ISSN 0291 - 5413

Publication du Département de Mathématique & d'Informatique

BULLETIN d'INFORMATIQUE approfondie & applications

J.-M. KNIPPEL

Nº 1

Laboratoire d'Informatique Théorique

ISSN 0291 - 5413

Informatique
fondamentale
et
Applications

Sommaire

- Editorial ... P.1

- Le Processeur Expérimental MCA-0 ... P.6

- Automate programmable Interpréteur de Réseaux de Petri ... P.12

- Une nouvelle machine...! ... P.23

J.-M. KNIPPEL

Juin 1981

Faculté des Sciences de Luminy
Département de Mathématique-Informatique
70, route Léon Lachamp - Case 901
13288 MARSEILLE CEDEX 9

Editorial

E. Bianco

Qu'est-ce que l'informatique ?

Comment peut-on poser une question pareille en 1981 ? D'abord, il me paraîtrait grave que l'on ne puisse plus se poser ce genre de question, même ce jour qui me semble encore bien lointain où l'Informatique, vieille dame au chignon bien arrangé, ne tentera plus guère de jeunes et fougueux amants. Et de solides domaines bien protégés par d'immenses savoirs encyclopédiques auraient à mon sens besoin d'être un peu soumis de temps en temps à un petit test de validité existentielle. Mais ne rêvons pas trop. Profitons encore un peu de ce que l'Administration ne sait pas encore bien ce qu'est l'informatique. Ce qui ne l'empêche pas sans haine et sans passion de prendre quelques décisions. Ne parle-t-on pas déjà d'introduire cette discipline dans le secondaire. Mais de l'introduire, comme on viole, en force et sans rien dire. A-t-on créé un poste d'informaticien ? où postuleraient des gens éventuellement intéressés et qualifiés ? Il semblerait bien que la charge échoirait aux professeurs de mathématiques. Pourquoi pas après tout ? Mais alors, ceux à qui cela ne plaît pas, ou qui ne sont pas au courant ? Et pourquoi le professeur de lettres ou d'éducation physique ne serait-il pas tout désigné dans certains cas. Mais je crois être là encore en train de soulever le problème des a priori.

Peut-on essayer de faire une sorte de bilan rapide qui donnerait une idée de la situation informatique. Il me semble d'abord que pour y voir un peu clair, il est bon de distinguer marché du micro-processeur ou de l'ordinateur et informatique. S'il est parfaitement vrai qu'il y a eu révolution sur le marché du matériel, qui a mis à la portée d'à peu près toutes les bourses un outil puissant et qui pourrait être bien adapté, il n'en est pas du tout de même au rayon informatique.

Le soft, comme on dit, le logiciel, n'a pas suivi. Contrairement à ce que l'on pourrait penser. Le logiciel en est toujours à l'âge de la pierre. Et encore ...

J'apprenais récemment que l'on utilise couramment des compilateurs que l'on réintroduit en mémoire pour chaque programme sur lequel on l'applique. Trois programmes, trois fois le même compilateur. Dans les laboratoires d'informatique on sait faire des compilateurs bien plus perfectionnés. Alors pourquoi ce hiatus.

Dans le domaine du contrôle des processus j'ai pu constater également que des armoires électroniques, à base de micro-processeur, étaient commercialisées et catégorisées selon le type précis de travail, ignorant donc la notion de procédure. Celle-ci n'est pourtant pas une notion bien neuve.

On est alors fondé à se demander pourquoi un tel décalage entre des résultats de recherche bien connus et l'informatique in vivo.

En fait, cela n'est pas aussi surprenant qu'il pourrait y paraître. Regardons d'un peu près. Quand les premiers ordinateurs sont arrivés sur le marché, en France, rapidement un tout petit nombre d'Universités s'est équipé. Ces ordinateurs étaient tout entiers utilisés à la recherche et à l'utilisation scientifique. Mais, très rapidement, la recherche fondamentale en informatique est passée au second plan. Car un ordinateur cela coûte cher à l'achat et encore plus cher à l'utilisation. Et le directeur du centre de calcul, placé devant ses échéances, était bien content de trouver acheteurs de calcul chez ses collègues scientifiques, voire dans le privé. Le client qui paye sent monter au fond de lui des exigences qui ne tardent pas à se manifester d'une manière ou d'une autre. D'autant qu'un ordinateur n'est pas un instrument à l'humeur éternellement sereine. Les pannes cela existe et les pannes de longue durée cela existe aussi, qui vous bouleversent tout un planning. Une telle sorte de centre de calcul n'est pas une entreprise destinée à faire du profit, les prix pratiqués sont minima mais pour un utilisateur à petit budget c'est encore bien cher et tout retard est ressenti comme un grave manque à gagner, voire une offense.

Et c'est toujours au moment où le retard s'est bien accumulé que cette sacrée machine tombe en panne. Quelle place reste-t-il au milieu de tout cela au chercheur qui essaye de faire de l'informatique fondamentale, à peu près la même que celle qui existe entre le marteau et l'enclume, pour ne pas utiliser une métaphore plus directe. Voilà à peu près la situation vers la fin des années cinquante, début des années soixante. Elle n'allait pas tarder à se compliquer.

Que l'on fasse les choses à moitié n'est pas nouveau, que l'on donne l'outil sans donner les moyens de le faire fonctionner est une situation somme toute courante. En informatique, cette petite économie de quelques millions est en train de coûter une quantité invraisemblable de milliards. Un gâchis fantastique, de quoi rêver.

La situation se complique car un troisième larron intervient, et non des moindres, aidé naïvement par les scientifiques eux-mêmes. L'outil se révèle si souple et efficace malgré ses défauts, qu'il apparait rapidement naturel de l'appliquer aux tâches imbéciles et indispensables de l'administration. Les résistances normales vaincues, non sans difficulté d'ailleurs, le tournant était pris.

Le mécontentement des clients, allié au fait que les ordinateurs coûtaient de plus en plus cher à l'achat et au fonctionnement, amena rapidement à la conclusion évidente qu'un tel trésor ne pouvait être abandonné entre des mains de spécialistes, l'informatique était visiblement chose trop sérieuse. C'est tout naturellement l'administration qui prit le relai. Et d'ailleurs, où voulez-vous trouver des crédits ?

Mais, "et la recherche fondamentale en informatique" demanderez-vous, que voulez-vous qu'on en fasse, mais de la théorie voyons. A mon sens, en dehors de l'Université, aucune entreprise n'a les moyens de faire vraiment de la recherche fondamentale dans ce domaine mais l'Université ne s'est pas donné non plus les moyens.

Je pense tout simplement que l'on ne s'est pas très bien rendu compte de l'importance d'une étude fondamentale sur le logiciel.

L'idée s'est répandue qu'il est relativement facile de devenir programmeur et c'est vrai en un certain sens. Un grand informaticien ne déclarait-il pas dès les années soixante : "nous savons écrire des compilateurs, ce chapitre est désormais clos". Et on continue à écrire des compilateurs comme en soixante. Cette déclaration n'avait rien de scientifique, mais c'était une prophétie. Le berger n'est-il pas prophète pour ses moutons ?

Qui fait effectivement du logiciel ? Il y a à mon sens deux sortes de producteurs importants : les constructeurs d'ordinateurs qui livrent du logiciel de base et les gros utilisateurs, qui enrichissent les programmathèques de produits les plus divers.

En général, pour sortir rapidement un nouveau modèle, on reprend un vieux logiciel et on rapièce. Ou alors on fait du neuf mais on le "protège" des indiscrétions en le compliquant pour le rendre impénétrable pendant un délai que l'on estime suffisant pour sortir le modèle suivant. Le secret industriel y gagne, mais pas l'efficacité du logiciel.

L'utilisateur, lui, construit des programmes parce qu'il en a besoin. Il se sert de langages de programmation que l'on dit évolués mais dont certains commencent à dater un peu. Dès l'instant où le programme donne ce qu'on en attend, pourquoi se préoccuper de sa structure ; c'est en fait, là, une préoccupation que l'on ne saurait imputer à un simple utilisateur. Cela me semble précisément faire partie de la recherche fondamentale. Cependant, l'intérêt du programme à partir du moment où il existe en programmathèque est qu'il peut être réutilisé, et pour cela être inséré dans un réseau d'autres programmes, eux-mêmes provenant d'horizons très différents. Et c'est là que commencent les difficultés.

Voilà pour la grosse informatique.

Mais la micro-informatique n'était-elle pas à son arrivée l'occasion d'un bon dépoussiérage de tout cela ? On aurait pu le souhaiter, beaucoup l'ont souhaité. Mais hélas économie oblige et le logiciel coûte aussi cher sur un petit que sur un gros système. Il y a des vérités qui mettent longtemps à sortir du puits. Et on écrit sur les micros comme on écrivait dans les années cinquante, le perfectionnement apparaît sous la forme, cautère sur jambe de bois, de ces ineffables O.S. (operating systems) célèbres dans les années soixante.

Le seul apport moderne étant cette recherche maladive de la protection et du secret par la complication, justifié peut-être par le foisonnement des micro-constructeurs, mais qui s'applique précisément à un domaine impossible à breveter. Et je vois très bien les logiciellistes-micro dans la situation d'Escartefigue à qui César déclarait : "Je te vois tenant les bornes du couillonisme entre tes bras et courant à toute vitesse pour agrandir ton domaine...".

