-1 est inférieur ou égal à N.
Au départ, la machine a son index sous la première case à gauche de la configuration qui lui est soumise; une configuration est une suite de n digits.
On veut que la machine explore la configuration, en la mettant à $\bar{0}$, au fur et à mesure qu'elle la parcourt, puis qu'elle écrive alors sur la file, une certaine suite de 2^n digits:

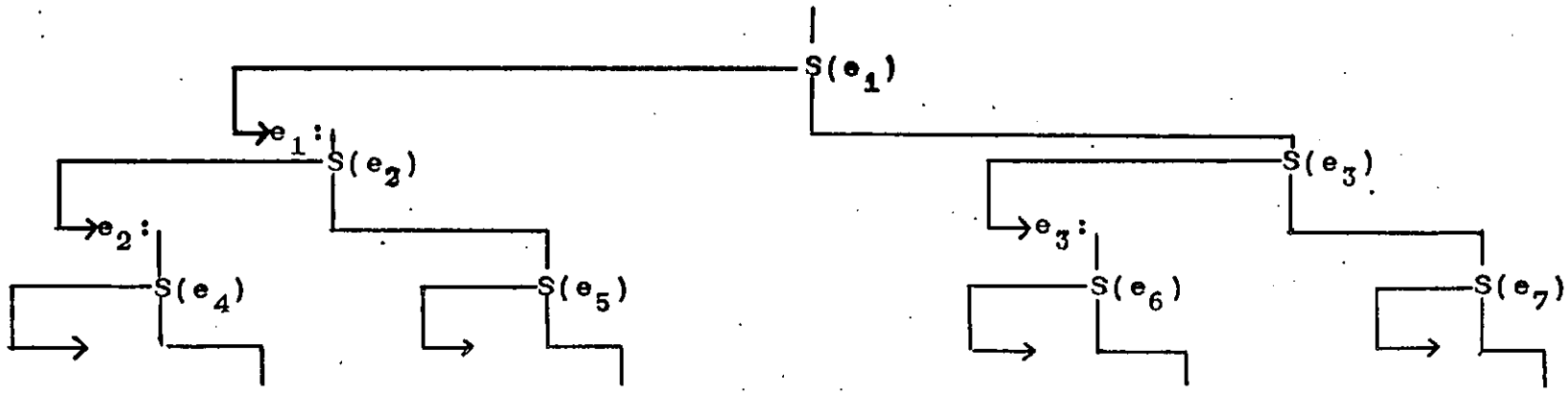
$\bar{0}\bar{0}\ldots\ldots\ldots\ldots\bar{0}$	si on avait lu	$\bar{0}\bar{0}\ldots\ldots\bar{0}$
$\bar{0}\bar{0}\ldots\ldots\ldots\ldots\bar{0}\bar{1}$	" " "	$\bar{0}\bar{0}\ldots\ldots\bar{0}\bar{1}$
$\bar{0}\bar{0}\ldots\ldots\ldots\ldots\bar{0}\bar{1}\bar{1}$	" " "	$\bar{0}\bar{0}\ldots\ldots\bar{1}\bar{0}$
$\bar{0}\bar{0}\ldots\ldots\ldots\ldots\bar{0}\bar{1}\bar{1}\bar{1}$	" " "	$\bar{0}\bar{0}\ldots\ldots\bar{0}\bar{1}\bar{1}$
.....		
.....		
$\bar{1}\bar{1}\bar{1}\ldots\ldots\ldots\ldots\bar{1}\bar{1}\bar{1}$	si on avait lu	$\bar{1}\bar{1}\ldots\ldots\bar{1}\bar{1}$

Il n'est pas douteux qu'on demande ici la réalisation d'un calcul: une production déterminée de signes dans un certain ordre. Que ce calcul admette une interprétation en termes de conversion binaire-unaire, n'a pas ici à être pris en considération.

Appelons alors $S(e_i)$ la sequence suivante:

Av
af
Si Ca = $\bar{0}$ vers e_i
$\bar{a}$
fa

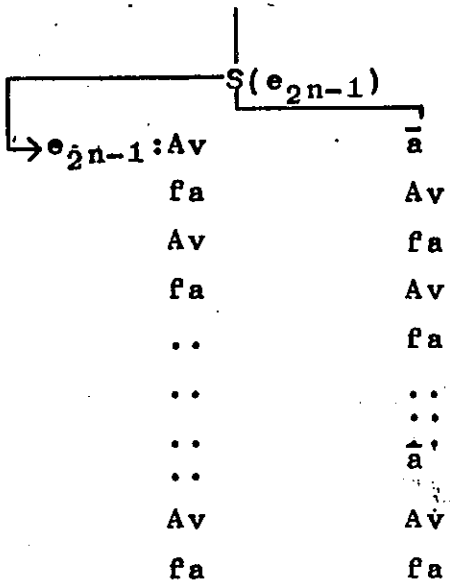
Un programme qui répond alors à la question est donné à la page suivante.
Anticipant un peu les choses, on serait tenté de dire, qu'à peu de choses près, il s'agit non pas d'un programme mais <u>du</u> programme convenable, en ce sens qu'il est unique(à quelques détails près).

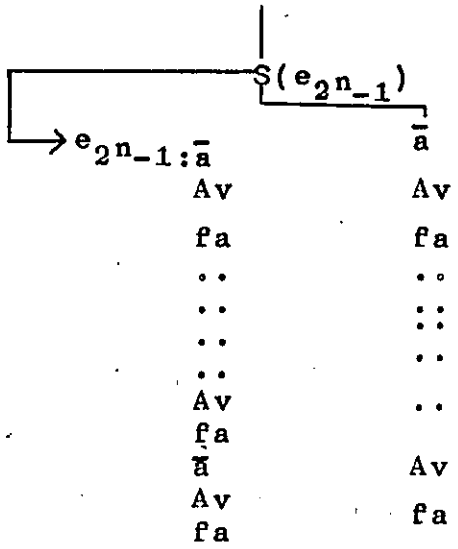


Examen case 1

Examen case 2

Examen case 3





La technique utilisée a consisté à aiguiller le processus dans 2^n chemins de programme distincts, chacun se terminant par la séquence des actes à réaliser, correspondant à la configuration reconnue.
On constate bien que le schéma de la page précédente comporte bien $(2^n-1-2^{n-1}+1) \times 2$ terminaisons, c'est à dire 2^n.
On note également qu'au total, il a fallu disposer de 2^n-1 étiquettes.

Supposons maintenant que le même problème ait été posé avec un nombre de digits dans la configuration initiale, n tel que 2^n-1 soit supérieur à N.
Il est clair que l'énoncé du problème oblige à parcourir la totalité de la configuration, pour pouvoir exercer les actions appropriées, qui sont différentes pour chaque configuration. En outre, comme on impose une destruction de cette configuration, il n'est plus question d'aller la consulter ultérieurement; il n'est pas possible non plus, de la mettre en réserve en M.C., puisque celle-ci est réduite à une case unique, qui d'ailleurs est obligatoirement confinée dans un rôle d'examen des éléments de la file.
Lorsque l'opération de remise à $\bar{0}$ des éléments de la configuration est achevée, il n'y a donc plus aucune information sur la file, et la M.C., à cause de sa structure, ne permet d'opérer au plus que deux branchements de programme: or, à ce moment là, il faut pouvoir exercer 2^n séquences d'actions distinctes, et il est donc nécessaire qu'existent 2^n chemins de programme distincts.
Ceci ne peut être réalisé qu'à la condition de disposer de 2^n-1 étiquettes. On notera au passage que la moitié de tous les chemins est obtenue par la commutation implicite du programme. Si par ailleurs le nombre des étiquettes est borné, le problème n'admet donc plus de solution.

Nous énoncerons donc: il existe des classes de processus de calcul, dont les descriptions dans les langages F_0, F_1, F_2, F_3, sont associées à des nombres d'étiquettes, l'ensemble de ces nombres étant illimité.

2) REMARQUE

Répondons par avance à un pseudo-paradoxe, qu'il serait possible de tirer des conclusions précédentes: on sait que dans (i)le langage F_3, on est en mesure de donner la description d'une machine universelle; tout programme de F_3 et toute configuration associée, peuvent être codés . Le programme universel appliqué à ce code, donne alors comme résultat un code susceptible de représenter le résultat du calcul opéré par l'algorithme qu'on a codé, sur la configuration associée, elle-même codée. Or, ce programme universel, ne demande pour être construit, qu'un nombre fini d'étiquettes.
Dans ces conditions on pourrait craindre qu'il n'y ait une contradiction avec le résultat établi dans le paragraphe précédent, où il est affirmé que la puissance du langage est effectivement réduite si le nombre des étiquettes est borné.

La réponse est la suivante:
On a pris soin d'indiquer au début de ce chapitre, qu'a priori, la question de la construction des objets-configurations soumis à un algorithme ne se posait pas: l'objet est donné, et s'il faut le représenter différemment, on sait le coder. Mais ici il n'en est pas du tout de même; le procédé de codage de la machine universelle, peut certes être parfaitement défini, mais encore faut-il lui présenter les objets dont on construira le code. Or ces objets ne sont maintenant plus donnés; il faut les construire, c'est à dire élaborer des programmes, et on vient de voir que certaines classes de problèmes imposaient le recours à un nombre non borné d'étiquettes.
Il n'est donc pas judicieux de prétendre limiter le langage par un bornage du nombre des étiquettes, sous le motif que la machine universelle n'en utilise qu'un nombre fini, puisque , par ailleurs, la description des objets soumis au codage, ne peut être correctement mise en oeuvre, qu'à la condition de disposer d'un nombre illimité d'étiquettes.
Du point de vue de la didactique, il nous semble que la nature de cette confusion, tient au terme "universel", qui par certains aspects est assez maladroit; le caractère universel de la machine du même nom n'est pas determiné par l'existence d'un programme qui aurait la vertu extraordinaire de repré-

senter, en quelque sorte, la somme de tous les algorithmes possibles et imaginables. Ce qui, en fait, est commun à tous les programmes, c'est la façon dont on les exécute.
En d'autres termes, la procédure d'exécution des algorithmes, est elle-même un algorithme; l'existence du programme universel traduit alors le fait qu'aussi complexes puissent être les algorithmes, notamment par le nombre de leurs étiquettes, la procédure d'exécution de ceux-ci, conserve le même niveau de complexité, indépendant des algorithmes qui lui sont soumis.
Comme conséquence, nous noterons qu'il faut donc éviter de considérer comme maximum, le niveau de complexité de l'algorithme universel.

(i):ce qui est dit du langage F_3, peut être dit au même titre des autres langages qui ont été présentés.