Logiciel anonyme ou à responsabilité limitée, logiciel pétassou, logiciel désuet, voici une image de l'informatique "moderne". Bien sûr ce que je dis là est très schématique, et bien souvent la conséquence d'une hyper-croissance ultra-rapide, n'ayons pas peur des hyperlatifs. Il s'agit d'un point de vue sur une situation dont le bouillonnement montre la richesse, et qui correspond à l'expérience que j'ai vécue de la naissance de l'informatique. Et ce point de vue est forcément partiel voire partial. Je conçois parfaitement que d'autres personnes aient pu conserver une autre impression, différente, voire contradictoire, de cette évolution. Nos colonnes sont ouvertes à qui voudrait raconter. J'essayerai dans un numéro ultérieur de parler un peu de l'informatique américaine, pour autant qu'il en existe une européenne.

N.B. Quand je parle d'administration, des esprits un peu superficiels pourraient penser que je dirige mes foudres vers une classe particulière d'employés. Il n'en est rien ; d'abord je n'ai aucune foudre à diriger contre personne, et ensuite par administration je désigne un état d'esprit, une attitude. Et j'ai pu souvent constater que l'emploi ne fait pas forcément la mentalité, ni la mentalité, l'emploi.

Le Processeur Expérimental

MCA-0

(suite)

E. Bianco

LA MEMOIRE CENTRALE

Je vais essayer à titre d'exemple de montrer comment s'effectue la commande de la mémoire centrale à partir du processeur.
La mémoire lui est connectée par deux bus : le data-bus et un bus de lignes de commandes. Le data-bus est à double sens, il sert à véhiculer l'information, soit vers la mémoire, soit à partir de la mémoire.
Les lignes de commandes dirigent l'information du processeur vers la mémoire afin de déterminer l'état de cette dernière : soit hors du circuit, soit en lecture, soit enfin en écriture. Trois états, il faut donc deux lignes. Ce sont : C_m et M_w .
Le tableau qui suit donne l'état logique :

Etat mémoire	C_m	M_w
Coupée	1	3e état
W en mém.	0	0
R en mém.		1

Cet ensemble d'informations est repris par les circuits de contrôle de la mémoire. Les choses se compliquent un peu du fait que celle-ci doit également être attaquée par les circuits d'échange. Ici l'exemple traité est relativement simple dans la mesure où la sortie se fait sur des néons, et l'entrée se fait au moyen d'un double jeu d'interrupteurs. Les néons visualisent en permanence l'information qui se trouve sur les bus.

Les interrupteurs, eux, n'interviennent que dans l'état "manuel". Mais en cet état, l'adresse est imposée par les clefs adresse, ce qui permet, soit de changer, soit de visualiser le contenu de chaque case de la mémoire. Il faut superposer à cela la nécessité, lorsque l'on est en écriture venant du processeur, de ne laisser la mémoire "ouverte" que pendant une durée restreinte par rapport à celle pendant laquelle est stabilisée l'information sur le data-bus. D'où intervention de Φ_r , et un fil de plus au bus commandes.

Le circuit le plus compliqué est celui qui va devoir commander les buffers X et Y qui contrôlent l'accès à la mémoire. Je vais construire d'abord le tableau des exigences, qui définit, en fonction des états, les niveaux de commande des buffers X et Y .

man. aut.	ecr. lec.	M'_w	X, WR	Y
0 (a)	x	0	0	1
	x	1	1	0
1 (m)	0 (w)	x	1	1
	1 (r)	x	0	0

a → automatique
m → manuel
w → écriture
r → lecture

La valeur "x" signifie que la valeur correspondante est sans importance sur l'état de la machine, dans la situation décrite sur la ligne. Je mets en correspondance avec ce tableau le tableau suivant.

man. aut.	ecr. lec.	M'_w	X, WR	Y
0	0	0	0	1
		1	1	0
0	1	0	0	1
		1	1	0
1	0	0	0	1
		1	0	1
1	1	0	1	0
		1	1	0

C'est le circuit "F" qui réalise la fonction logique décrite dans ce dernier tableau.

Le signal M'_w est une fonction de M_w et de l'horloge Φ_r. Comme on le constate sur le schéma.

FONCTIONNEMENT ET DEPANNAGE

Ce MCA-0, instrument expérimental, a été conçu pour un maximum de souplesse dans son fonctionnement. C'est ainsi qu'il était destiné à pouvoir "tourner" à grande vitesse, aux alentours de 1 MHZ et aussi à très basse vitesse, 1 HZ ou une fraction de herz.
Cette souplesse présente des avantages évidents aussi bien en enseignement que dans la nécessité d'un dépannage. Un jeu de visualisateurs permet de contrôler à la fois le numéro de ligne du programme interne, le contenu de la mémoire mA et le code de l'instruction.

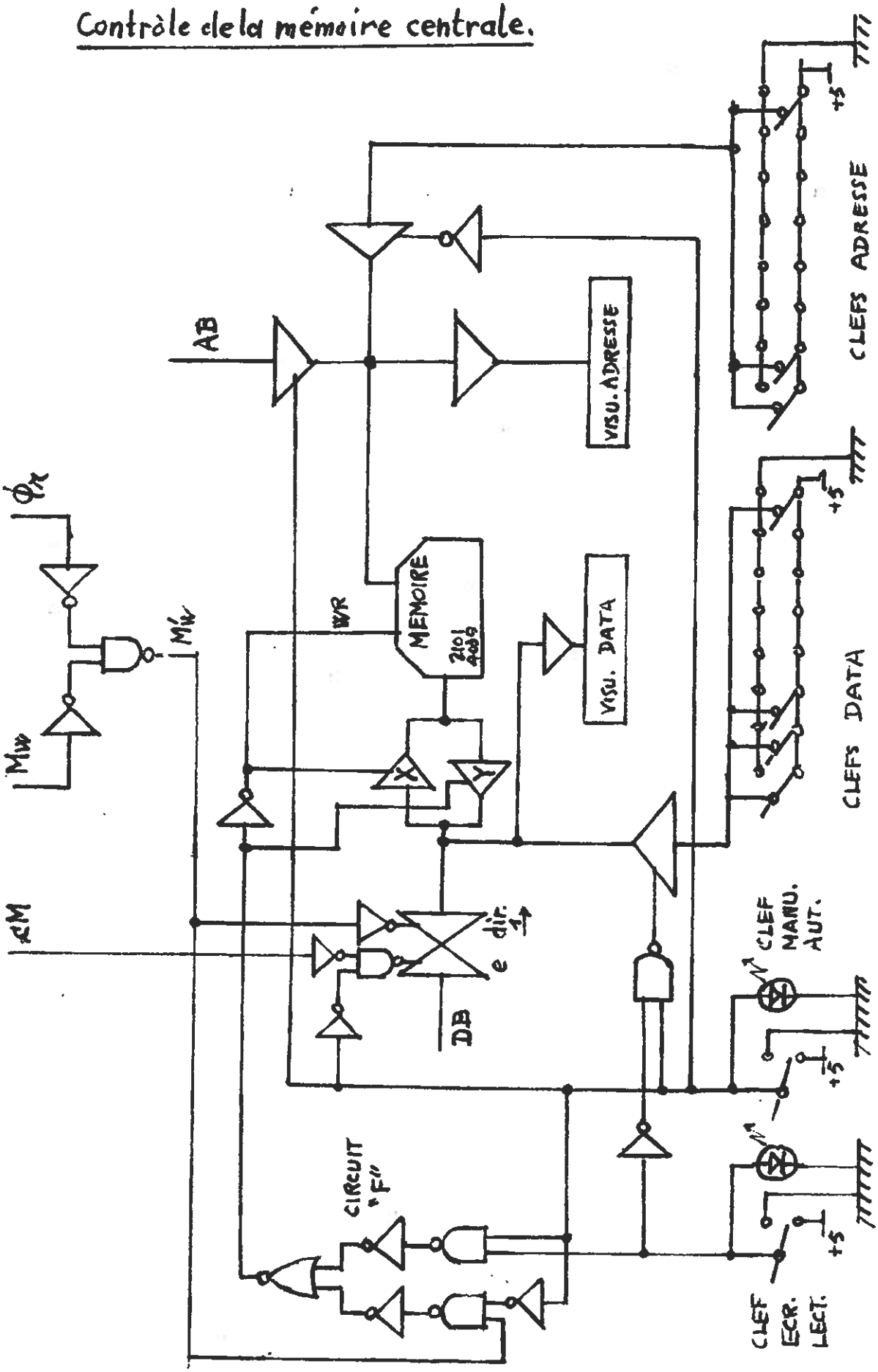
Il suffirait de peu de choses pour visualiser également les contenus des autres registres : PC, m2 et m3 . Cela donnerait à chaque instant la vision complète de l'état du processeur, montrant l'évolution de la situation à chaque pas du programme interne. Un interrupteur électronique permet à volonté de masquer l'oscillateur-pilote de l'horloge, ce qui fige le processeur sur un état stable de calcul, ou de libérer l'oscillateur, ce qui relance le calcul.

J'ai également prévu un dispositif qui permette le pas à pas pour le déroulement d'un programme LUP. Seul le temps a manqué pour sa mise en place. Toutefois, le programme interne est déjà structuré pour recevoir ce dispositif. Il suffit de rajouter en tête de ce programme interne une instruction de test sur une clef, dont la position décide du blocage ou du déblocage.

Un processeur statique.

Comme on peut le constater, ce processeur est statique, c'est à dire qu'en dehors des transitions où les échanges entre registres se produisent, le processeur peut rester bloqué indéfiniment, son état se conserve. C'est cette propriété qui le rend utilisable dans une large gamme de fréquences. Les bornes de cette gamme sont limitées différemment vers le haut ou vers le bas. Vers le haut, c'est le temps minimal que demande la transition la plus longue qui est pris en compte. Cette durée se mesure ainsi : à partir du moment où un signal est envoyé sur un bus, on décide de stabiliser une commande d'ouverture sur une mémoire réceptrice connectée à l'autre bout du bus. On combine la durée nécessaire à la prise en compte de la commande et la durée nécessaire pour que l'information qui pénètre alors soit également prise en compte, et c'est une majoration suffisante de la valeur obtenue qui donne la période minimale. Or, toutes ces données dépendent de la technologie utilisée pour l'ordre de grandeur et du type du circuit pour la valeur effective. C'est ainsi qu'un latch servant de registre, en TTL normale, réagit en moins de 20 nanosecondes environ. Alors qu'un buffer réagit en 10 nanosecondes environ.

Contrôle de la mémoire centrale.



Les circuits de ce processeur qui réalisent les transitions ne dépassent pas 100 nanosecondes de temps de réaction. Mais le processeur est forcément ralenti dans la mesure où j'ai dû utiliser des mémoires MOS pour le circuit pilote et pour la mémoire centrale. Pour le circuit pilote j'ai l'intention, à partir du moment où le processeur sera à sa phase d'équilibre, de remplacer les EPROM MOS par des PROM bipolaires dont le temps de réponse est de l'ordre de 40 nanosecondes au lieu de 1000 nanosecondes pour le MOS. Il demeure que la mémoire centrale est en général à conserver en MOS pour des questions de prix et de possibilités d'intégration. Mais à ce moment là on peut modifier la structure du processeur, en utilisant par exemple une horloge programmable dont la fréquence est différente selon qu'elle contrôle une opération interne ou un échange avec la mémoire centrale. D'autres solutions sont également possibles en modifiant la structure du processeur de façon qu'il ait un fonctionnement interne indépendant et des phases d'accès à l'extérieur, c'est à dire à la mémoire centrale. Une petite mémoire intermédiaire rapide pouvant être alors prévue, pour accroître le rendement.

à suivre.

Automate Programmable

C.R. : Subject classifications informatics 1.52, 3.82, 4.13, 6.35, 8.1

Résumé : L'auteur rappelle brièvement les propriétés des réseaux de Petri et du Grafcet. Il explicite les critères d'élaboration d'un automate programmable par définition fonctionnelle et décrit ensuite la réalisation logicielle et matérielle. Réalisé à l'aide d'une carte microprocesseur TM 990/189 (Texas Instrument)l'automate permet, à partir d'un langage efficient, la conduite de processus et le traitement des réseaux de Petri généralisés et à fonctionnement parallèle.

Automate Programmable

Interpreteur de Reseaux de Petri

J.C. FUMANAL

La conception des systèmes logiques fait bien souvent apparaitre des difficultés de compréhension et d'élaboration. L'utilisation de méthodes fondées sur l'algèbre de boole n'a pas apporté de solution vraiment satisfaisante, notamment à cause de leur manque de souplesse. La réalisation de systèmes programmés basée sur ces méthodes n'a fait que concrétiser leur rigidité.

L'automate décrit dans cet article utilise les propriétés des réseaux de Petri (RDP). Il ne nécessite de ce fait qu'une définition fonctionnelle de l'automatisme et permet le traitement de RDP généralisés et à fonctionnement parallèle.

Elaboré initialement pour l'enseignement, il se caractérise par un support de dialogue et une réalisation matérielle qui en fait un outil puissant et simple à mettre en oeuvre.

Cet article permet de dégager les règles qui sont à la base de la réalisation de l'automate. Les concepts introduits à cette occasion ne sont développés que pour les implications qu'ils entrainent, ayant fait par ailleurs l'objet d'une abondante littérature.

I. ELABORATION DES SYSTEMES LOGIQUES PAR LES METHODES CLASSIQUES.

Selon leur mode de fonctionnement, on distingue les circuits logiques combinatoires et séquentiels.

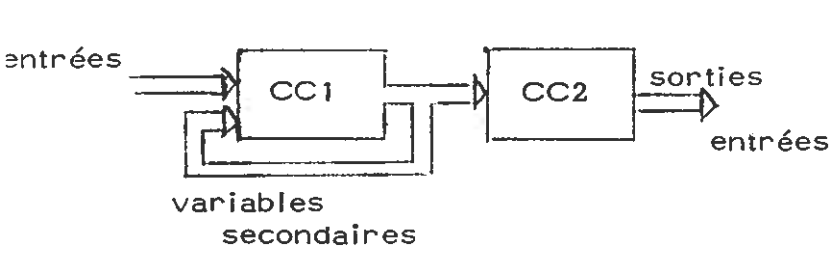
Dans les circuits combinatoires, l'état des sorties ne dépend que de la configuration des entrées. Des méthodes très rigoureuses, basées sur l'algèbre de boole, permettent d'en réaliser la synthèse optimale.

Par contre, pour les circuits séquentiels intervient une donnée que les méthodes précédentes peuvent difficilement intégrer : c'est le "passé" du processus. Par exemple, un compteur d'impulsion, lors de la détection d'une impulsion sur son entrée, doit considérer le nombre d'impulsions déjà passé pour actualiser son affichage.

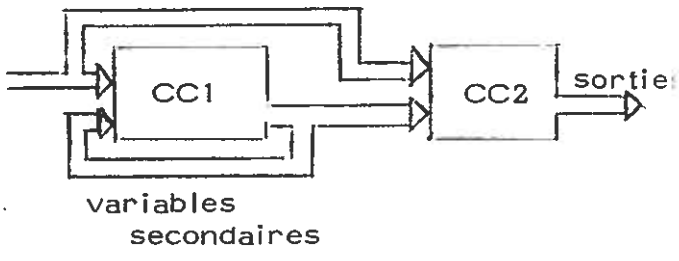
L'idée fondamentale, pour tenir compte de ce facteur temps, a été d'introduire des variables secondaires dont les durées d'élaboration et de propagation permettent à un instant donné d'avoir les informations sur l'état présent et de définir l'état futur.

La synthèse se ramène alors au calcul de fonctions combinatoires dont les sorties dépendent des entrées et de ces variables secondaires représentant l'état présent du système. Dans les sorties figurent également les variables secondaires représentant l'état suivant après leur élaboration.

Une machine séquentielle aura l'un des aspects suivants :



Machine de MOORE



Machine de MEALY

On remarquera que la machine de Mealy tient compte des entrées pour élaborer les sorties sur l'état présent.

Les difficultés d'application des méthodes bâties sur ces modèles sont inhérentes à la synthèse des circuits combinatoires qui nécessite la détermination rigoureuse de toutes les variables, même quand elles ne sont pas significatives. En effet, pour un système industriel dont le nombre d'états est en général important, la définition des variables secondaires amène la construction de diagrammes binaires (table de fluence, table d'adresse ...) dont les dimensions deviennent rapidement prohibitives et par là inexploitables.

Ces méthodes n'ont pas, ou très peu, pénétré le monde industriel, l'intuition et l'expérience du concepteur étant bien souvent suffisantes dans les cas où elles pourraient se révéler efficaces. Il y a également à ce niveau de nombreux facteurs d'incompréhension et de contraintes qui n'ont pas joué en leur faveur.

II. LA DEFINITION FONCTIONNELLE D'UN PROCESSUS - LES SYSTEMES PROGRAMMES.

Face à ces difficultés, le concepteur réagit généralement en effectuant une approche fonctionnelle. Pour celà il ne fait intervenir dans un état donné que les événements susceptibles d'influencer le processus. C'est la notion de receptivité.

Le concepteur et l'utilisateur peuvent alors travailler sur des représentations qui ne comportent que l'essentiel du dispositif. Malheureusement, si ces pratiques permettent une analyse beaucoup plus aisée, il n'en demeure pas moins que la synthèse matérielle du modèle définitif fait ressurgir les difficultés précédentes. L'utilisation de bascules comme élément temporel permet toutefois d'en atténuer l'ampleur.

Avec la pénétration des microprocesseurs et de leur environnement, la gestion d'un schéma fonctionnel a pu être technologiquement réalisable et ceci de la façon suivante :

La mémoire contenant les informations fonctionnelles, à partir d'un programme spécifique, le calculateur traite uniquement l'évolution sur l'état présent et génère les sorties correspondantes. Ce sont les automates programmables par définitions fonctionnelles, par opposition aux automates programmables par équations logiques (ou schémas à relais).

III. LES RESEAUX DE PETRI.

Les réseaux de Petri, définis par A. HOLT (66) à partir des travaux de C.A. PETRI (62), ont bien souvent servi de modèles de base dans la recherche d'un outil de modélisation et d'analyse des systèmes, notamment ceux à évolution parallèle.

Un réseau de Petri est un graphe orienté composé des éléments ci-après :

- les places, symbolisées par des cercles,
- les transitions, symbolisées par des traits,
- les arcs orientés reliant une place à une transition ou inversement,
- les marqueurs, symbolisés par des points.

Les places et les transitions constituent les noeuds du réseau (points de convergence ou de divergence des arcs).

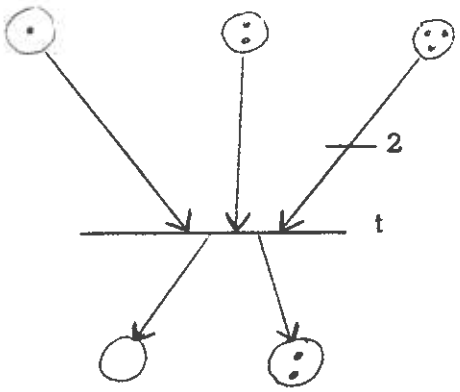
La représentation d'un automatisme se fera en associant :

- aux places, les sorties agissant sur l'extérieur,
- aux transitions, les événements extérieurs agissant sur l'automatisme,
- aux arcs, les chemins selon lesquels se fait l'évolution.