II CARACTERE MINIMAL, STRUCTUREL ET LOGICIEL, DE LA FORME F_3

1) PRECISIONS

Nous nous assurons maintenant que la forme F_3, ne peut plus être réduite d'aucune manière.
On vient de voir qu'on ne peut espérer limiter le nombre des étiquettes; on a montré également la nécessité de disposer de deux symboles, et celle de conserver la case unique de la M.C.
Il est donc clair, que du point de vue de l'architecture de la machine, le matériel minimal a été atteint. Du point de vue externe, c'est à dire relativement à la file, nous négligerons le fait qu'il serait possible de la remplacer, par une file illimitée dans un seul sens.
D'ailleurs il serait tout à fait possible de ne pas considérer qu'il s'agit là véritablement d'une opération de réduction puisque la sémantique de l'instruction Ar (ou Av selon la limitation introduite) s'en trouverait modifiée: en effet il faudrait introduire un concept de butée, jusque là étranger au langage.
Ceci nous amène à préciser dans quel sens nous utilisons le terme minimal.
En premier lieu on notera que c'est le terme "minimal" et non "minimum", qu'on a retenu. Ceci pour indiquer que le proces-

sus de réduction qu'on a suivi, n'est pas forcément unique. Il est, par exemple, possible d'imaginer qu'une forme réduite ultime ait été constituée, par une machine d'une case unique de M.C., à deux files, l'une sur laquelle ne pourraient s'exercer que des transferts du type af, l'autre sur laquelle on se limiterait aux transferts fa. Toutes sortes de combinaisons peuvent être imaginées.

Il est clair qu'une machine du genre décrit plus haut, n'est pas, directement, comparable aux machines du type F_3, en ce sens qu'elle n'en constitue, ni un prolongement ni une restriction.

Par contre on observera que les différents passages que nous avons suivis, des formes F_i aux formes F_{i+1}, sont caractéristiques en ce sens qu'à chaque pas de la réduction, on obtient une machine qui, du point de vue structurel, est une sous-machine de la précédente(par exemple quand on diminue le nombre des cases de la M.C.), et du point de vue logiciel, possède un ensemble d'instructions qui est un sous- ensemble du précédent. Ainsi le cheminement suivi nous permet, non seulement d'affirmer que la forme réduite est équivalente en puissance, dans le sens qu'on a défini, aux autres formes, mais encore que toutes les formes précédant d'autres, en constituent des extensions.

2) CARACTERE MINIMAL

Nous montrons maintenant que la forme F_3 forme un ensemble minimal. Pour cela il suffit de nous limiter à l'aspect logiciel. On a vu en effet que sur le plan structurel, la réduction ne pouvait plus être poursuivie. La technique va consister à supprimer dans l'ensemble des instructions un schéma quelconque, puis à montrer qu'il existe dans l'ancienne forme des processus de calcul descriptibles, qu'il n'est plus possible de représenter dans la nouvelle.

— Suppression du schéma:Si $\bar{0}$ vers e_i

Il est évident qu'on ne saurait se passer de ce schéma, puisqu'alors tout programme est formé d'un seul chemin . Toute procédure est alors forcément "aveugle", puisque, si subsiste la possibilité de saisir les données et de les transférer sur la file, on ne dispose plus d'aucun moyen, permettant de savoir quelle est la donnée qu'on vient par exemple de saisir.

Or il est clair que certaines procédures de calcul, réalisables dans la forme F_3, nécéssitent en elles-mêmes de prendre connaissance des configurations, pour pouvoir exercer des actions appropriées et sélectives.

— Suppression de af ou fa

Ici la situation confine à l'incapacité totale: soit, en supprimant af, on empêche la saisie des données, et on voit mal comment pourrait s'opérer un calcul dont les données resteraient inconnues, soit, en supprimant fa, on empêche toute modification de la configuration initiale.

— Suppression de Av ou Ar

La nouvelle machine, non dotée du mouvement Av, par exemple, ne peut évidemment pas accomplir les mêmes actes que la précédente, mais l'impossibilité qu'il faut traduire est d'une autre nature: ce qu'il faut prouver, c'est qu'il existe des algorithmes descriptibles dans l'ancien langage, qui ne peuvent en aucun cas être représentés dans la nouvelle.
Exhibons un type de problème répondant à la question:

Une configuration est une suite d'au moins n+2 digits, et d'au plus 2n digits, ainsi formée: n est le nombre des digits $\bar{1}$, consécutifs, en tête de la séquence; la suite de la séquence peut alors être formée de toutes les suites : $\bar{0}\bar{1}$ ou $\bar{0}\bar{0}\bar{1}$, ..., ou $\underbrace{\bar{0}\bar{0}.....\bar{0}\bar{0}}_{n}\bar{1}$, ou $\underbrace{\bar{0}\bar{0}.....\bar{0}\bar{0}}_{n}\bar{0}$. ($n \geqslant 2$)

Le nombre n, n'est pas connu à l'avance. On demande alors de construire une machine qui détecte si, oui ou non, la séquence comporte autant de $\bar{0}$ que de $\bar{1}$.
L'ensemble des configurations qui peuvent se présenter est formé par la liste:

1101
(i) 1100
11101
111001
(i) 111000
.....
.....
11.....101
11.....1001
.....
.....
11.....100....01
(i) 11.....100.....0
.....
.....

Il s'agit de différencier les séquences marquées d'un (i), des autres.
Si, seule l'action Av est disponible, aucune prise de décision ne pourra avoir lieu tant que l'action Av n'aura pas permis d'atteindre un $\bar{0}$; en effet, ce n'est qu'à ce moment que la longueur de la demi-séquence est connue. Cette information est évidemment indispensable, pour espérer résoudre le problème; il faut donc l'acquérir, le seul moyen étant de faire Av, et puisqu'il est alors exclu de retourner la consulter, il faut également la mémoriser. Puisque tous les moyens de stockage nous sont otés, il ne reste que la commutation dans le programme, pour définir des chemins distincts correspondants aux actions distinctes qu'il faut pouvoir exercer. Or, si le nombre des étiquettes n'est pas borné, il est cependant fini dans tout programme; de ce fait, la suite initiale des $\bar{1}$, étant de longueur imprévisible, il devient impssible de définir une commutation <u>finie</u> qui puisse la mémoriser.

Par contre le problème posé, reçoit une solution dans le langage réduit. A titre de curiosité, nous donnons ci-après une solution dans la forme F_2:
Au départ l'index pointe sur le premier digit de la séquence.

```
   af
   Ar      | marque le début de la
   a       | séquence par un 0
   fa      |

e1:Av
   af      |
   a       | franchit la suite
   Si 0,e1 |    des 1

   Av
   af      |
   a       | élimine les séquences
   Si 0,non|    11...101

   Ar
e2:Ar      |
   af      | retour à la marque.
   a       | On est sur d'avoir:
   Si 0,e2 | 011...100XXX...
```

```
e3:Av
   af      | franchit les 1
   a       |
   Si 0,e3 |

   ā
   fa      |
   Ar      | amorce le pro-
   a       | cessus d'inver-
   fa      | sion des 0 et 1

e4:Ar
   af      |
   a       | retour à la
   Si 0,e4 |   marque
```

A la fin de cette phase, on est sur d'avoir à traiter une séquence de la forme: 011...1010XXX...

Dès lors, la méthode consiste à épuiser la demi-séquence de début par celle de droite: si on peut épuiser la suite initiale des 1, par la suite des 0, avant d'avoir rencontré un 1 en fin de séquence, la réponse est oui; la marque joue le rôle de butée.

e_5:Av
af
a
Si 0,e_5

e_6:Av
af
Si 0,e_6

e_7:Av
af
a
Si 0,e_7

Av
af
Si 0,e_8
Ar

e_9:Ar
af
a
Si 0,e_9

e_{10}:Ar
af
Si 0,e_{10}
Ar
af
Si 0, oui
non

e_8:Ar
af
a
fa

e_{11}:Ar
af
a
Si 0,e_{11}

e_{12}:Ar
af
Si 0,e_{12}

af
a
fa
Ar
af
Si 0,oui
a
Si 0,e_4 (réitération du processus)

En conclusion de ce chapitre nous énoncerons:
La forme F_3 du langage M.A.C. est minimale.

Dans la deuxième partie de cet article, qui sera publiée, dans le prochain bulletin, nous établirons l'équivalence de puissance entre la forme réduite de la M.A.C., et la machine de Turing; ceci nous donnera l'occasion de relier deux notions, celle d'état et celle de commutation.

Bibliographie:

(1) Minsky.M. Computation. Finite and infinite machines Prentice-Hall. 1967.

(2) Nolin.L. Formalisation des notions de machine et de programme. Gauthiers-Villars. 1969.

Bianco.E. Informatique fondamentale. Birkhauser. 1979.

Autojection et Compilateurs

Procédure Formelle Symbolique

e. bianco

C.R. Subject classification informatics 4. 12 4. 21 4. 22

Résumé

Demeurant dans le cadre des machines et des langages adaptés aux machines,cet article présente l'introduction d'une notion supplémentaire: la notion de statut.Il y est étudié les implications de ce nouvel apport sur la structure du langage.

AUTOJECTION ET COMPILATEURS

PROCEDURE FORMELLE SYMBOLIQUE.

Je vais décrire l'essentiel d'un langage conçu pour la description des compilateurs autojectifs.Avant d'entrer dans le vif du sujet,je rappelle brièvement le chemin parcouru avant d'en arriver à cette étape.D'abord la machine à cases adressables,issue de la machine de Turing,et dans laquelle on fait apparaître le morceau de ruban: fini et borné.

Puis ensuite la "Procédure Formelle" dans laquelle on conserve l'avantage du ruban,mais dans laquelle,pour retrouver la propriété de procédure,perdue par rapport à la machine de Turing on rajoute une case et quelques règles de structure à respecter. On fait ainsi apparaître la notion d'index,qui se justifie alors par une utilisation précise: création d'une origine variable dans le morceau de ruban.

L'adoption d'une structure sur le morceau de mémoire -le morceau de ruban- dont l'origine est fixée par l'index,donne alors un sens précis à l'insertion de procédure et permet d'en définir une sémantique dont l'intérêt déborde du cadre d'un langage particulier.Je renvoie à l'ouvrage (1) pour le détail des propriétés de la procédure formelle.

La première utilité de la procédure formelle réside dans la prise en compte de la commutation globale,ce qui est fondamental pour la structure du programme.Cela a la conséquence directe de donner un moyen de définir proprement la notion de page,c'est-à-dire d'élément de programme,qui devient souple et adaptée à la

gestion de la mémoire.Le programme est alors considéré comme l'enchaînement de plusieurs éléments de programmes ou "procédures",définis indépendamment les uns des autres,et reliés par la commutation globale ou "insertion de procédure".

Le but de l'étape suivante est de mettre en avant une autre préoccupation,celle de la description des éléments de données à structures multiples.On peut ainsi constater qu'on introduit alternativement des notions extraites de l'organisation des données,et des notions tirées de celle des algorithmes.

J'aimerais souligner qu'il s'agit en fait de la mise en oeuvre d'une méthodologie: on dissocie certains problèmes particuliers de leur contexte,on apporte chaque fois un nouvel outil pour les prendre en compte,on vérifie l'adéquation de l'outil à la tâche qui lui a été assignée,on vérifie qu'il n'apporte pas de contradictions ou d'impossibilités par rapport à l'application des outils précédents,après cela on peut dégager un nouveau champ de recherches.

Pour l'instant je vais préciser le concept de statut défini par ailleurs en (2) à propos du MIL qui est un langage de programmation symbolique.Ici nous allons construire un langage machine.

Les opérandes ne seront pas désignés par des symboles du type identificateurs,en fait,l'écriture qui les représente fait directement appel à des organes de la machine,registres ou autres.Par contre,les désignations relatives ou absolues des cases qui se font normalement avec des entiers,se feront ici avec des identificateurs bien plus commodes.

NOTION DE STATUT.