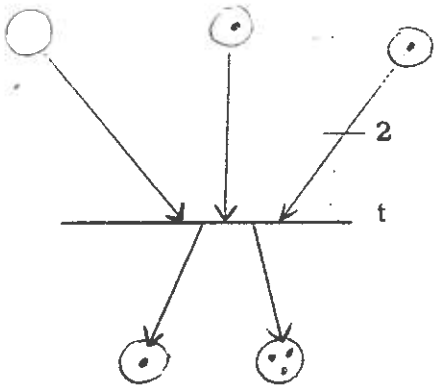
L'état du système est représenté par des "marqueurs" situés à l'intérieur des places. L'ensemble des places marquées traduit l'état de l'automatisme.

La règle d'évolution des marqueurs est la suivante :

Lorsqu'une transition est franchie, on enlève une marque à chaque place située en amont et on ajoute une marque à chaque place située en aval de la transition.



avant



après

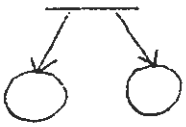
franchissement d'une transition

Pour qu'une transition soit franchissable il est nécessaire, l'événement lié à la transition étant supposé réalisé, qu'il y ait dans chaque place amont au moins le nombre de marques qui sera prélevé lors du franchissement.

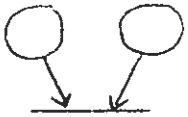
Les quatre configurations suivantes sont alors suffisantes pour la représentation d'un automatisme.

convergence (ou)

divergence (ou)



divergence (ET)



convergence (ET

IV. LE GRAFCET.

Ce modèle a été défini en vue de normaliser la représentation du cahier des charges des automatismes. C'est une version des réseaux de Petri dépouillée de certaines considérations théoriques dont l'interprétation au niveau de la synthèse risquerait d'être hermétique.

Les principales différences portent sur 4 points :

1°. La représentation graphique fait appel à des étapes (places) et des transitions qui sont les noeuds du réseau. Les notations utilisées présentent des différences par rapport à celles des RDP mais ceci ne semble pas fondamental.

2°. Une étape ne peut contenir qu'une seule marque. Dans ce cas elle sera dite active. Elle est désactivée dès que l'une de ses transitions de sortie est validée.

3°. Les conditions associées à une transition peuvent se referer à l'état d'activité des étapes.

4°. Une étape active amène le franchissement simultané de toutes les transitions de sorties valides. Un marqueur est donc divisible à volonté. De même, plusieurs marqueurs arrivant sur une place sont confondus.

Les trois derniers points donnent apparemment au GRAFCET une certaine souplesse dans la représentation des automatismes. Elles sont toutefois contraignantes au niveau de l'analyse et de la vérification.

V. EXEMPLE.

Il va nous permettre d'appliquer la formalisation des RDP à la représentation du cahier des charges d'un système logique.

a) Cahier des charges : système de réenclenchement d'un disjoncteur.

Ce système a pour but d'ouvrir un disjoncteur lorsqu'un défaut de durée supérieure à 0,2 secondes est détecté. Après la disparition de ce défaut consécutif à l'ouverture du disjoncteur, un ordre de réenclenchement est donné au bout de 0,6 secondes. Une temporisation de 10 secondes, nécessaire au remontage du mécanisme, débute. Si au bout de ce temps aucun défaut n'apparait, le système repasse à l'état de veille. Par contre, tout défaut supérieur à 0,2 secondes provoque le déclenchement définitif (un signal témoignant cet état est émis). La fermeture ne peut s'opérer alors que manuellement.

b) Représentation à l'aide des RDP (figure 1)

Mis à part les différences symboliques, cette représentation ressemble à un grafcet de niveau 1. Implicitement, par souci de commodité, les arcs convergent vers une même place sont représentés parfois comme convergent sur un arc aboutissant à cette place.

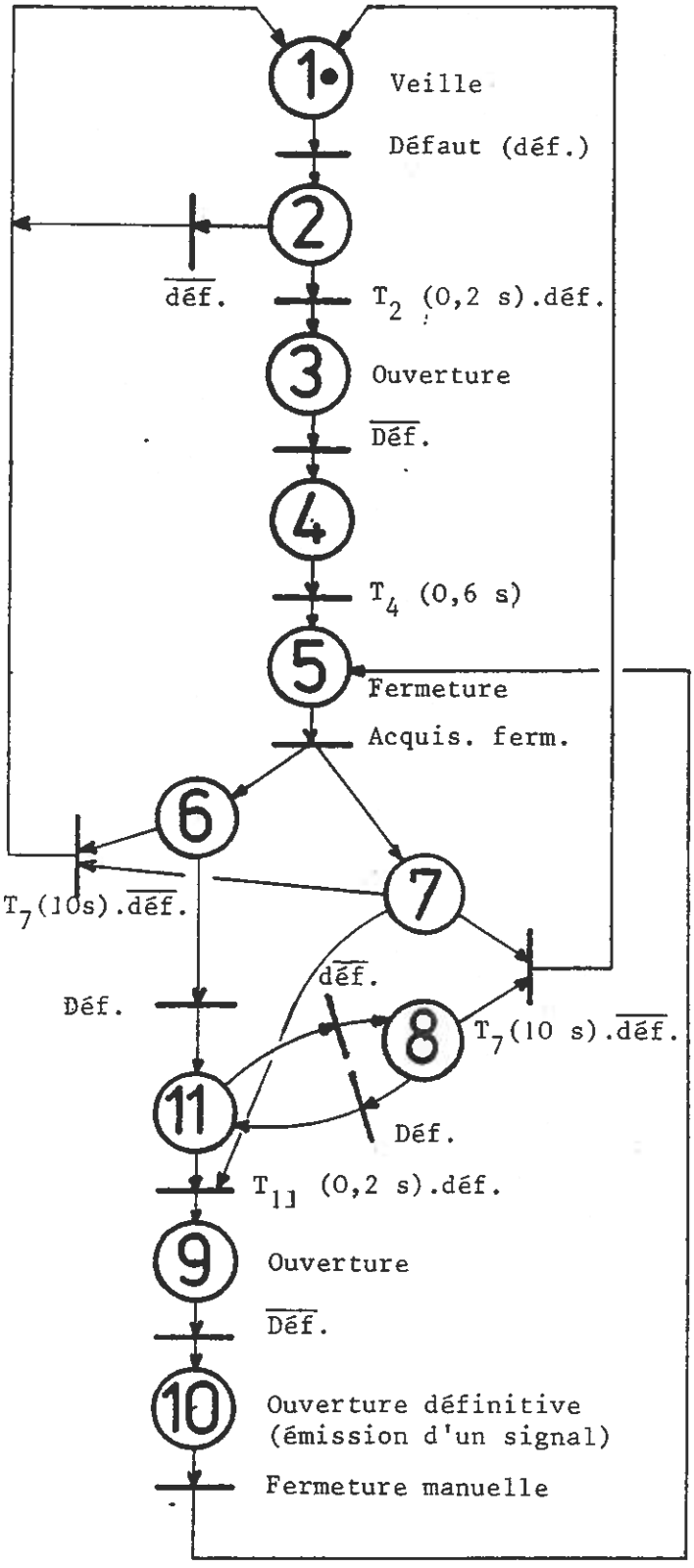


Fig. 1 : Système de réenclenchement.
Représentation fonctionnelle,
le marqueur initial est sur la
place 1.

Les temporisateurs (T) portent un indice indiquant la place où ils sont activés.

Cette convention est théoriquement fausse, les arcs ne pouvant constituer des noeuds du réseau. (La même notation est utilisée dans les organigrammes).
Ce schéma s'interprète de la façon suivante :

L'état de veille est représenté par la place 1. L'apparition d'un défaut fait passer le marqueur en 2 puis 3 si sa durée est supérieure à 0,2 secondes ; il y a alors ouverture du disjoncteur qui provoque la disparition du défaut et le réarmement du disjoncteur au bout de 0,6 secondes (place 4 puis 5). L'acquittement de la fermeture provoque le marquage de 6 et 7 (la place 7 est réservée au départ de la temporisation de 10 secondes). L'apparition d'un défaut inférieur à 0,2 secondes fait passer le marqueur de la place 6 à 11 puis 8. Par contre, si le défaut est supérieur à 0,2 secondes, le marqueur effectue le trajet 11, 9 puis 10 (ouverture définitive). Il y a au passage de 11 à 9 démarquage de la place 7, le temporisateur de 10 secondes devant être désactivé. La fermeture manuelle renvoie sur la place 5. Par contre, si à partir des marqueurs de 6 et 7 ou 8 et 7 il n'y a pas de défaut, le système repasse à l'état de veille au bout de 10 secondes.

A partir de cette étape fonctionnelle nous pouvons associer aux transitions et aux places les variables électriques qui serviront d'entrées/sorties à l'automatisme. Nous utiliserons pour cela la configuration de l'automate où X représente les entrées et Y les sorties.

Par convention nous considèrerons les variables actives au niveau 1.

- Variable de défaut : X0
- Acquittement de la fermeture du disjoncteur : X1
- Fermeture manuelle : X2
- Commande de fermeture du disjoncteur : Y0
- Commande d'ouverture du disjonteur : Y1
- Sortie indiquant une ouverture définitive : Y2

Nous obtenons alors le réseau de la figure 2.

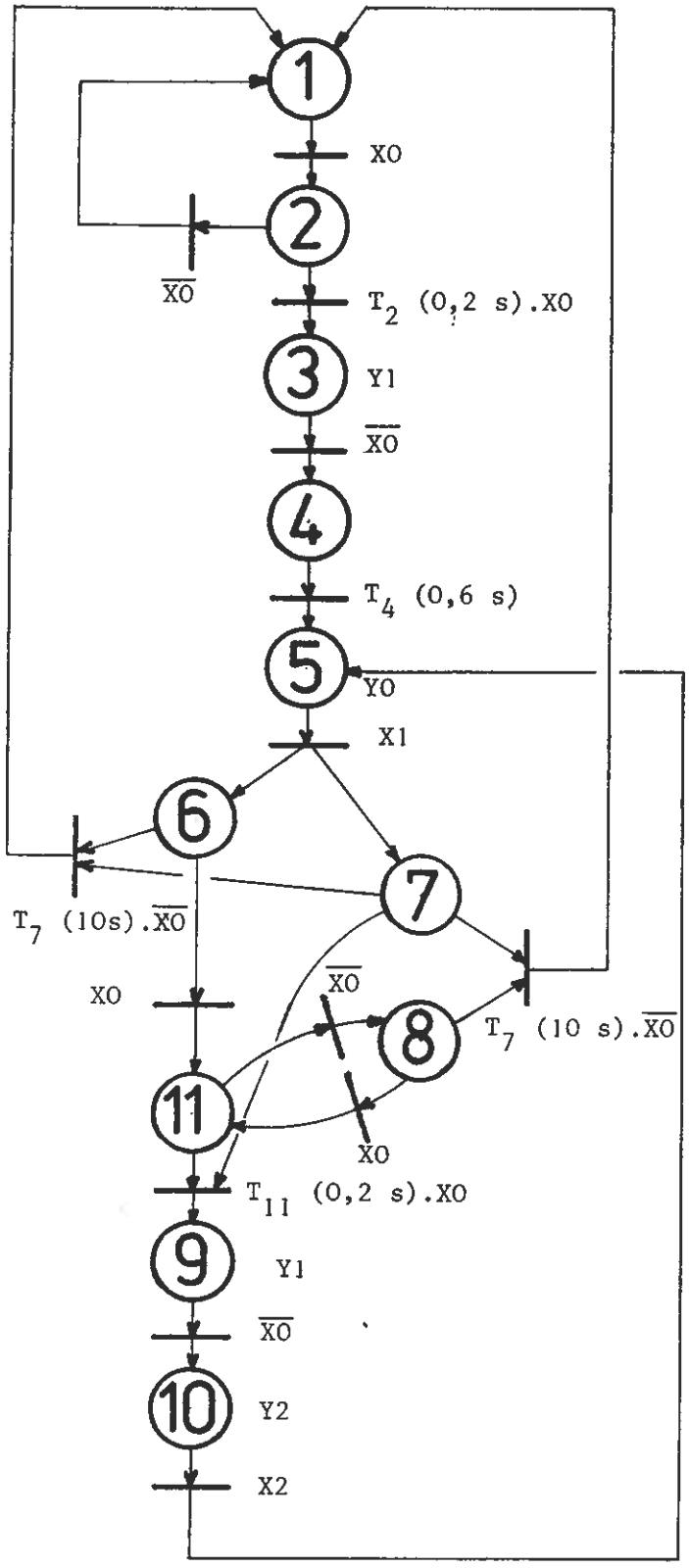


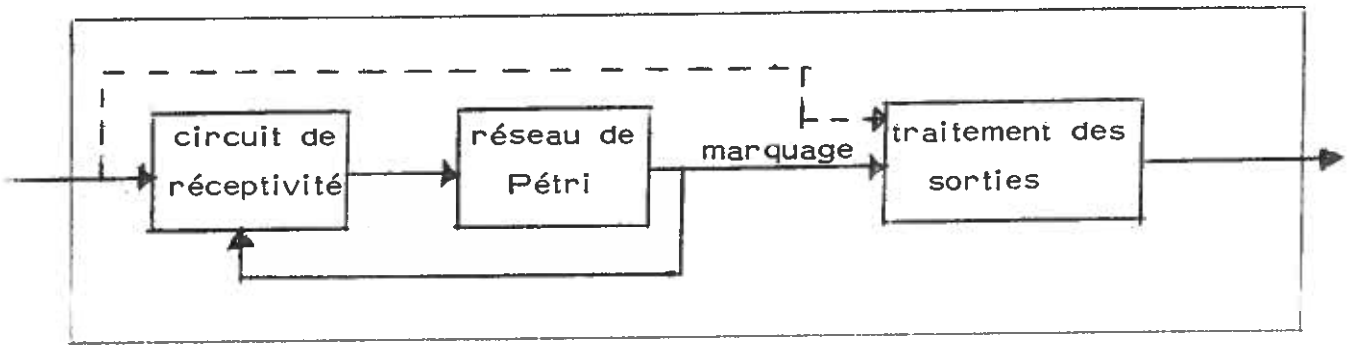
Fig. 2 : Système de réenclenchement. Représentation fonctionnelle à partir des variables électriques.

Une transition peut être franchie si la combinaison des variables qui lui sont associées est au niveau logique 1.

VI. REPRESENTATION D'UNE MACHINE DEFINIE PAR UN RDP.

Les méthodes classiques font intervenir des variables secondaires pour la détermination de l'état du système.

Dans le cas d'une représentation fonctionnelle par les RDP, ce sont les positions des marqueurs et leur évolution qui reflètent le fonctionnement de la machine. On peut la schématiser de la façon suivante :



Machine de MOORE

(La machine de Mealy comporte en plus la liaison en pointillée)

Trois types d'information permettent de la définir :

- D'une part, les conditions explicitant l'évolution des marqueurs, notamment celles liées aux transitions (réceptivité). Ce sont les relations d'incidences.
- D'autre part, les activités des sorties en fonction du marquage (Machine de Moore) et éventuellement des entrées (Machine de Mealy) ; c'est la définition d'un circuit combinatoire comportant notamment comme entrées des variables associées au marquage du réseau.
- Enfin le marquage initial.

La formalisation et le traitement de ces informations doivent être particulièrement analysés lors de la conception d'un automate programmable par définition fonctionnelle.

A suivre.

Une Nouvelle Machine...!

Patrick ISOARDI

C.R. : Subject classifications Informatics 4.21, 4.22, 5.25, 5.26

Résumé : "Une nouvelle machine...!" est la première partie d'une étude théorique dont le but est de réaliser enfin un processeur pour les informaticiens. Partant d'une conception un peu particulière de la mémoire, l'auteur développe au fil de son exposé un langage adapté à cette structure et qui conserve les propriétés contenues dans la Procédure Formelle pour finalement aboutir à la construction de cette nouvelle machine.

I. AU TITRE DE L'INTRODUCTION : QUELQUES BANALITES INFORMATIQUES.

1°. La machine

Si je considère une machine simple, je dirai au sens le plus large, voire le plus théorique, qu'elle se compose d'un algorithme et d'une mémoire. L'algorithme de cette machine est indéformable par construction ; la mémoire, elle, doit permettre d'enregistrer toutes les configurations sur lesquelles l'algorithme est susceptible de travailler ainsi qu'une représentation non ambiguë de l'algorithme.

Si l'on pose $M = A, S$

A c'est l'algorithme,

S la mémoire support de $\mathcal{C}$: $\mathcal{C}$ étant l'ensemble des configurations sur lesquelles travaille A .

Une application de la machine M donne :

$$M : R_S(A),\ C_1 \rightarrow R_S(A),\ C_2 \ .$$

C_1 et C_2 sont respectivement une configuration initiale et finale de $\mathcal{C}$.
$R_S(A)$ est une représentation de A .

De nombreuses idées, méthodes, ... emplâtres le plus souvent, ont été avancés concernant les algorithmes et la configuration des données, mais rien de définitif n'a encore été élaboré concernant la structure matérielle de la mémoire. A tel point qu'une machine simple, au sens où je l'ai définie, semblerait se présenter sous deux aspects : l'algorithme et la configuration des données. Certes, ces deux entités sont une partie importante de la machine et je les considèrerai comme il se doit dans la suite de mon exposé, mais elles ne sont pas disjointes de la mémoire, tant s'en faut.

2°. L'algorithme et les variables

La notion d'algorithme très répandue dans l'univers qui nous entoure est pourtant délicate à définir. L'une des difficultés qui font qu'elle est difficile à exposer tient sans doute à ce qu'elle se présente sous des formes diverses.

De plus, bien souvent les éléments qui la composent font l'objet d'appréciations subjectives. Avant toutefois d'aller plus loin, je vais essayer de construire et de commenter une définition précise de l'algorithme.

Je commence par me donner des ensembles d'objets appartenant à certaines catégories qui les caractérisent.
Je me donne ensuite des opérations élémentaires qui permettent d'obtenir un objet d'une catégorie déterminée à partir d'un ou de plusieurs objets de différentes catégories. Les objets peuvent être en nombre fini ou infini, mais les opérations et les catégories sont en nombre fini.

Un algorithme se compose d'une suite d'opérations. La description de l'opération précise le type de celle-ci et, en plus, les objets sur lesquels elle porte. L'algorithme représente ainsi une transformation appliquée à des ensembles d'objets.

On voit aussi apparaitre la notion de variable quand il s'agit de définir une sorte de creuset qui permet de traiter des objets différents mais possédant certaines propriétés communes et on verra que ceci tient à ce que l'on est obligé d'user de représentation : les objets eux-mêmes abstraits n'étant pas toujours manipulables. Mais qu'est-ce qu'une variable sinon le doublet (m, V) où m représente le nom de la variable et $V = \{x_1, x_2, \ldots, x_n\}$, x_i étant l'une des valeurs de la variable.

3°. Représentation des objets de la programmation

On n'a que trop tendance à considérer que les objets de la programmation sont, soit exclusivement les contenus de la case mémoire, soit exclusivement les objets que l'on manipule avec un crayon sur du papier. Il existe des relations entre ces deux sortes d'objets et ces relations sont précisément du ressort de la programmation au sens le plus large.