La variété des objets traités en ordinateur ainsi que la variété des traitements qui leur sont appliqués obligent à choisir soigneusement les représentations de ces objets.A chaque jeu de variables théoriques,on fait correspondre un jeu de valeurs effectives qui doivent à un moment ou à un autre apparaître en mémoire.Et chaque valeur elle-même doit se matérialiser par le contenu d'une ou plusieurs cases de la mémoire.

En ce point deux notions se dégagent: l'une ayant trait, je dirai à la géométrie de la représentation,c'est-à-dire par exemple l'exigence en nombre de cases et comment ce nombre peut varier,l'autre s'attachant à la façon dont la valeur doit être traitée,c'est-à-dire à l'aspect algorithmique du traitement.

Dans les premiers temps de l'informatique,alors que "calcul" signifiait calcul arithmétique,on avait besoin de représenter des nombres.On distinguait les entiers et les réels.Et puis on s'est préoccupé de doubler et quadrupler les longueurs des représentations pour augmenter la précision du calcul.Ce n'était là qu'une timide amorce,le développement des modes de représentation est apparu quand on s'est apperçu que l'ordinateur savait aussi "calculer" sur du texte.La notion de calcul s'enrichissant alors singulièrement.

Il devenait dès lors nécessaire d'introduire dans les langages de programmation des supports canoniques pour prendre en compte la diversité des représentations utiles.C'est ainsi que,par exemple,avec MIL et PASCAL et dans des buts différents,

apparaissait la notion de statut.

Ce que je veux ici c'est utiliser cette notion de statut en la matérialisant à l'aide d'un type de variable sur lequel on puisse calculer.En Procédure Formelle Symbolique,seul l'aspect géométrique du statut sera explicite.Je me donne donc une définition spécifique du statut dans ce sens.

Définition de statut.

Si je me donne une structure de mémoire de référence,par exemple une suite ordonnée d'octets,le statut est la description d'une représentation donnée en terme de cette référence.

Pour introduire la composante algorithmique,il aurait suffi de particulariser le statut en le rattachant à un jeu d'algorithmes.

DESCRIPTION DU LANGAGE PROCEDURE FORMELLE SYMBOLIQUE.

Comme toujours et partout,il y a l'aspect de l'algorithme et l'aspect sous lequel sont manipulées les données.Et ce qui assure la liaison entre ces deux aspects,c'est la désignation des opérandes.Je vais préciser ce dont j'ai besoin et sous quelle forme je l'exprime.

Le PFS doit me servir en principe,à décrire des compilateurs en un premier temps,des systèmes ensuite.Je vais me donner ce dont j'ai besoin pour la compilation,puis je complèterai ensuite.L'expérience montre qu'on a besoin de deux sortes de structures: les files à arrangement répétitif et les files à arrangement disparâte.Je me munis en conséquence de deux sortes de constructions,d'une part les objets issus des problèmes à programmer et d'autre part les sortes de rayonnages qui vont

me servir à ranger ces objets.Je crée donc des structures adaptées et je me fixe le calcul qui permet d'y localiser les objets.

Les supports.

Je dispose d'une configuration-support de base à partir de laquelle je définis les moyens de construire tous les autres objets-supports.Je vais essayer de m'écarter le moins possible de la procédure formelle.Les objets-supports sont des cases ou des agglomérats de cases.

Elément-minimum.

C'est l'objet-support élémentaire,par exemple: l'octet. En général c'est la case de la machine dont on se sert.

Les files.

Je définis deux sortes de files.Faisant référance au MIL la première traduit les propriétés de la file périodique je l'appelle "tableau",l'autre,les propriétés de la file apériodique,et je l'appelle "file dynamique".

Les tableaux.

Les files sont déclarées mais ne portent pas de nom,leur structure est déclarée en donnant la suite des éléments qui les composent.Ces éléments,eux sont nommés.Prenons un exemple de déclaration de tableau:

30 (ident(5) , no(2) , dim(2) , type(1)):x1:x2;

Cette écriture signifie qu'on dispose de 4 supports dénommés ident,no,dim,type et qui comprennent chacun successivement 5,2,2,1 cases ou éléments-minimum.Le groupe formé par ces 4 supports,constitue également un support de 5+2+2+1=10 cases.

Enfin on dispose d'un support total qui fait 30 fois celui-là et qui représente la file.

Il faut remarquer que le tableau est défini par lignes, le premier nombre indique le nombre de lignes que le programmeur prévoit.La ligne elle-même est définie par la suite des éléments qui la composent.Chacun de ces éléments est nommé.Et entre-parenthèse le nombre indique la dimension du support correspondant.La ligne peut comporter autant d'éléments qu'on veut et au moins un.

Je préciserai plus loin le rôle que joue l'identificateur de l'élément dans l'écriture de l'opérande.Ainsi que son intérêt.

La référence.

Il faut un moyen,maintenant pour atteindre à ces divers niveaux de supports.J'appelle les variables désignées par x1 et x2 des "références".Les références se trouvent être attachées à un tableau déterminé.Ce fait est marqué dans ce langage par la relation qui existe entre l'identificateur de référence et la déclaration de tableau.La conséquence normale est qu'à deux tableaux différents doivents s'attacher des jeux d'identificateurs de référence différents.

La référence sert à désigner une ligne complète du tableau. Cela signifie que la variable référence doit pouvoir prendre autant de valeurs différentes qu'il y a de lignes dans le tableau.Comme il est commode en plus que les lignes soient ordonnées dans le tableau alors alors on peut se donner les modalités suivantes du calcul des valeurs de la référence:

I1 x1 := initial ;

I2 x1 := 5 →x1 ;

I3 x2 := ultime ;

I4 x2 := n ← x2 ;

L'instruction I1 donne à la référence x1 une valeur telle que l'écriture suivante: I5 ,désigne dans son ensemble la première des 30 lignes du tableau.

I5 Γ_{x1}

I6 $\Gamma_{x1,ident}$

Alors que l'écriture I6 désigne dans la ligne en question l'élément repéré par "ident".

Remarque.

Comparer les écritures I5 et I6 à des écritures comme I7 et I8 de la procédure formelle,montre la correspondance qui existe entre les désignateurs: x1 et ident avec leurs homo-

I7 $\Gamma_{21,5}$

I8 Γ_{3}

logues.De même que 21 et 3 qui indiquent une case qui doit contenir une adresse,x1 est de la dimension d'une adresse. Alors que ident est,comme 5 ,une correction d'adresse.

Sous sa forme I5 ou I6 l'opérande,comme dans les langages machines,fait appel aux organes de la machine,ici,d'abord l'index I dont le contenu indique l'origine relative,et ensuite le premier indice renvoit à une variable -au contenu d'une case- qui fournit une adresse relative à cette origine. Les identificateurs n'interviennent plus,toujours comme dans

les langages machines,que pour éviter au programmeur d'avoir à construire des listes de variables repérées uniquement par des entiers,ce qui n'est pas très lisible.

Par rapport à la simple procédure formelle il a donc été introduit une notion importante: le statut,sous la forme d'une liste d'éléments réproductibles en séries finies dans le tableau,et une commodité: dans la liste des éléments,chacun d'eux est désigné par un identificateur.

Une organisation de tableau telle que je viens de la définir est à la fois simple et d'usage courant.Mais on a aussi besoin de types d'organisation plus complèxes.

La file dynamique.

Il arrive qu'on ait besoin d'entasser des paquets d'éléments dont les formes sont différentes,il devient difficile d'employer une structure de tableau,car elle est assez rigide.Je vais m'intéresser à quelquechose de plus compliqué mais de beaucoup plus souple.Je prends un exemple:

```
10 000,1 000,4 ( top(1) , adres(2) ),
    1 ( top1(1) , ad1(2) , ad2(2) ),
    3 ( top1(1) , adr2(2) , index(2) ),
    2 ( top1(1) , const(1) ): y1: y2: y3 ;
```

Tout ceci signifie qu'on dispose d'un support de 10 000 éléments minimum,que sur ce support on peut placer au maximum 1 000 paquets d'information,ou encore 1 000 lignes dont les statuts sont à choisir entre les 4 statuts offerts à la suite. Pour le premier statut décrit entre parenthèses,comme les suivants,les éléments portent les noms:

"top" et "adress"

pour le second les éléments portent les noms:

"top1" et "ad1" et "ad2"

et ainsi de suite.

Le statut.

En Procédure Formelle Symbolique,le statut est défini par une liste de statuts d'éléments.Le statut d'élément est un entier qui représente un nombre d'éléments-minimum.

Cette définition va pour le tableau comme pour la file dynamique.

Je me donne alors les moyens d'atteindre à un élément d'une telle file.Les identificateurs y1,y2 et y3 représentent des références attachées à la file dynamique déclarée ci-dessus. La valeur d'une telle référence est en fait un doublet constitué par la désignation du paquet d'élément ou ligne,et par la désignation du statut qui décrit la structure de la ligne en question.Par exemple:

```
I9      y2 := initial , 3
I10     y2 := 3 → y2 , 1
I11     y1 := ultime , 1
I12     y1 := 5 ← y1 , 2
```

L'instruction I9 signifie qu'il est attribué à la référence y2 une valeur composée de la désignation du premier élément de la file et de la désignation du statut N°3.L'instruction ,elle,indique qu'on attribue à y2 le couple constitué par la valeur du nom du 4^e élément,et par le nom du statut N°1. Le sens des deux autres instructions est alors évident.

Exemples.

En compilation je prends pour exemple le programme qui, une variable venant d'être déclarée,vérifie que dans le tableau des variables l'une d'entre elles ne possède pas le même identificateur,et sinon place la nouvelle variable dans le tableau.

```
30 ( ident(8) , type(1) , no(2) , dim(2) ): td: tc: tf;
 1 ( id(8) , tp(1) , num(2) , d(2) ): x;
            proc decvar (td,tf,tc,x);
            x := initial ;
            tc := initial ;
e1 : itère rechnom (tc de td à tf) ouverture 5 ;
  aiguillage identité (Γ tc,ident = Γ x,id : er sinon suite)
            suite : répétition rechnom ;
            Γ tf := Γ x ;
            sortie identité ;
            er : Γ y,er := "var déja décl" ;
            sortie identité ;
            noeud identité ;
            fin
```

En dehors des problèmes que pose la commutation et qui sont envisagés plus loin,je fais l'hypothèse qu'il existe une file décrite ailleurs et dont y est une référence.Dans cette file on note qu'on a trouvé une erreur.

Pour se donner un exemple de file dynamique,prenons un cas simple de construction de programme généré en langage machine. Les 4 statuts décrivent les seules possibilités de structure

du code à construire,et imaginons que les valeurs obtenues par le calcul soient dans le tableau de référence "const".

```
6000 , 120 , 1 ( to(1) , index(1) , ad1(2) , ad2(2) ),
             2 ( to(1) , index(1) , ad1(2) ),
             3 ( to(1) , ad1(2) , ad2(2) ),
             4 ( to(1) ): po : pc ;
( (cop(1) , ind(1) , adres1(2) , adres2(2) ) : constr ;
             - - - - - - - - - - - -
             - - - - - - - - - - - -
             pc := pc , 3 ;
             Γpc,to := Γconstr,cop ;
             Γpc,adr1 := Γconstr,ad1 ;
             Γpc,adr2 := Γconstr,ad2 ;
             pc := → pc ;
             - - - - - - - - - - - - -
```

On remarque que la valeur de la référence peut être affectée en deux temps si on le désire.La valeur-statut n'étant attribuée qu'au moment où on la connaît.