Si on sait décrire un algorithme dont j'appellerai programme sa représentation, il ne nous reste plus qu'à choisir le code et la manière dont le programme va s'étaler en mémoire.

Diverses solutions peuvent être envisagées : en général le choix dépend de la solution technologique et d'un ensemble de facteurs généralement complexes. De même, si les noms et les valeurs des variables sont représentés à l'aide d'un même ensemble de configuration - ceci étant une conséquence de la technologie choisie - il n'en reste pas moins vrai que leurs représentations en mémoire n'ont pas la même structure, pas plus d'ailleurs que le programme.

L'ambiguïté apparait alors lorsqu'on se préoccupe de réaliser des empilements d'information pris dans ces diverses catégories d'objets. En effet, on se complait de mélanger, certes le plus agréablement possible mais aussi le plus souvent sans grande réussite, trois objets de propriétés différentes dans ce même plat à structure unique qui est la mémoire.

4°. La mémoire

Pour lever cette ambiguïté, deux solutions me semblent convenir :

- La première solution a été développée par E. Bianco à propos du MIL [1] où il a montré la nécessité d'une description complète des propriétés géométriques des empilements qui apparaissent nécessairement au cours de la programmation de problèmes complexes (compilation, gestion de fichiers, etc...). Dans le cas le plus général, la connaissance du problème n'entraine pas de manière unique la détermination de la structure des empilements ; mais une restriction de cette dernière afin de limiter les possibilités de structure rend la description et la relecture plus aisées.

- La deuxième solution est celle que j'apporte : ce n'est pas une révolution, c'est dans un premier temps une évolution théorique.
La suite logique de mes précédentes considérations m'impose d'être ici radical en considérant non pas une seule mémoire unique, mais trois mémoires distinctes : l'une pour supporter, que dis-je accueillir le programme ; une autre pour les noms ou adresses et la dernière pour les valeurs des variables. Chacune de ces mémoires étant bien entendu structurée en fonction de l'unique catégorie d'objets qu'elle est susceptible de contenir.
Il est certain que les cages subsistent, mais ne sont-elles pas plutôt des habits confectionnés que des armures imposées ? Ne sont-elles pas mieux adaptées au traitement des objets qu'elles accueillent ?

Comme je l'ai dit précédemment, cette rupture de la mémoire influe sur la description de l'algorithme et sur la configuration des données. Je vais donc maintenant me préoccuper de ces deux derniers points en essayant toutefois de conserver des propriétés fondamentales qué j'ai fait miennes parce que tout simplement je les ai éprouvées, qu'elles se sont avérées interessantes et que je vais ici rappeler.

5°. La Procédure Formelle

On remarque dans la définition de l'algorithme que la recherche de l'acte élémentaire dont la composition avec d'autres actes élémentaires sert à décrire un acte plus complexe ne peut se faire dans l'absolu. Ainsi, rien n'empêche que ces opérations complexes particulières ne deviennent à leur tour acte élémentaire pour réaliser la construction d'opérations encore plus complexes. C'est cette notion de procédure avec son insertion qui se trouve dans la Procédure Formelle telle qu'elle est décrite dans le cours de C4 [2] . La Procédure Formelle est un langage de programmation de la puissance de la Machine de Turing dont la structure permet une programmation beaucoup plus facile, très proche des langages machines les plus perfectionnés.

Pourtant, là aussi travailler sans déclaration sur les noms et sur les valeurs des variables avec le même type d'instruction crée une ambiguïté. En effet, à la simple lecture d'un algorithme écrit en Procédure Formelle, on ne sait pas séparer la partie traitement : c'est à dire le calcul effectif sur les valeurs et la commutation interne : c'est à dire le calcul sur les noms.

La solution de ce problème se trouve dans ma thèse de 3ème cycle [3] . C'est là que j'ai démontré qu'en déclarant dans chaque procédure les cases mémoire comportant des adresses, on peut séparer dans l'algorithme les instructions portant sur les valeurs de celles portant sur les adresses. Le fait de rajouter au langage de la Procédure Formelle une simple déclaration est alors suffisant pour séparer le calcul effectif de la commutation interne.

Qu'advient-il de ce langage avec la nouvelle conception de la mémoire ?
Je vais donc, à partir du langage de la Procédure Formelle, définir une machine qui conserve les propriétés ci-dessus énumérées.

II. LA NOUVELLE MACHINE.

1°. La mémoire

Partons de la nouvelle conception de la mémoire :

- une mémoire technologiquement structurée pour l'accueil des valeurs des variables que j'appellerai : mémoire des valeurs.
- une mémoire réservée aux adresses.
- une mémoire pour les programmes.

2°. L'algorithme et la configuration des données

Pour parfaire à la définition de l'algorithme, je rajouterai les évidences suivantes :

Il est clair que l'algorithme et la configuration des données ne sont pas des entités disjointes : en effet, dans l'algorithme doivent apparaitre les désignations des objets qui composent la configuration et la manière dont les objets évoluent dans la configuration dépend évidemment de l'algorithme.

Il apparait aussi clairement qu'il existe deux catégories de variables utilisées dans un algorithme :

- celles qui n'ont d'existence que pendant le calcul qui correspond à cet algorithme : ce sont les variables locales.
- celles qui ont une existence extérieure fournissant les données et véhiculant les résultats : ce sont les paramètres.

De plus, dans l'insertion, au lieu de transférer les valeurs, on va transférer les noms des variables. Si volumineuse que soit la représentation de la valeur d'une variable (vecteur, matrice) son nom est unique ; c'est lui que nous transmettrons, plus, évidemment, des indications sur la nature de la représentation (exemple : la dimension du vecteur) ; mais là encore, ce ne sera pas la dimension elle-même mais son nom que l'on transmettra.

On part de l'idée que tout ce qui caractérise la valeur d'une variable ne doit apparaitre qu'une seule fois dans la mémoire ; ce sont seulement les noms de ces objets qui sont dupliqués et traités lors de l'insertion.
On va étudier les structures à appliquer à ces parties de la machine pour obtenir ces propriétés précises.

3°. La procédure

Je vais d'abord, à propos des configurations et de l'algorithme, rappeler quelques définitions bien connues [2] et que je respecterai dans la construction de mon langage :

Définition I : On dit que deux configurations sont homologues si elles diffèrent par les noms des cases mémoire et les contenus de ces cases mais si les types de variables représentées par les cases se correspondent deux à deux.

Définition II : On dit que deux algorithmes sont équivalents s'ils ne diffèrent que par les configurations des noms de cases sur lesquelles ils travaillent et si ces configurations sont homologues. On appelle procédure la classe d'équivalence ainsi définie.

La définition I suggère la remarque suivante : il est commode, bien que ce ne soit pas contenu dans la définition, d'imaginer que les éléments d'une même configuration sont ordonnés une fois pour toutes. Cela implique que tous les autres membres de la même classe d'homologie ont aussi leurs éléments ordonnés par rapport aux types de variables.

En transposant cette remarque sur les mémoires de notre machine, on va pouvoir définir un procédé de construction : en effet, on peut remplacer les noms des objets par une numérotation dans la suite des éléments entiers ordonnés. L'algorithme ne connaitra des variables que ces numéros. Il suffira de lui fournir en mémoire à partir d'origines qui deviennent arbitraires toute suite de cases dont les contenus sont de type compatible avec la classe d'homologie.

Pour réaliser cela facilement, on va rajouter à notre machine trois cases ou registres A, V, P dont le rôle est ainsi fixé :

a) Si les mémoires des valeurs, adresses et programmes disposent respectivement de N_v, N_a, N_p cases, les contenus des registres V, A et P peuvent varier de 0 à $N_v - 1$; 0 à $N_a - 1$; 0 à $N_p - 1$ respectivement.

b) Chacun des contenus des registres V, A et P notés CV, CA et CP est en fait considéré comme un numéro de case absolu ou relatif pour chacune de leur mémoire respective : c'est à dire mémoire des valeurs, mémoire des adresses, mémoire des programmes.

c) On peut exercer sur ces registres toutes les opérations que l'on peut exercer sur les cases mémoire dans la correspondance des types.

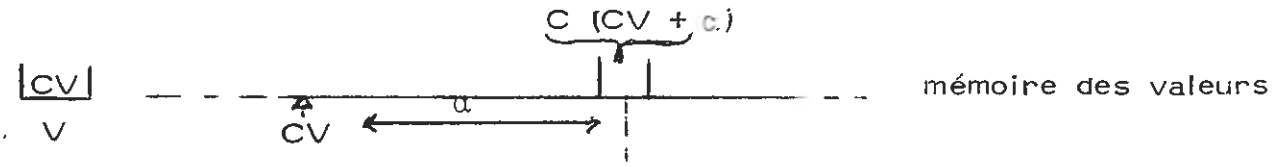
4°. Les opérandes

A l'exception de ces cases spéciales A, V, P les autres opérandes seront désignés sous la forme :