Pour relire un élément de file dynamique,deux cas peuvent se présenter.Quand on connaît le statut de cet élément alors on est ramené au cas du tableau.Si les circonstances veulent qu'on ne puisse plus se souvenir de son statut,il suffit de le rechercher en employant un aiguillage:

```
aiguillage stat ( statut pc = ( 1:e1 , 2:e2 , 3:e3
                              , 4:e4 ) ) ;
```

qui signifie que si le statut attribué à l'élément que désigne la référence pc, est le statut N°1 alors on va à l'étiquette e1 s'il est le statut N°2 on va en e2 etc.

Commutation.

Ce langage,je l'ai dit est destiné à matérialiser l'autojectivité.C'est donc à propos de la commutation que j'introduis des formes particulières.J'avais déja défini dans le MIL la notion de commutation finie-bornée.Je me donne deux manières de calculer cette commutation qui m'assurent d'un déroulement qui possède la propriété du fini-borné.L'aiguillage qui exprime les choix descendants et la boucle qui décrit les itérations.

Aiguillage.

Cette forme représente la commutation aval qui découle d'un choix.L'aiguillage se construit avec un en-tête , une fin de - dérivation et un noeud .

En-tête d'aiguillage.

Cette partie contient le nom de l'aiguillage et la liste des conditions,chacune d'entre elles est attachée à une étiquette qui désigne la suite du programme quand la condition est vérifiée.

La liste des conditions peut être sous la forme d'un opérande suivi d'un "=" suivi d'une liste de valeurs possibles attachées chacune à une étiquette.Quand la valeur de l'opérande est l'une de celles proposées dans la liste,alors on va à l'étiquette qui lui est attachée.

Les conditions peuvent également être exprimées par des suites de comparaisons entre deux opérandes,chacune étant attachée à une étiquette.C'est alors la première qui se vérifie qui renvoit à son étiquette.

Sortie.

Toute dérivation dans l'aiguillage se ponctue d'une sortie

dans laquelle on précise l'identificateur de l'aiguillage.

Noeud.

Lorsque l'ensemble des dérivations est écrit ,on le ponctue lui-même par une instruction noeud ,qui précise le nom de l'aiguillage,et qui indique sur quel point toutes les dérivations se referment.

La forme suivante illustre l'un des aspects de l'aiguillage:

```
Aiguillage top ( r_p,car = ( 0:e0 , 1:e1 , 2:e2 ) );
                e0 : C1
                   sortie top ;
                e1 : C2
                   sortie top ;
                e2 : C2
                   sortie top ;
                   noeud top ;
```

les C1, C2, et C3 sont des éléments de programme qui traitent de chaque dérivation.

La boucle.

La commutation aval est représentée par l'aiguillage,la commutation amont,celle qui est susceptible de provoquer des itérations,est traduite exclusivement sous la forme de ce que j'appellerai la boucle.En toute généralité une itération ne saurait être bornée a priori.J'introduis donc un mode d'organisation: l'ouverture,qui redécoupe le déroulement de la boucle en tranches maîtrisables.

La boucle se ponctue par une répétition qui précise le nom de la boucle.Si la boucle comporte de l'ouverture alors il faut que soient mis en place des jeux de commutation et de

coupure dans la chaîne des procédures qui s'insèrent les unes dans les autres,et qui s'achève par la procédure où se trouve la boucle dont on se préoccupe.

En-tête de boucle.

A titre d'exemple,l'en-tête se présente ainsi:

itère ba3 (z de to à tq) ouverture 5 ;

Cela signifie que,quel que soit le nombre de valeurs que peut prendre z ,la boucle ba3 ne se déroulera que par tranches de 5 tours au maximum.

Répétition.

L'instruction répétition qui ponctue la boucle,précise le nom de cette dernière et l'identificateur qui est attaché au fin de la procédure.Ce nom du fin de la procédure n'est pas indispensable.Il ne sert à rien si la procédure n'a pas d'ouverture,mais il est commode pour la compilation dans le cas contraire.

Commutation.

J'imagine la chaîne des procédures qui s'insèrent les unes dans les autres,qui commence à la plus extérieure et s'achève à celle qui contient la boucle,je vais devoir en tête de chacune de ces procédures,placer une instruction commutation. Dans cette instruction on précise la paire formée par le nom de la boucle et l'étiquette attachée à l'instruction d'insertion correspondante.

Coupure.

De la même manière,dans chacune de ces procédures,mais à la suite de l'instruction d'insertion dont je viens de parler,je porte une instruction de coupure en précisant un dou-

blet constitué du nom de la boucle et de l'étiquette de l'instruction fin de la procédure.

Cas d'une boucle qui n'en contient aucune autre.
Si la boucle a de l'ouverture n ,on va parcourir un chemin égal à n fois le corps de la boucle.Puis si la boucle n'est pas achevée,il faut retourner au système,et faire en sorte que le prochain quantum de travail soit constitué des seuls n tours suivants de la boucle.C'est précisément ce à quoi servent les coupures qui n'ont d'autre rôle que de permettre le retour direct au système sans dérouler autrechose, ainsi que les commutations qui ramènent directement du système au coeur de la boucle.

Cas d'une boucle qui en contient une autre.
Coupures et commutations sont évidemment des aller-à conditionnels,dont la condition ne se vérifie que lorsqu'on travaille dans la boucle qui va avec.On voit que si une boucle est dans une autre,il y aura tant qu'on est dans la boucle intérieure,deux conditions de coupure et de commutation vérifiées.Or, c'est de toute évidence la boucle interne qui doit se dérouler en priorité,l'ordre de présentation des coupures et des commutations en découle tout naturellement.

à suivre

LE GRAFCET

ETUDES ET REFLEXIONS

Suite du N°5

P. ISOARDI

C.R. Subject classification informatics 4. 20

GRAFCET ET SYSTEMES LOGIQUES SEQUENTIELS

L'étude précédente nous a permis de présenter le GRAFCET : Sa conception, ses avantages et ses inconvénients.

Deux questions essentielles restent posées :

- Le GRAFCET suffit-il pour représenter n'importe quel cahier des charges ; Bien entendu dans son cadre naturel, c'est-à-dire où les spécifications s'expriment uniquement en termes de variables logiques ?

- Qu'apporte le GRAFCET par rapport aux approches classiques et plus particulièrement la méthode d'Huffman ?

Ce chapitre apporte une réponse à ces questions.

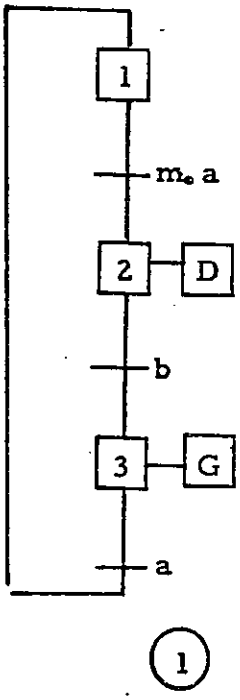
I. Equivalence GRAFCET ⇔ Matrice des états.

Cette analyse peut se placer également par rapport au diagramme des phases ou des graphes de fluence qui sont de même conception que la matrice des états alors qu'il n'en est pas de même pour le GRAFCET ; Il est un outil de représentation permettant de poser les problèmes alors que les méthodes traditionnelles offrent des moyens pour les résoudre.
En effet, une matrice d'états se construit :

a) - En observant les instructions entre le processus et l'automate
b) - En donnant une traduction de ces échanges sous forme de table et sans omettre aucun détail.

1) Peut-on à partir d'un GRAFCET obtenir une table d'états ?

L'inverse est évident et sans intérêt. L'étude de l'exemple introductif (page 4) donne une illustration de la réponse. Le Cahier des charges s'exprime par le GRAFCET de la figure 1 .



① ② ③

		entrées								sorties	
		$\bar{a}b\bar{m}$	$\bar{a}bm$	$\bar{a}bm$	$\bar{a}b\bar{m}$	$ab\bar{m}$	abm	$ab\bar{m}$	$ab\bar{m}$	G	D
situations	S1	①	①	①	①	①	2	2	①	0	0
	S2	②	②	3	3	3	3	②	②	0	1
	S3	③	③	③	③	1	1	1	1	1	0

Si on appelle SITUATION d'un automatisme l'ensemble de ses étapes actives à un moment donné, il y a correspondance directe entre cette notion de situation dans un GRAFCET et celle d'ETAT STABLE dans la représentation classique d'un système séquentiel.
On peut ainsi établir à partir du GRAFCET précédent la matrice primitive des états.

La situation initiale S1 = (1) correspond à un état stable ① lorsque : $\bar{m}.a.b$, $m.a.\bar{b}$, $m.\bar{a}.b$, $m.\bar{a}.\bar{b}$, $\bar{m}.\bar{a}.b$, $\bar{m}.\bar{a}.\bar{b}$ sont vraies.
Les combinaisons d'entrées qui vont permettre de passer dans l'état stable ② ≡ situation S2 = (2) sont : $m.a.b$ et $m.a.\bar{b}$, c'est-à-dire lorsque la condition $m.a$ est vraie ; On est alors dans un état qualifié de TRANSITOIRE noté 2 .
On peut poursuivre le même raisonnement à partir des situations S2 et S3 (état stable ③) . On obtient la matrice primitive des états de la figure 2 .

Cette matrice ne contient aucune case "interdite" ou "impossible", car toute combinaison d'entrée qui ne fait pas évoluer la situation du GRAFCET le laisse dans la situation actuelle c'est-à-dire dans l'état stable correspondant.

Considérons à présent la table des états obtenue par les méthodes traditionnelles de description des échanges entre l'automatisme et le processus.

		entrées								sorties	
		$\bar{a}\bar{b}m$	$\bar{a}\bar{b}m$	$\bar{a}\bar{b}m$	$\bar{a}\bar{b}m$	$\bar{a}\bar{b}m$	$\bar{a}\bar{b}m$	$\bar{a}\bar{b}m$	$\bar{a}\bar{b}m$	G	D
situations	S1							2	①	0	0
	S2	②	②	3	3			②	-	0	1
	S3	③	③	③	③			-	1	1	0

Figure 3

Les cases hachurées sont considérées comme "impossibles" : en effet, les capteurs a et b ne peuvent être simultanément actifs.

L'action sur le bouton poussoir m n'a aucune influence pendant le trajet du chariot. Par contre (case [-]) :

a) - on interdit que le bouton poussoir m soit relâché avant le démarrage du chariot, de l'état stable ① .

b) - Faut-il interdire que m soit actif lors de l'arrivée du chariot en A ? Sinon le cycle risque de ne pas s'arrêter.

Si ces deux matrices ne sont pas identiques, elles ne sont pas non plus incompatibles. La matrice de la figure 2 couvre celle de la figure 3 : on n'a fait que rendre stables des cases indifférentes de la table de la figure 3 où la description de l'automatisme est beaucoup fine et précise. Ainsi, a tout GRAFCET on peut faire correspondre une matrice d'états, mais ce n'est pas celle que le concepteur aurait établi directement.

La force des méthodes traditionnelles c'est qu'elles permettent une analyse fouillée du fonctionnement de l'automatisme. C'est aussi leur faiblesse car dès que la taille du problème augmente, le concepteur est vite découragé par :

a) - le maniement très lourd de tableaux de grande dimension.

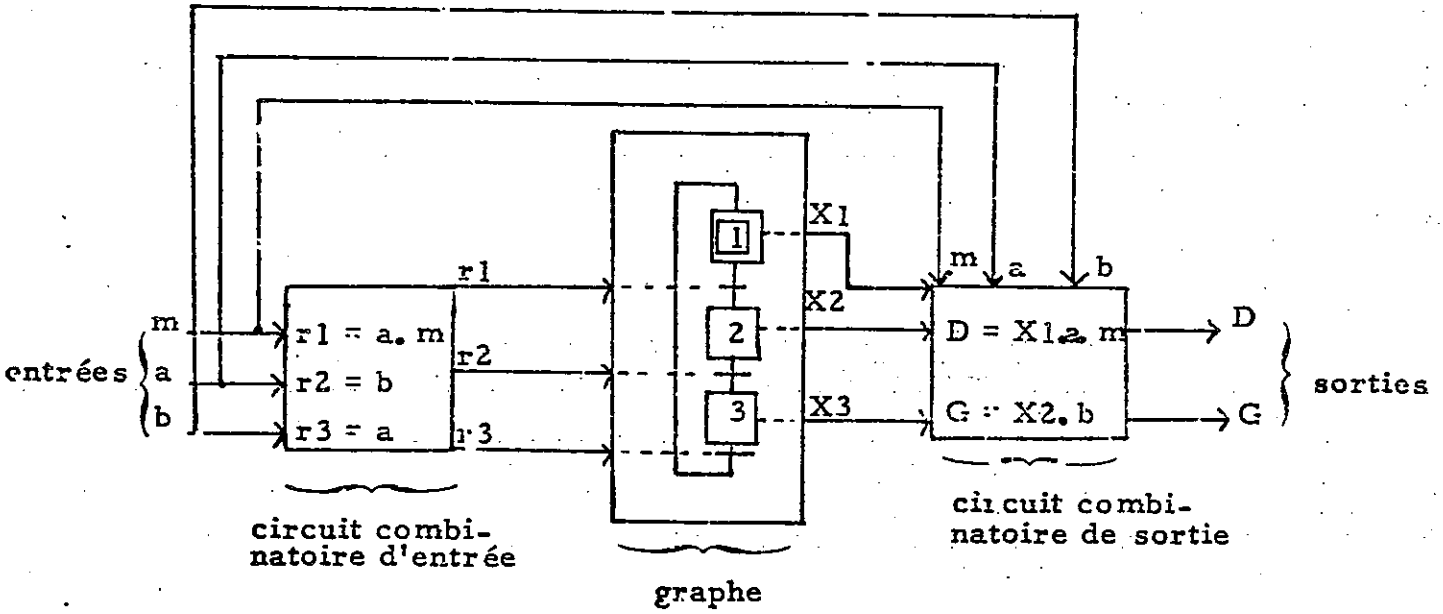
b) - la prise en compte fine et globale de tout ce qui se passe, sans possibilité de sélectionner les seules informations utiles en ignorant celles qui ne sont pas significatives.

Ainsi, si on considère l'exemple précédent, lorsque l'opérateur appuie sur le bouton poussoir m pour commander le départ du chariot, l'information importante à cet instant est l'apparition de m . Mais il faut aussi considérer que l'opérateur va relâcher le bouton, qu'il peut le faire avant ou après que le chariot ait quitté sa position initiale ... autant d'informations qui en fait n'ont aucune influence sur le déplacement du chariot et qui compliquent considérablement la description du fonctionnement. En contre partie, l'utilisation du GRAFCET impose de n'oublier aucune réceptivité et qu'elles soient de plus suffisantes. De tels oublis sont évidemment impossibles dans l'approche classique où toutes les combinaisons d'entrée sont systématiquement examinées ainsi que leurs conséquences sur l'évolution de l'automatisme.

II GRAFCET et Machines séquentielles.

Le GRAFCET par ses propriétés facilite la description du fonctionnement de l'automatisme sans simplifier sa réalisation en la rendant la moins séquentielle possible ; Ce qui est, ne n'oublions pas l'objectif des méthodes traditionnelles de synthèse.

On peut représenter le GRAFCET de l'exemple précédent par le schéma de la figure suivante :



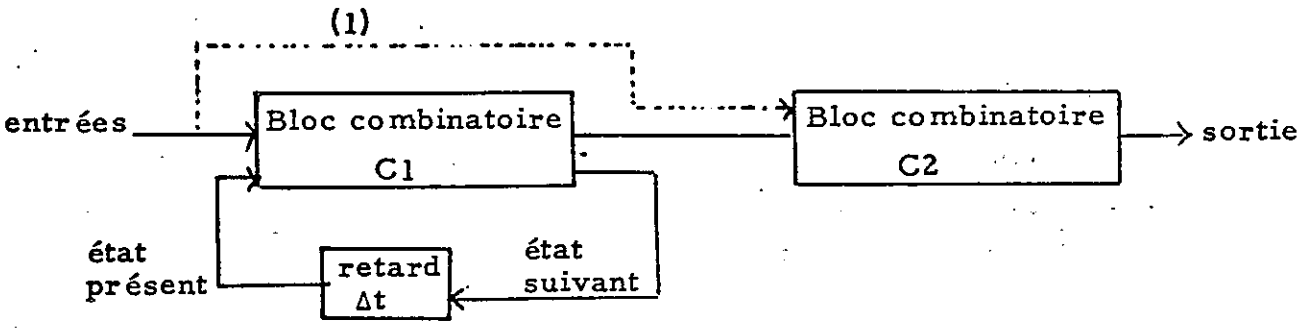
On distingue dans ce schéma :

- un bloc combinatoire d'entrée qui à partir des variables d'entrée élabore les réceptivités qui sont associées aux transitions du graphe.

- le graphe qui est un bloc séquentiel puisque l'étape caractérise un comporte ment invariant de tout ou partie du système.

- un bloc combinatoire de sortie qui à partir de l'état actif ou inactif des étapes du graphe est éventuellement des variables d'entrée élabore les sorties.

Le mode de représentation traditionnel des machines séquentielles est représenté par la figure suivante :



Ceci est une interprétation d'une machine de MOORE ou de MEALY (option 1).

Ce mode de représentation classique des machines séquentielles n'est composé que de circuits combinatoires. C'est de façon artificielle qu'il représente un système séquentiel ; Alors que le modèle GRAFCET a un bloc séquentiel : le graphe.

C'est là que réside l'intérêt du GRAFCET car le contenu du cahier de charges s'exprime par des évolutions séquentielles et/ou simultanées, même lorsqu'elles sont de nature combinatoire. Une machine séquentielle est un GRAFCET particulier où une seule étape peut être active à un moment donné et où la notion de réceptivité n'est pas utilisée.

C'est aussi la faiblesse du GRAFCET ; Bien qu'il puisse être simplifié en fusionnant des étapes, nous dirons qu'il n'est pas dans ses objectifs de représenter le fonctionnement d'un automatisme de la façon la moins

séquentielle possible. Il n'est pas question d'obtenir un GRAFCET "optimal" en tentant de le réduire. Nous verrons plus loin que cette affirmation est basée sur les moyens technologiques disponibles aujourd'hui.

DU GRAFCET A LA REALISATION DE L'AUTOMATISME

Ce dernier chapitre qui conclue l'étude et la réflexion précédente est une ouverture sur l'utilisation de moyens matériels ou logiciels adaptés à l'implantation du GRAFCET.

I. Synthèse d'un système logique.

Le GRAFCET est un outil de spécification ; La réalisation matérielle de l'automate est une étape supplémentaire issue de la précédente.

Nous allons dégager ici l'ensemble des éléments à considérer pour effectuer ce passage.

Traditionnellement, la synthèse d'un système logique comporte trois phases :

a) - Représentation du cahier des charges.
b) - Méthode de synthèse. Le résultat obtenu est en général un ensemble d'équations que l'on cherche à minimiser.
c) - Utilisation d'une technologie particulière pour matérialiser la solution obtenue. Le plus souvent, ce choix est déjà fait, dans les méthodes classiques, au niveau de l'étape précédente.

La minimisation des circuits entraîne évidemment une diminution du coût du matériel intervenant dans la réalisation. En contre partie, face à la complexité des problèmes abordés, elle nécessite des études qui prennent la part la plus importante du temps de conception.

Aujourd'hui, l'évolution spectaculaire de la technologie permet une diminution considérable du prix des composants de base (combinatoires et surtout séquentiels) ainsi qu'une nette amélioration des performances.

Le problème actuel n'est donc plus une "optimisation" mais la recherche d'une certaine harmonie et coordination entre les différentes phases de la conception.

L'approche par le GRAFCET, qui permet de réduire considérablement le temps d'étude, va dans le sens de ces préoccupations.

II. Implantation du GRAFCET.

Le fonctionnement de l'automatisme doit être une image des évolutions du GRAFCET. Pour cela, à chaque objet de ce dernier on doit associer une traduction technologique dans la réalisation.

1) Etape ; Transition.

On retrouve là l'approche de l'exemple introductif (ch. III)

- A chaque étape du graphe on associe une mémoire.
- Les transitions ont pour fonction d'activer les mémoires (étapes) ou de les effacer en respectant la structure du GRAFCET ainsi que les règles d'évolution.

2) Règles d'évolution.

En reprenant les règles d'évolution (ch. IV) on constate que :

- Lors de la mise sous tension la règle 1 impose l'activation des mémoires matérialisant les étapes initiales,
- les règles 2 et 3 laissent apparaître des opérations simultanées. Cette notion est, technologiquement difficile à traduire. On est donc amené à envisager :
  - Soit une réalisation synchrone,
  - Soit des réalisations asynchrones si l'analyse ne laisse pas apparaître de conséquences néfastes (aléas),
- Si les règles 4 et 5 sont respectées, la réalisation doit être obligatoirement à évolutions synchrones, c'est-à-dire une synchronisation, a un instant donné, du franchissement de toutes les transitions franchissables.

3) Actions ; Réceptivités.

Pour que toute réceptivité soit prise en compte, il faut tenir compte des contraintes technologiques des entrées : durée de maintien, retard, ...

La durée de génération des actions impulsionnelles doit être parfaitement définie.

III. Réalisations cablées. Réalisations programmées.

Nous allons rapidement faire le point des différentes approches de réalisations cablées ou en logique programmée permettant l'implantation du GRAFCET.

1) Réalisation à partir de mémoires.

Voir l'exemple introductif (ch. III).

A chaque étape on associe une mémoire. Il ne reste plus qu'à établir les équations d'activation et d'effacement de ces mémoires en tenant compte des réceptivités et de la structure du graphe ; Sans oublier non plus les contraintes technologiques ci-dessus mentionnées.

2) La logique programmée.

On peut implanter un GRAFCET dans un P. L. A. (Programmable Logic Array) ou utiliser un microprocesseur. A cette dernière solution se rattache la notion de programme enregistré, exécuté séquentiellement.

3) Les automates programmables.

C'est un matériel de nature informatique mais conçu pour des automaticiens.