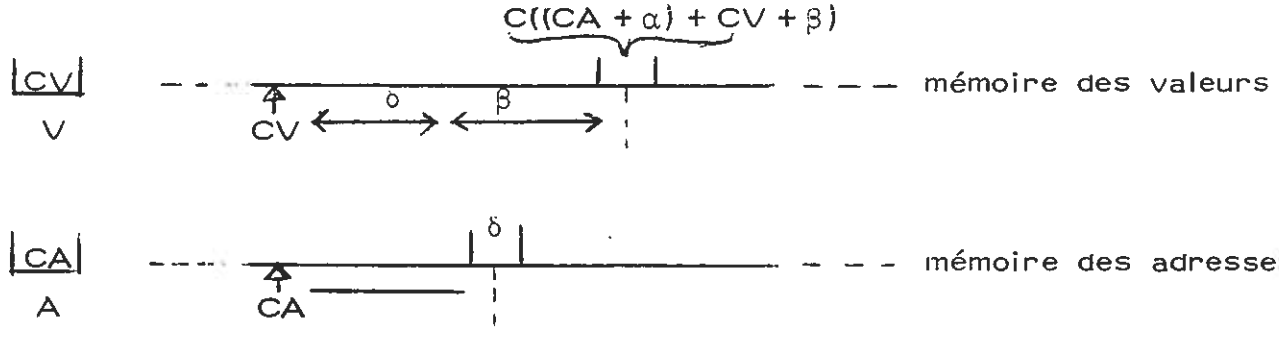
C (CV + α)
C (CA + α)
C ((CA + α) + CV + β)
C ((CA + α) + CA + β)

dont la sémantique est définie sur les schémas suivants :

a) Atteinte directe de la valeur : C (CV + α)



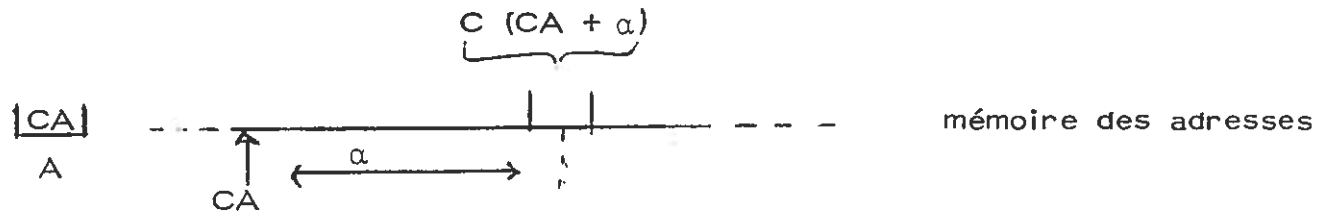
b) Adressage indirect de la valeur : C ((CA + α) + CV + β)



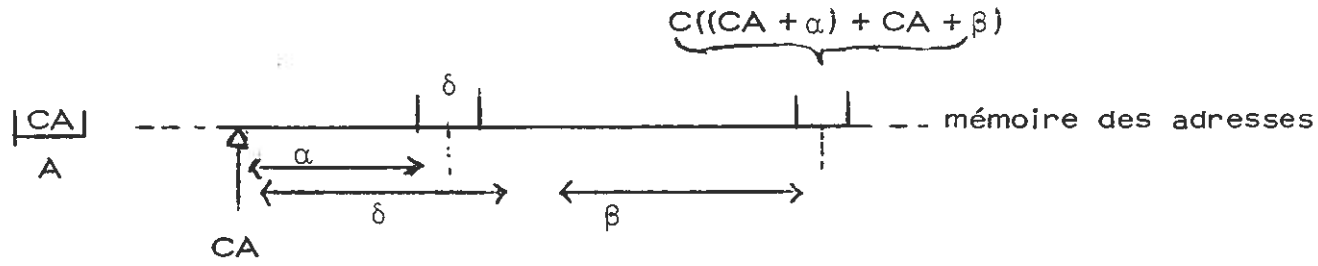
Ces deux modes d'adressage permettent de faire du calcul effectif : c'est à dire du calcul sur la valeur de la variable ; le premier adressage est utilisé pour les variables locales de l'algorithme ; le deuxième est utilisé, lui, pour atteindre les valeurs des paramètres par l'intermédiaire de leur adresse.

Contrairement à cela, les deux autres modes permettent de faire de la commutation interne : c'est à dire du calcul sur les adresses. Je les définis de la façon suivante :

a) Adressage direct : C (CA + α)



b) Adressage indirect : C ((CA + α) + CA + β)

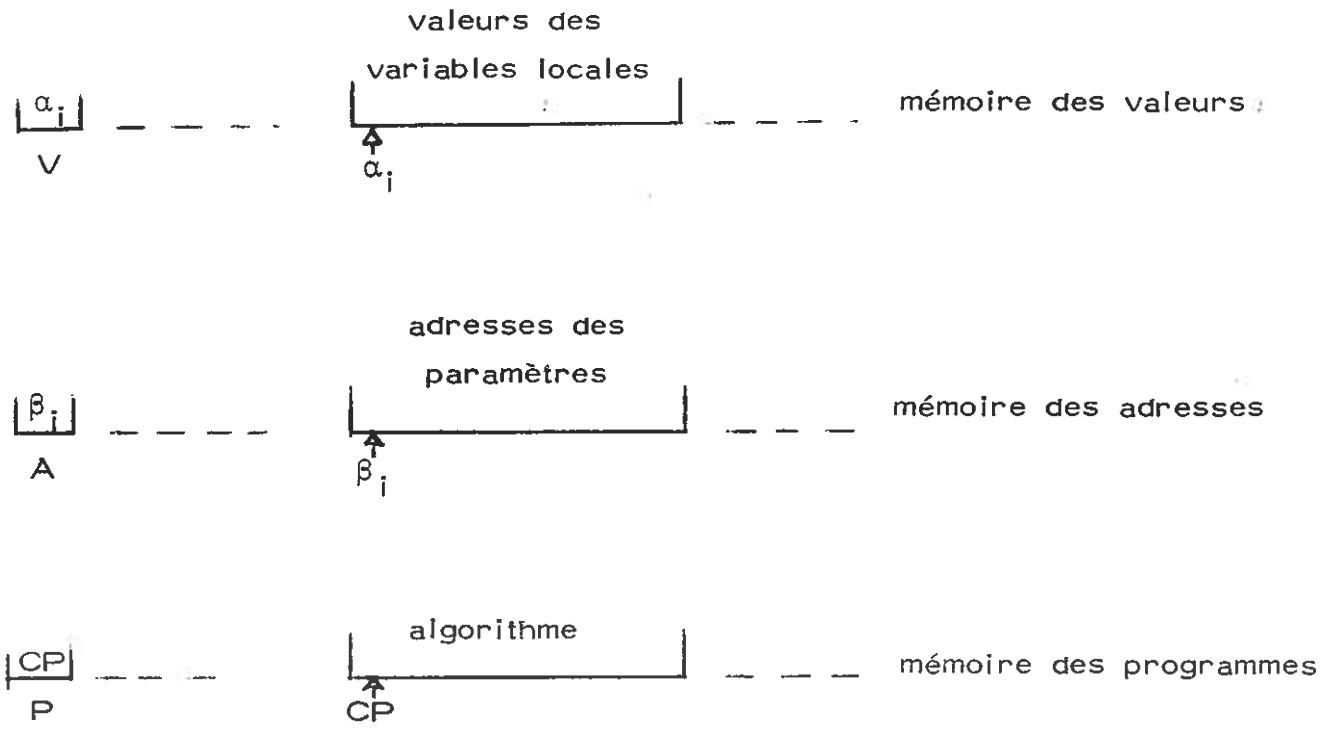


La séparation entre le calcul effectif sur les valeurs et le calcul sur les adresses apparait ici directement dans le langage et cela sans aucune déclaration.

5°. Configuration de la procédure

La configuration des données de l'algorithme est alors un couple (R (v), R (a)) où R (v) représente la configuration des valeurs des variables locales et se trouve dans la mémoire des valeurs et R (a) représente la configuration des adresses des paramètres dans la mémoire adresses.

On a ainsi le schéma suivant de la procédure :



En supposant que dans les mémoires se trouve un certain nombre de configurations de données $(R(v_1), R(a_1))$; ... ; $(R(v_n), R(a_n))$ et si ces n configurations sont disjointes et repérées chacune par les numéros de leurs premières cases $(\alpha_1,\beta_1),\ldots,(\alpha_n,\beta_n)$, il suffit de donner l'une quelconque des valeurs α_i au registre V et la valeur β_i à A pour que l'algorithme puisse opérer sur le jeu des variables correspondant. On a bien ce que l'on cherchait : une écriture de l'algorithme indépendante du jeu des variables sur lequel il travaille.
Ceci a été obtenu moyennant les contraintes suivantes :

a) A chaque procédure on fait correspondre deux suites de cases suffisantes pour représenter sa configuration de valeurs et sa configuration des adresses.

b) Chacune de ces suites est constituée de cases successives prises par adresses croissantes à partir d'une case-origine.

c) Les suites de cases en question n'existent comme support de configuration que pendant le déroulement de la procédure correspondante.

d) Le repérage de chaque configuration se fait au moyen d'un couple d'adresses qui sont les valeurs des variables supplémentaires qui apparaissent dans l'algorithme : A et V .

Ces contraintes ne sont pas exhorbitantes au regard du gain obtenu dans la reconnaissance du type des variables.

6°. L'insertion de procédure

Nous avons construit une sorte de programme-standard capable de travailler sur n'importe quel jeu de données pourvu que ce jeu soit compatible selon les types. Il est nécessaire dès lors d'en prévoir l'utilisation. Ceci signifie que ce programme ayant une existence unique (son intérêt est là) il faudra pouvoir s'y référer en divers points d'un autre programme : on dira l'insérer en ces points car tout doit se passer comme si effectivement on introduisait en ces points une suite d'instructions identiques à celles de notre programme-standard.
Pour que ceci soit possible il faut connaitre :

- Le nom de l'algorithme à insérer. Au niveau du langage on apporte les compléments suivants en leur accordant la sémantique nécessaire :

a) Chaque algorithme de procédure constitue un bloc encadré par deux symboles : "début" ; "fin". On attribue à cet algorithme un nom qui se note comme une étiquette et qui est accolé au début de l'algorithme.