Avec un seul processeur, le parallélisme est traité séquentiellement au cours d'un cycle de calcul. C'est la rapidité du temps d'exécution qui donne l'impression de la simultanéité et de la prise en compte instantanée des entrées. Cette "impression" doit tout de même respecter les normes de durée définies dans le cahier des charges. Cet automate ne peut pas traiter simplement des GRAFCET à évolutions synchrones.

Pour implanter un GRAFCET à évolutions simultanées, on le décompose en séquences. Le système est construit autour :

- d'une unité de recherche des étapes actives à l'instant donné,
- d'une unité de traitement, le plus souvent microprogrammee au cours de laquelle on exécute uniquement la partie du programme spécifique de la situation actuelle : actions des étapes actives à l'instant donné et évolutions sur les réceptivités.

Ces deux unités travaillent en parallèle.

Ces réalisations proches dans leurs principes, ont en commun la notion d'exécution sélective des programmes correspondant aux étapes acti et le traitement séquentiel des évolutions simultanées.

Pour conclure, les architectures de processeurs en parallèle apporteront très certainement une ouverture supplémentaire dans la réalisation des automatismes.

AIDE A LA CONSTRUCTION DE PROGRAMMES BASIC SUR SYSTEMES APPLE

J. PH. LEHMANN

C.R. Subject classification informatics 4. 20 4. 30

Résumé

Est-il possible de négliger l'importance d'un langage comme BASIC pour les utilisateurs? La quantité et la qualité des réalisations menées à bien avec son aide poussent tout naturellement à réutiliser des programmes existants dans de nouveaux programmes.Cela BASIC ne le permet pas directement.Il faut faire intervenir des propriétés du système.Cet article expose une méthode pour y parvenir.

AIDE A LA CONSTRUCTION DE PROGRAMMES EN BASIC SUR SYSTEMES APPLE.

(J.Ph.Lehmann)

I. INTRODUCTION

Que ce soit à l'occasion de l'élaboration d'un programme complexe, ou de plusieurs programmes développés autour d'un thème commun, le problème essentiel que rencontre le programmeur apparaît, quand il s'agit de composer entre eux, différents programmes ou tronçons de programme, qui par ailleurs, ont déjà été mis au point.

Un premier aspect de la question, se situe au niveau du passage des paramètres; celui-ci n'a aucun caractère automatique. L'utilisateur a la charge d'organiser ce passage, en prenant les précautions annexes qui s'imposent, notamment en vérifiant que les variables locales du programme appelé, ne sont pas déjà incluses dans le programme appelant, en tant que paramètre, ou même en tant que variable locale active, au moment de l'appel. A cet égard, seule une certaine pratique de la programmation, en organisant un choix contrôlé des noms des variables et des paramètres, permet de résoudre le problème.

Nous fixerons notre attention sur un second aspect du problème, le plus important, nous semble-t-il, puisqu'à la différence du précédent, la seule habileté dans la programmation, ne permet pas de le résoudre: lorsqu'on désire constituer un programme, faisant intervenir des subroutines déjà existantes, il n'est pas possible d'utiliser des LOAD successifs, puisque leur exécution entraîne l'écrasement de tout programme déjà chargé en mémoire centrale. L'instruction CHAIN, lorsqu'elle est disponible, ce qui n'est pas toujours le cas, ne constitue qu'un palliatif partiel; il peut se faire qu'on veuille, non seulement disposer des résultats déterminés lors de l'exécution d'un programme, mais encore de ce programme lui-même. Dans ces conditions, seule l'instruction EXEC, peut permettre d'apporter une solution.

II EXEC

EXEC s'applique à un fichier texte; sa fonction peut être définie en disant qu'elle simule un opérateur fictif contrôlant le système depuis le clavier, à cela près que la suite des actions de cet opérateur est en fait déjà déterminée par le contenu du fichier sur lequel précisément s'applique EXEC.
Tout se passe comme si, la suite des codes ASCII, habituellement transmise au système depuis le clavier, était communiquée depuis le disque. En réalité, le D.O.S. supervise ce transfert.
On tient là un avantage considérable: de même que les codes transmis depuis le clavier, lorsqu'ils représentent une succession de lignes de programme, sont empilés dans la mémoire sous le contrôle du BASIC actif à cet instant, le contenu du fichier en cours d'EXECution, viendra résider en mémoire sans rien écraser de ce qui s'y trouvait déjà, à la condition que le texte même du fichier soit une succession de lignes de programme.
Dès lors, il suffit de constituer sous forme de fichiers textes, l'ensemble des programmes et sous-programmes, en définissant pour chacun des implantations distinctes: pour construire un programme bâti avec ces composants prédéfinis, on appliquera EXEC sur chacun des fichiers.
Le problème revient alors à donner un procédé automatique, permettant de constituer de tels fichiers; le bloc élémentaire autour duquel tout sera articulé est le suivant:

```
1   D$ = CHR$(4)
2   PRINT D$;"OPEN LISTING"
3   PRINT D$;"WRITE LISTING"
4   POKE 33,33
5   LIST XY,ZT
6   PRINT D$"CLOSE"
7   TEXT:END
```

Cette methode, simple et efficace, indiquée dans le manuel du D.O.S. d'APPLE, consiste à ouvrir un fichier texte, puis, après avoir défini une largeur de fenêtre qui force l'instruction LIST à compacter l'édition des lignes de programme, à LISTer le tronçon de programme à transformer; LIST n'étant elle-même qu'une forme évoluée de PRINT, tout se passe alors comme si on exécutait l'ordre d'impression des chaînes de caractères constituées par les lignes de programme spécifiées; le fichier LISTING étant ouvert, il se remplit du texte voulu.

III. LA PROCEDURE PROC1

Dans les quatre pages suivantes, nous donnons le contenu du fichier PROC1.

Plus loin nous commentons et expliquons la structure de ce fichier et ses fonctions.

PROC1 va lui même constituer un fichier texte.
Pour lancer la procédure, les conditions initiales sont les suivantes:

- Le disk sur lequel se trouve PROC1, doit être mis dans le drive D1.
- Le programme qu'on veut transformer en fichier texte doit être chargé en mémoire centrale.
- Pour gagner du temps, il est préférable d'installer en D2 le disk sur lequel sera écrit le programme sous forme texte.

Remarques:
Au cours de l'exécution de PROC1, un fichier auxiliaire sera créé, de nom T1, en D1. Il faut prendre garde de ne créer sur le disk concerné aucun fichier portant ce nom; il serait détruit. De même le programme sous sa forme habituelle, sera sauvé en D1, sous le nom DR.PROG. Là aussi, il est indispensable que sur ce disque ce nom soit réservé.
Sur le disk en D2(qui recevra le texte), un fichier texte provisoire sera créé, sous le nom PROV. Il faut également réserver ce nom sur le second disque.
Notons enfin qu'il aurait été tout à fait possible de contrôler les erreurs éventuelles commises par l'opérateur, au moment de l'initialisation du processus;nous avons préféré supposer que l'utilisateur appliquerait correctement les directives très simples qui lui sont données.

III.1 LE FICHIER PROC1

Tronçon 1

```
Ø           UNLOCK DR.PROG
1           DELETE DR.PROG
2           SAVE DR.PROG
3           UNLOCK T1
4           DELETE T1
5           FP
```

Bout 1

```
6           (Blanc)
7  Ø D$ = CHR$(4):PRINT D$;"CLOSE PROC1"
8  1 TEXT:HOME:INVERSE:PRINT"POUR QUITTER, TAPER:CTRL-A":
     POKE 34,2:NORMAL:HOME
9  2 PRINT"NOM DU PROGRAMME ?":PRINT:PRINT
1Ø 3PRINT"AU PLUS 15 CARACTERES":PRINT"AU MOINS 1 CARACTERE"
11 4PRINT"PAS DE BLANCS NI DE"+CHR$(34)+","+CHR$(34)+"OU" +
     CHR$(34)+":"+CHR$(34)
12 5 INPUT A$: L=LEN(A$): X$=A$: HOME
13 6 IFA$=CHR$(1) THEN POKE 34,Ø:HOME:PRINTD$;"EXEC PROC1,D1,
     R127":END
14 7 IF L>15 THEN HOME: GOTO2
15 8 IF L = Ø THEN HOME: GOTO2
16 9 FOR I=1 TO L
17 1Ø IF MID$(X$,I,1)="," THEN HOME:GOTO2
18 11 IF MID$(X$,I,1)=":" THEN HOME:GOTO2
19 12 U=ASC(MID$(X$,I,1))
2Ø 13 IFU>95 THEN U=U-64:GOTO13
21 14 IFU>9Ø THEN HOME:GOTO2
22 15 IFU<33 THEN HOME:GOTO2
23 16 NEXT I
24 17 POKE 34,1:HOME:INVERSE:PRINT"NOM: "+A$:NORMAL:POKE 34,3
25 18 PRINTD$;"OPEN T1":PRINTD$;"WRITE T1":PRINT A$:PRINT D$;
     "CLOSE"
26 19 POKE 216,Ø:PRINT"INSERER EN D2, LE DISK DES SBR.TEXTES."
27 2Ø PRINT"PUIS, TAPER 1 POUR CONTINUER."+CHR$(7)
28 21 A$="":INPUTA$:IF A$=CHR$(1) THEN POKE 34,Ø:HOME:PRINT D$;
     "EXEC PROC1,D1,R127":END
29 22 IF A$><"1" THEN HOME:GOTO2Ø
3Ø 23 ONERR GOTO 25
31 24 PRINTD$:"RENAME U,U,D2":GOTO 26
32 25 Y=PEEK(222): IF Y=8 GOTO 19
33 26 POKE 216,Ø:PRINTD$;"EXEC PROC1,D1,R36":END
34          (Blanc)
```

Tronçon 2

```
35          RUN
36          TEXT
37          HOME
38          LOAD DR.PROG
```

Bout 2

```
39              (Blanc)
40   8 D$ = ""                          (Il y a CTRL-D)
41   9 PRINTD$;"OPEN PROV,D2"
42  10 PRINTD$;"WRITE PROV"
43  11 POKE 33,33
44  12 LIST Ø,7
45  13 PRINTD$;"CLOSE"
46  14 TEXT
47  15 END
48              (Blanc)
```

Tronçon 3

```
49              RUN 8
50              LOAD DR.PROG,D1
```

Bout 3

```
51              (Blanc)
52   Ø D$ = ""                          (Il y a CTRL-D)
53   1 PRINTD$;"APPEND PROV,D2"
54   2 PRINTD$;"WRITE PROV"
55   3 POKE 33,33
56   4 LIST 8,32567
57   5 PRINTD$;"CLOSE"
58   6 TEXT
59   7 END
60              (Blanc)
```

Tronçon 4

```
61              RUN
62              LOAD DR.PROG,D1
63              NEW
```

Bout 4

```
64              (Blanc)
65   Ø AA=1
66   1 AA1=2
67   2 D$ = ""                          (Il y a CTRL-D)
68   3 PRINTD$;"APPEND T1"
69   4 PRINTD$;"WRITE T1"
70   5 PRINT AA
71   6 PRINTD$;"CLOSE"
72   7 END
73              (Blanc)
```

Tronçon 5

```
74              RUN
75              FP
```

Bout 5

```
76              (Blanc)
77   Ø D$=CHR$(4):?D$;"OPEN T1":?D$;"READ T1"
78   1 INPUT A$:INPUT A:?D$;"CLOSE"
79   2 IF A=1 THEN ?D$;"EXEC PROC1,R89":END
80              (Blanc)
```

Tronçon 6

```
81              RUN
82              LOAD DR.PROG,D1
```

Bout 6

```
83              (Blanc)
84    Ø D$=CHR$(4):?D$;"APPEND PROV,D2":?D$;"WRITE PROV"
85    1 POKE 33,33: LIST 32768-:?D$;"CLOSE":TEXT
86    2 ?D$;"EXEC PROC1,D1,R92":END
87              (Blanc)
```

Tronçon 7

```
88              RUN
89              NEW
9Ø              Ø D$ = CHR$(4): A$ ="I"
91              EXEC PROC1,R95
92              NEW
```

Bout 7

```
93              (Blanc)
94    Ø D$=CHR$(4): A$="A"
95    1 ?D$;"OPEN T1":?D$;"POSITION T1,R1":?D$;"WRITE T1"
96    2 ?A$:?D$;"CLOSE"
97              (Blanc)
```

Tronçon 8

```
98              RUN
```

Bout 8

```
99              (Blanc)
1ØØ   Ø D$=CHR$(4):?D$;"OPEN PROV,D2":?D$;"READ PROV"
1Ø1   1 POKE 216,Ø: ONERR GOTO4
1Ø2   2 INPUT A$: IF LEN(A$)<>Ø GOTO 5
1Ø3   3 GOTO 1
1Ø4   4 POKE 216,Ø:?D$;"CLOSE":?D$;"DELETE PROV":?D$;"DELETE T1,
        D1":?D$;"DELETE DR.PROG":HOME:?"M.C., VIDE A L'APPEL DE
        L'EXEC DE PROC1":?CHR$(7):?D$;"EXEC PROC1,R125":END
1Ø5   5 A3$=STR$(VAL(A$)):B$=A$
1Ø6   6 POKE 216,Ø: ONERR GOTO 1Ø
1Ø7   7 INPUT A$: IF LEN(A$)> <Ø  GOTO 9
1Ø8   8 GOTO 6
1Ø9   9 B$=A$: GOTO 6
11Ø   1Ø POKE 216,Ø:A4$=STR$(VAL(B$)):?D$;"CLOSE":?D$;"OPEN T1,
        D1":?D$;"READ T1":INPUT A2$:INPUT A1$:?D$;"CLOSE"
111   11 A$=A1$+A2$+"("+A3$+"/"+A4$+")"
112   12 ?D$;"OPEN T1":?D$;"DELETE T1"
113   13 ?D$;"OPEN T1":?D$;"WRITE T1"
114   14 ?"RENAME PROV,"+A$+",D2":?"LOCK"+A$:?"TEXT":?"HOME":
        ?"?"+CHR$(34)+"LE NOM DEFINITIF EST:"+CHR$(34): PRINT
        "SPEED=Ø":?"?"+CHR$(34)+A$+CHR$(34):?"SPEED=<55"
115   15 ?"EXEC PROC1,D1,R119":?D$;"CLOSE"
116             (Blanc)
```

Tronçon 9

```
117         RUN
118         EXEC T1,D1
119         ?"OPERATION TERMINEE"+CHR$(7)
120         ?"PROGRAMME TEXTE EN D2"
121         ?"PROGRAMME EN D1 SOUS LE NOM:DR.PROG"
122         ?"ATTENTION: AU PROCHAIN EXEC DE PROC1"
123         ?"DR.PROG SERA DETRUIT"+CHR$(7)
124         DELETE T1,D1
125         NEW
126         CLOSE PROC1
127         NEW
128         ?"M.C., SAUVEE PAR PRECAUTION EN D1"
129         ?"SOUS LE NOM:DR.PROG"
130         DELETE T1,D1
```

III.2 COMMENTAIRES SUR PROC1

- Tronçon1.

En tout premier lieu, est sauvé sur D1 le programme à transformer;un éventuel fichier texte T1 est détruit, et le compilateur Applesoft prend le contrôle.

- Bout1.

Est alors chargé en mémoire, de Ø à 26, un programme qui a pour fonction, en première opération de clore le fichier PROC1 lui-même: ceci est indispensable pour permettre les input dans un programme; si le fichier en cours d'exécution était resté ouvert, un INPUT se serait automatiquement adressé au disk.
Il est alors possible à l'opérateur de communiquer, depuis le clavier le nom qu'il désire affecter au programme traité.
Après avoir vérifié la correction syntaxique du nom choisi par l'opérateur, PROC1 met en réserve ce nom dans T1, et demande à installer en D2 le disk qui recevra le texte.
Lorsque tout le programme est déroulé, la dernière instruction permet de réouvrir PROC1, là où il avait été quitté.

- Tronçon2.Bout 2.

Le programme à transformer est chargé, puis un autre permettant de lister le précédent de Ø à 7. Ce programme est rédigé de telle manière qu'il soit exécutable aussi bien en BASIC entier qu'en Applesoft.

- Tronçon3.Bout3.

Comme Tronçon2 permettait de lancer Bout1, Tronçon3 permettra de lancer Bout2 (RUN8), puis de nouveau le programme à traiter est rechargé, ainsi que celui qui le listera de 8 à 32567, dernier numéro de ligne autorisé en BASIC entier.

- Tronçon4.Bout4.

Après avoir exécuté Bout3, le programme est à nouveau chargé; il s'agit maintenant de déterminer sous quel BASIC il a été construit; le programme formé par Bout 4 répond à la question; il est exécutable sous les deux BASIC, mais interprété différemment par chacun d'eux: AA contiendra 1 si on était en BASIC entier, 2 dans le cas contraire. Cette valeur de AA est rangée dans le fichier T1, pour être consultée ultérieurement.

- Tronçon5.Bout5.

Après avoir lancé Bout4, on repasse en Applesoft, et Bout5 permet de réaliser dans PROC1 un aiguillage, selon la valeur de

AA. Si AA=1, Bout6 ne sera pas exécuté. Dans le cas contraire il l'est.

- Tronçon6.Bout6.

On est certain, en exécutant Bout6, que le programme traité était en Applesoft. On liste donc l'éventuel reliquat de 32768 au maximum autorisé en Applesoft.

- Tronçon7.Bout7.

Comme on l'a indiqué plus haut, ces deux zones du fichier PROC1 seront EXECutées différemment selon le BASIC qui était attaché au programme traité:selon le cas, la première ligne du programme chargé sera celle définié par le champ 9Ø ou 94 du fichier. Dans T1, on trouvera alors soit, A, soit I.

- Tronçon8.Bout8.

Tronçon8 lance l'exécution de ce qui a déjà été chargé, puis s'implante Bout8. La fonction de Bout8 est d'analyser le fichier provisoire qui a été constitué, et de déterminer quelss sont les numéros de lignes extrêmes, pour les adjoindre au nom définitif qui sera attribué au texte final.

- Tronçon9.

Après avoir lancé l'exécution de Bout8, il ne reste plus qu'à indiquer le résultat de l'opération, et à remettre les choses en l'état standard.

En définitive, le programme chargé à l'origine en mémoire centrale, se trouve à la fin sous forme texte en D2, sous un nom ainsi formé: X.....(n° ligne mini/n° ligne maxi)

X = A ou I selon le BASIC, Applesoft ou entier.

..... Nom donné par l'opérateur.

IV REMARQUES

Les quelques indications qui ont été données dans ce qui précède ne décrivent évidemment pas dans le détail le plus fin tout ce qui se passe au moment où l'opérateur lance la procédure en faisant EXEC PROC1,D1.

De nombreux détails n'apparaîtront que si on étudie avec soin, le contenu du fichier PROC1; d'autres ne se révèleront que si on étudie le comportement de la commande EXEC, qui diffère selon qu'on l'utilise en mode immédiat ou à l'intérieur d'un programme.

Il ne reste plus qu'à fermer le fichier, à reformatter en standard la fenêtre, et à arrêter l'exécution.

Il est clair, toutefois, que cette méthode laisse encore à l'opérateur un travail considérable à accomplir:

- A chaque opération, il faut repérer les lignes de programme à lister; ces lignes de programme ne doivent pas se trouver implantées dans la zone où le programme de LISTing se trouve lui-même.
- Il faut modifier manuellement le nom du fichier ouvert, à chaque nouvelle opération.
- Il faut modifier le programme, si le texte qu'on veut constituer est celui d'un programme en BASIC entier.
- En fin, il faut se souvenir, de l'implantation du programme transformé: en effet EXEC ne détruit pas les lignes préexistantes à son appel, sauf si des numéros de lignes, sont communs au texte qu'EXEC charge et à celui déjà chargé. Ceci est particulièrement contraignant, puisqu'il n'y a plus aucun moyen, une fois le texte constitué, de connaître les numéros de lignes, autrement qu'en chargeant le programme, c'est à dire en risquant de détruire ce qui est déjà implanté en mémoire.
- Il n'y a aucun moyen de savoir sous quel BASIC, le texte a été constitué.

Nous avons donc construit une procédure, entièrement automatique, donc sans aucune intervention de l'opérateur, qui permet de résoudre l'ensemble des questions évoquées ci-dessus.

Remarque: il existe, fournis dans les Master Disk, des programmes, qui apportent des solutions partielles aux problèmes étudiés ici; celles-ci sont très incomplètes; signalons par exemple qu'elles ne fonctionnent pas en BASIC entier, et même qu'elles réduisent la zone programme utilisable. Dans leur principe, elles sont très différentes de ce qui est exposé ici.

Enfin, si un lecteur désire constituer pour son utilisation personnelle, le fichier PROC1, il risque d'être étonné de voir figurer certains champs vides dans la description du contenu de ce fichier.

Nous suggérons d'appliquer "manuellement", la procédure de base telle qu'elle a été définie au tout début de ces notes, aux différents morceaux de programme, que nous avons appelés bout1,bout2,....,bout8.

Les tronçons pourront être rangés dans des chaînes de caractères , puis écrits dans PROC1 au moyen de très simples programmes, utilisant APPEND à la place de OPEN(sauf pour le premier tronçon).

L'utilisateur comprendra alors la raison d'être des blancs que nous avons signalés: la fonction LIST,a pour effet, dans les conditions ici décrites, avant d'imprimer le texte du programme qui lui est soumis, d'expédier d'abord une ligne blanche, puis, après la dernière ligne de programme, à nouveau une autre ligne blanche. Ces deux lignes, à chaque intervention de LIST, vont se constituer telles quelles dans le fichier PROC1, en définissant des champs vides, mais dont le décompte doit être tenu, lorsqu'on désire se pointer précisément dans le fichier, ce qui est ici, le cas.

Le comportement de LIST est d'ailleurs, en réalité plus complexe encore, mais ici, ceci n'a aucune influence.

VOUZZAVEDIBISAR.

Citations.

LES PRINCIPES DU NOVLANGUE

Le novlangue a été la langue officielle de l'Océania. Il fut inventé pour répondre aux besoins de l'Angsoc, ou socialisme anglais.

De plus, et ceci s'appliquait encore en principe à tous les mots de la langue, n'importe quel mot pouvait prendre la forme négative par l'addition du préfixe *in*. On pouvait en renforcer le sens par l'addition du préfixe *plus*, ou, pour accentuer davantage, du préfixe *doubleplus*. Ainsi incolore signifie pâle, tandis que *pluscolore* et *doublepluscolore* signifient respectivement « très coloré » et « superlativement coloré ».