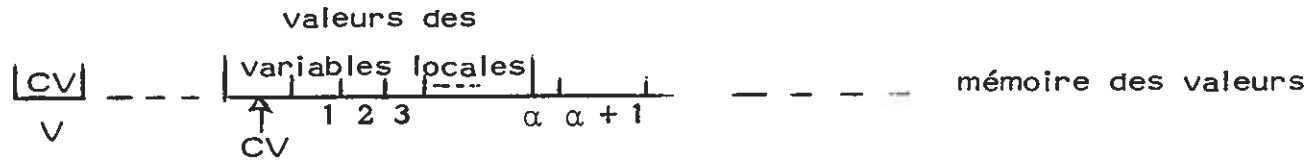
b) On rajoute une instruction qui se note "insérer P" : le P représente le nom de l'algorithme au sens où on vient de le définir.

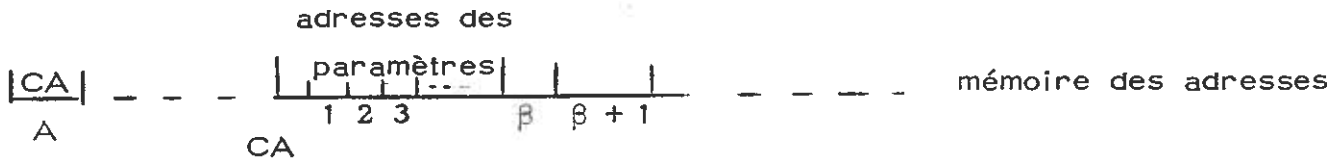
- L'algorithme inséré ayant une configuration de données dont la structure est fixée par ailleurs, il s'agit d'établir la liaison entre la configuration de la procédure insérante et celle de la procédure insérée. Pour ce faire :

a) Dans l'insertion, au lieu de transférer les valeurs, on va transférer des noms de variables : il s'agit là de la poursuite intuitive de la démarche précédente. Quant aux variables de la procédure insérante que l'on met en correspondance avec les paramètres dits formels de la procédure insérée, on les appellera : paramètres effectifs.

b) C'est dans celle de la procédure insérante que l'on fait apparaitre la zone des données de la procédure insérée : une première solution consiste simplement à accoler, dans chaque mémoire, les zones à insérer à côté des zones insérantes.

Si l'on imagine que les variables de la procédure insérante sont ainsi réparties dans les mémoires :

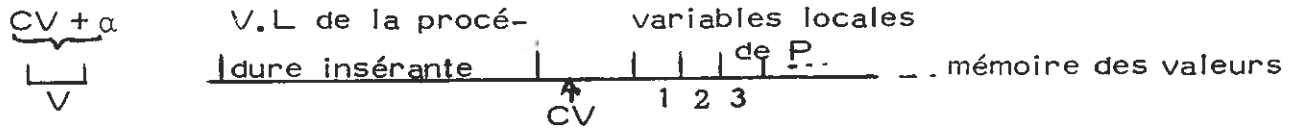


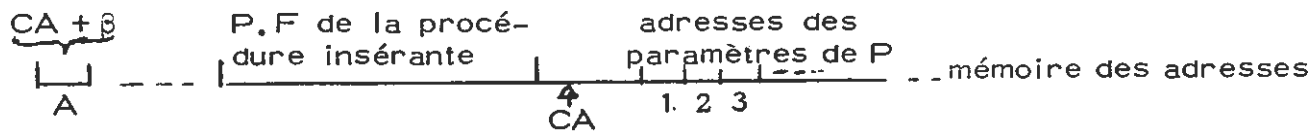


On suppose qu'au moment d'insérer, les cases de numéros α, $\alpha + 1$, et β, $\beta + 1$,... sont libres et vont être désignées pour contenir les variables de l'algorithme P à insérer.
Le déroulement des opérations se produit dans cet ordre :

a). Correspondance entre les paramètres formels et les paramètres effectifs sur la zone des adresses en prévoyant le futur déplacement des cases-origine dans la détermination de l'adresse.

b). Déplacement des index sur α et β .





c). "Insérer P" : Déroulement de l'algorithme de la procédure P.

d). Récupération des anciennes valeurs de A et de V .

e). Retour à l'instruction qui suit "Insérer P".

7°. Le programme

La manière dont nous décrivons désormais les algorithmes appelle quelques réflexions car nous avons introduit des notions que nous avons justifiées de raisons intuitives, mais sur lesquelles il est bon de revenir. On a vu qu'en fait, l'insertion se traduit par une instruction :

"Insérer P"

qui joue le rôle d'une liaison un peu comme :

"Vers" E

juste un peu plus compliquée : le P ou le E correspondant à une adresse de programme ou d'instruction dans le seond cas ; c'est à dire relative au contenu de P : CP .

Pourtant, il n'existe pas dans le langage d'opérande qui permette d'atteindre le contenu d'une case d'un programme. Je dirai : heureusement...! En effet, pour protéger un programme des indiscrétions, on le rend impénétrable ; le plus souvent, pour atteindre le comble du couillonisme - ceci a été dit, mais il faut insister - on le construit autodestructible.

Pour ma part, n'ayant aujourd'hui de secret pour personne et considérant qu'un programme ne doit être protégé que de lui-même, le langage que je développe ici ne permettra pas à un programme de s'autodétruire : c'est à dire d'avoir le pouvoir de modifier ses propres instructions pendant qu'il se déroule.

Cela ne veut pas dire que l'on ne peut pas accéder à un programme ou atteindre divers points d'un programme. Bien au contraire : pour cela il nous faut enregistrer l'adresse du début du programme ou dans le deuxième cas l'adresse de retour dans la procédure ainsi que les origines de la configuration de données. Toutes les valeurs sont des adresses ; n'ont-elles pas leur place dans la mémoire des adresses au côté des noms des paramètres formels ?

8°. Le langage symbolique

Si je reprends l'insertion de procédure, on a vu qu'il fallait plusieurs instructions pour créer la zone de données de la procédure insérée et en particulier faire évoluer les origines de la configuration avant et après l'insertion. On a donc présenté les choses de manière sinon explicite tout au moins formelle.
Mais on aurait pu présenter les choses implicitement en écrivant par exemple :

Insérer P $(\alpha_1, \alpha_2, \ldots, \alpha_i)$

les α_k étant des variables de la procédure insérante (disons des numéros relatifs de cases des paramètres effectifs) à faire correspondre dans l'ordre avec la liste des paramètres formels de la procédure à insérer P .
Le travail non négligeable qui consiste à calculer et à transférer les adresses dans la zone à créer est alors supposé fait automatiquement : on dit qu'on a créé un langage de programmation symbolique, puisque on laisse sous-entendre une partie du traitement dont on sait par ailleurs qu'il est calculable.

De même, les noms des variables qui se notent :

C (CV + α)
C (CA + α)
C ((CA + α) + CV + β)
C ((CA + α) + CA + β)

sont des écritures compliquées qui n'ont d'intérêt que si elles servent d'aide-mémoire pour retrouver l'algorithme de calcul de l'adresse effective. Mais l'algorithme étant saisi, l'écriture se révèle bien lourde : on va la condenser. Pour cela, il suffit de remarquer que l'information suffisante pour caractériser les différentes variables sont A, V et les paramètres α et β . On choisira donc les écritures condensées :

V_α pour C (CV + α)
A_α pour C (CA + α)
$V_{\alpha,\beta}$ pour C ((CA + α) + CV + β)
$A_{\alpha,\beta}$ pour C ((CA + α) + CA + β)

Pour ce qui est des instructions, je considèrerai comme faisant partie intégrante du langage toutes les instructions opérationnelles et conditionnelles des langages machines les plus performants et les plus évolués. Ce n'est pas ici une gêne ... bien au contraire !

9°. Quelques propriétés

J'ai mis en évidence lors de la construction que la séparation entre le calcul effectif et la commutation interne est portée par les différences de type entre les noms des variables ou opérandes du langage.

Une autre propriété apparait : la récursivité ; En effet, dans une procédure P on peut insérer P . Ceci est essentiellement dû à la structure de la configuration des données et au fait que le support de cette configuration n'existe que pendant le déroulement.

Mais la propriété essentielle reste la séparation fondamentale entre les valeurs, les noms et les programmes.

III. CONCLUSION.

Il faudrait démontrer que la machine que je viens de construire est aussi puissante que n'importe quelle machine universelle. Je le crois fermement, mais ceci n'est pas une démonstration. La démonstration viendra, peut-être dans un autre article ; pour le moment c'est tout simplement une idée que j'ai développée - Le bulletin n'a-t-il pas été créé pour cela !
Ce n'est pas non plus une idée pour essayer de plaire,..., mais pour montrer que l'expérience acquise, les erreurs commises permettent de progresser et d'innover.

Jusqu'à présent, le dialogue entre les informaticiens et les constructeurs de processeurs a été inexistant. Celui-là s'est accommodé de la machine de celui-ci. Aussi, le but final non encore avoué est de définir une structure de processeur pour l'informaticien, d'en tirer les avantages et les inconvénients. En un mot de tirer profit de l'expérience et de l'erreur.

Peut-être que cette nouvelle structure théorique ne sera pas -disons- "intéressante" du point de vue pratique. Puisse cet exposé créer le dialogue tant recherché ! Le grand pas ne serait-il pas alors franchi ?

= : = : = : = : =

BIBLIOGRAPHIE

[1] Edmond BIANCO — Etude et formalisation d'une classe de systèmes Thèse d'Etat, 1969.

[2] Edmond BIANCO — Informatique Fondamentale - ISR 70 - Birkhauser-Verlag.

[3] Patrick ISOARDI — Structures de commutation. Thèse de 3ème cycle, 1978.