Il était aussi possible de modifier le sens de presque tous les mots par des préfixes-prépositions tels que *anté, post, haut, bas*, etc.

Grâce à de telles méthodes, on obtint une considérable diminution du vocabulaire. Etant donné par exemple le mot *bon*, on n'a pas besoin du mot *mauvais*, puisque le sens désiré est également, et, en vérité, mieux exprimé par *inbon*. Il fallait simplement, dans les cas où deux mots formaient une paire naturelle d'antonymes, décider lequel on devait supprimer. *Sombre*, par exemple, pouvait être remplacé par *inclair*, ou *clair* par *insombre*, selon la préférence.

La seconde particularité de la grammaire novlangue était sa régularité. Toutes les désinences, sauf quelques exceptions mentionnées plus loin, obéissaient aux mêmes règles.

— — — —

Vocabulaire B. — Le vocabulaire B comprenait des mots formés pour des fins politiques, c'est-à-dire des mots qui, non seulement, dans tous les cas, avaient une signification politique, mais étaient destinés à imposer l'attitude mentale voulue à la personne qui les employait.

Il était difficile, sans une compréhension complète des principes de l'angsoc, d'employer ces mots correctement. On pouvait, dans certains cas, les traduire en ancilangue, ou même par des mots puisés dans le vocabulaire A, mais cette traduction exigeait en général une longue périphrase et impliquait toujours la perte de certaines harmonies.

Les mots B formaient une sorte de sténographie verbale qui entassait en quelques syllabes des séries complètes d'idées, et ils étaient plus justes et plus forts que ceux du langage ordinaire.

Les mots B étaient toujours des mots composés.

exemple, le mot « bonpensé » signifiait approximativement « orthodoxe » ou, si on voulait le considérer comme un verbe « penser d'une manière orthodoxe ». Il changeait de désinence comme suit : nom-verbe *bonpenré*, passé et participe passé *bienpenré*; participe présent : *bonpensent*; adjectif : *bonpensable*; nom verbal : *bompenseur*.

Les mots B n'étaient pas formés suivant un plan

article de fond du *Times : Ancipenseur nesentventre Angsoc*. La traduction la plus courte que l'on puisse donner de cette phrase en ancilangue est : « Ceux dont les idées furent formées avant la Révolution ne peuvent avoir une compréhension pleinement sentie des principes du Socialisme anglais. »

— — — — —

Mais la fonction spéciale de certains mots novlangue comme *ancipensée*, n'était pas tellement d'exprimer des idées que d'en détruire. On avait étendu le sens de ces mots, nécessairement peu nombreux, jusqu'à ce qu'ils embrassent des séries entières de mots qui, leur sens étant suffisamment rendu par un seul terme compréhensible, pouvaient alors être effacés et oubliés. La plus grande difficulté à laquelle eurent à faire face les compilateurs du dictionnaire novlangue, ne fut pas d'inventer des mots nouveaux mais, les ayant inventés, de bien s'assurer de leur sens, c'est-à-dire de chercher quelles séries de mots ils supprimaient par leur existence.

— — — — —

Crimesex concernait les écarts sexuels de toutes sortes. Ce mot englobait la fornication, l'adultère, l'homosexualité et autres perversions et, de plus, la sexualité normale pratiquée pour elle-même. Il n'était pas nécessaire de les énumérer séparément puisqu'ils étaient tous également coupables. Dans le vocabulaire C, qui comprenait les mots techniques et scientifiques, il aurait pu être nécessaire de donner des noms spéciaux à certaines aberrations sexuelles, mais le citoyen ordinaire n'en avait pas besoin. Il savait ce que signifiait *biensex*, c'est-à-dire les rapports normaux entre l'homme et la femme, dans le seul but d'avoir des enfants, et sans plaisir physique de la part de la femme. Tout autre rapport était *crimesex*. Il était rarement possible en novlangue de suivre une pensée non orthodoxe plus loin que la perception qu'elle était non orthodoxe. Au-delà de ce point, les mots n'existaient pas.

Il n'y avait pas de mot, dans le vocabulaire B, qui fût idéologiquement neutre. Un grand nombre d'entre eux étaient des euphémismes. Des mots comme, par exemple : *joiecamp* (camp de travaux forcés) ou *minipax* (ministère de la Paix, c'est-à-dire ministère de la Guerre) signifiaient exactement le contraire de ce qu'ils paraissaient vouloir dire.

D'autre part, quelques mots révélaient une franche et méprisante compréhension de la nature réelle de la société océanienne. Par exemple *prolealiment* qui désignait les spectacles stupides et les nouvelles falsifiées que le Parti délivrait aux masses.

— — — — —

Les mots du vocabulaire B gagnaient même en force, du fait qu'ils étaient presque tous semblables. Presque invariablement, ces mots — *bienpensant, minipax, prolealim, crimesex, joiecamp, angsoc, ventresent, penséepol*... — étaient des mots de deux ou trois syllabes dont l'accentuation était également répartie de la première à la dernière syllabe. Leur emploi entraînait une élocution volubile, à la fois martelée et monotone. Et c'était exactement à quoi l'on visait. Le but était de rendre l'élocution autant que possible indépendante de la conscience, spécialement l'élocution traitant de sujets qui ne seraient pas idéologiquement neutres.

Pour la vie de tous les jours, il était évidemment nécessaire, du moins quelquefois, de réfléchir avant de parler. Mais un membre du Parti appelé à émettre un jugement politique ou éthique devait être capable de répandre des opinions correctes aussi automatiquement qu'une mitrailleuse sème des balles. Son éducation lui en donnait l'aptitude, le langage lui fournissait un instrument grâce auquel il était presque impossible de se tromper, et la texture des mots, avec leur son rauque et une certaine laideur volontaire, en accord avec l'esprit de l'angsoc, aidait encore davantage à cet automatisme.

— — — — —

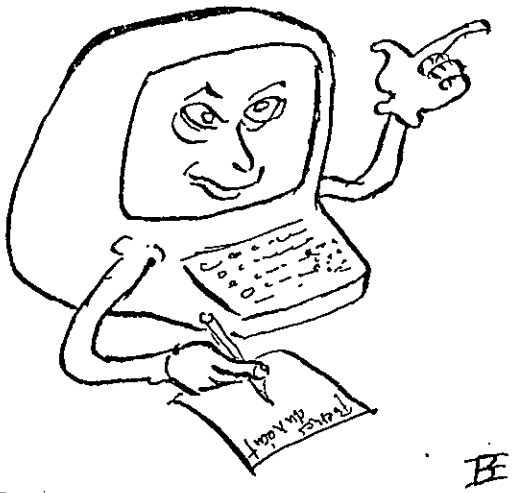
Enfin, on espérait faire sortir du larynx le langage articulé sans mettre d'aucune façon en jeu les centres plus élevés du cerveau. Ce but était franchement admis dans le mot novlangue : *canelangue*, qui signifie « faire coin-coin comme un canard ». Le mot *canelangue*, comme d'autres mots divers du vocabulaire B, avait un double sens. Pourvu que les opinions émises en *canelangue* fussent orthodoxes, il ne contenait qu'un compliment, et lorsque le *Times* parlait d'un membre du Parti comme d'un *doubleplusbon canelangue*, il lui adressait un compliment chaleureux qui avait son poids.

Un seul mot est souligné -par nous- mais il montre combien était grand le génie de l'auteur qui écrivit cette oeuvre d'actualité en 1948. Aussi nous permettons-nous de nous adresser à nos lecteurs cultivés, en leur posant des questions assorties de récompenses.

Question 1 : Les personnes érudites qui auront découvert à la lecture de ces quelques citations, de quelle oeuvre est tiré cet extrait, gagnent le droit de se regarder dans leur glace avec un beau sourire.

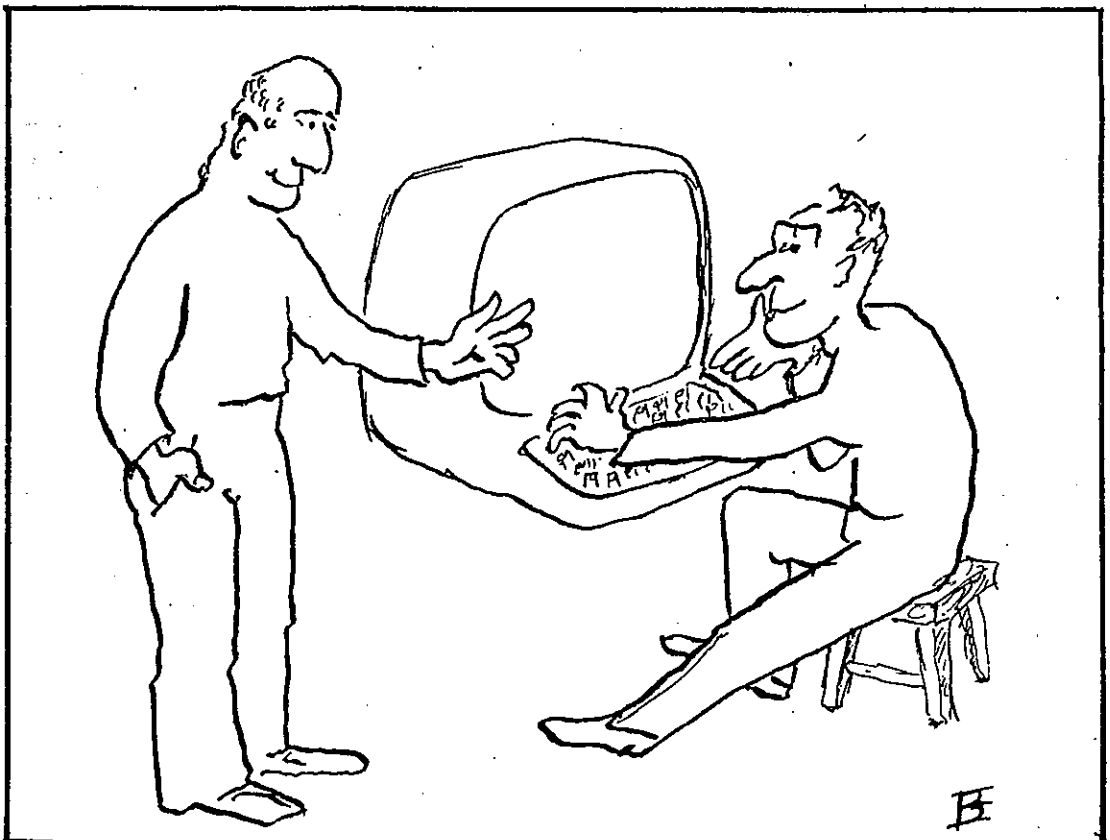
Question 2 : Pourrait-on imaginer un rapport quelconque entre le Novlangue et une langue parlée et écrite couramment de nos jours ? Notre laboratoire est ouvert à tout Novpenseur, chercheur dans ce sens.

Tendances...



-"Quand fuit le vent d'hiver et que fleurit mon coeur au souffle du printemps...

-"Angsocbut : Defsoc. Ancimot 'informatique' insens car defsocpenseur sentventre novlangue doubleplusbon...



-C'est du basic ?

-Non c'est du Novlangue.