BULLETIN D'INFORMATIQUE APPROFONDIE ET APPLICATIONS

N° 47 JUIN 1997

SCIENCES DE L'EDUCATION ET DE L'INFORMATION

REDACTION

Université de Provence
Equipe Hermès. Case 33
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: 04 91 10 62 30
Télécopie : 04 91 50 91 10

DEPOSITAIRE

Université de Provence
Bibliothèque Universitaire
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: 04 91 62 44 16
Télécopie : 04 91 95 75 57

D'ici quelque temps le bulletin aura ses informations sur WWW:
http://www.univ-mrs.fr http://www.u-3mrs.fr

Publication trimestrielle, gratuite, de l'Université de Provence. **ISSN 0291 - 5413**

EDITORIAL,

QUESTIONS.

S'est-on demandé comment réagiraient les ordinateurs à qui on proposerait 35 heures de travail payées 39 ?

Rajustant quelque peu le problème de Turing, pouvons-nous penser qu'un ordinateur banal, comme on en trouve dans toutes les bonnes grandes surfaces, dûment muni d'une boussole et abandonné au milieu d'un désert retrouverait son chemin ?

Peut-on imaginer le roman comique que pourraient écrire des auteurs tels que Cyrano de Bergerac, Alphonse Allais ou Alfred Jarry à propos de personnages qui, après avoir subrepticement fait passer de 85 à 128 heures les charges d'enseignement des professeurs de faculté, sans aucune compensation, viennent leur proposer 35 heures payées 39 ?

Mais, " foin de questions oiseuses ", voire amusantes, de temps à autres n'est-il pas bon de poser de véritables questions sérieuses, cet énoncé étant lui-même une question, qui entraîne immédiatement la question de savoir dans laquelle de ces catégories on doit la classer. Ainsi de suite. Jusqu'au sexe des anges.

Et pendant ce temps-là, les chauffeurs de poids-lourds qui ont compris quelles positions névralgiques et stratégiques ils occupent dans la société, bloquent tout, c'est-à-dire l'accès à l'essence, aveuglément. Ces gens-là n'ont visiblement rien compris à l'importance de la construction européenne. Ils n'ont pas compris que seule la loi du profit maximal permettait à nos financiers de manoeuvrer le maximum de divisions de milliards de francs, face aux divisions de milliards de marks, de dollars, de florins, de lires et toutes autres monnaies un peu fortes. Ils n'ont pas compris que, simples pièces d'une machine à faire du fric, mais pièces coûteuses surtout, semble-t-il à cause de vieilles lois sociales périmées, ils se trouvent au premier plan des "économies" à réaliser. Leur mouvement dérange la foule des technocrates européens, qui, brusquement s'agitent comme des poules troublées dans leur sommeil.

Et puis on s'aperçoit soudain à quel point le transport routier pollue et encombre, coûte cher en réseau autoroutier et devient dangereux dans les circulations de plus en plus denses. Oui mais. La vente du mazout rapporte beaucoup d'impôts, la construction d'autoroutes est une manne pour les grandes sociétés de travaux publics amies de tous les partis politiques. Alors ?

Alors Monsieur le Contribuable arrive à point pour payer la note. Ce constat est devenu si banal qu'on n'y porte pas plus attention qu'à un quelconque cliché, ou à un leitmotiv, un poncif, un truisme. Mais autre question sinon sérieuse, du moins dangereuse, le contribuable ne va-t-il pas se lasser de jouer perpétuellement ce rôle, surtout s'il perçoit qu'il n'a plus grand-chose à perdre ?

Tout est prévu, on lui a organisé la solution à tous ses maux, face aux petites réalités mesquines, on va l'inonder d'une mer d'illusions. Il va pouvoir embarquer sur des bateaux du rêve comme personne n'aurait pu en imaginer jusqu'à ce jour, on va lui donner l'occasion de se confronter au monde entier, voire à tout l'univers, sans même sortir de chez lui. Bref, INTERNET va lui permettre de combler enfin la soif d'absolu dont il se consume.

Espérons toutefois qu'aucune catastrophe écologique naturelle ou provoquée ne vienne interrompre la distribution d'électricité ...

Edmond Bianco

CHAPITRE 2. UNE MACHINE ELEMENTAIRE,

Edmond Bianco

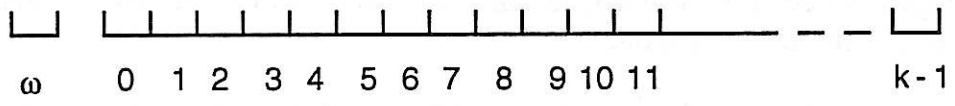
LA MACHINE DE NOLIN.

La variété des mécanismes qu'on peut se procurer dans le commerce pourrait laisser imaginer qu'il existe une quantité illimitée de manières de calculer. Ce qui, vrai dans un sens, n'éclaire guère le sens profond de ce qu'est le calcul automatique. Or on va essayer de montrer que s'il existe une grande variété d'ordinateurs tous différents les uns des autres, et tous meilleurs les uns que les autres, les règles de base du calcul sont les mêmes pour tous. Dans son apparence, le problème se complique d'autant que les systèmes d'exploitation sont variés, les logiciels d'application divers voire disparates, certaines marques plutôt spécialisées dans certaines catégories de traitement, d'autres dans d'autres. Vu de l'extérieur une richesse qui, d'ailleurs, s'accroît sans cesse, mais ne facilite pas l'appréhension du phénomène fondamental.

Je vais choisir un modèle de machine très dépouillé pour ne pas se perdre dans un dédale de détails peut-être importants sur le plan commercial, mais sans aucun intérêt quand il s'agit de la compréhension. La machine de Nolin, d'une extrême simplicité quant à sa structure va nous permettre d'analyser clairement les propriétés qu'on rencontre sous une forme plus ou moins implicite dans rigoureusement tous les ordinateurs. Ceci afin d'effacer toute plage de sémantique floue, propice à la rêverie de science fiction, mais néfaste à la réflexion.

LES DONNEES.

Je me donne d'abord un moyen mécanique de noter l'information. Je l'appelle **mémoire centrale**, et je la dote d'une case spéciale "w", ainsi que de k cases toutes identiques entre elles de par leurs propriétés mais dont chacune est reconnaissable parmi les autres grâce à son nom.



Il me faut alors définir les contenus possibles de chacune de ces cases. Je dirai simplement qu'à l'exception de w le contenu de la case est l'un des objets pris dans une liste de K objets différents. Par exemple:

0, 1, 2, 3, 4, 5, ... (K-1)

C'est ainsi qu'à titre d'exemple, si je choisis pour K la valeur K=9, le contenu de la case est l'une des valeurs:

0, 1, 2, 3, 4, 5, 6, 7, 8

et exclusivement l'une de ces valeurs possibles. On remarque que dans ce cas, le contenu global de la mémoire est, en quelque sorte un nombre à n chiffres écrit en base 9. Plus généralement donc, un contenu de mémoire à un instant déterminé se comporte comme un nombre de n chiffres écrit en base K.

L'ALGORITHME.

Je me suis donné un moyen de noter l'information, il faut alors pouvoir faire évoluer cette information. Je munis la machine d'un jeu d'opérateurs à partir desquels je pourrai construire tout élément de calcul. Pour utiliser tout de suite le jargon du spécialiste je baptise **instructions** ces opérateurs. Des "**statements**" en informaticien international.

C1. **ca := 0**
C2. **ca := cb**
C3. **ca := ca +$_K$ 1**
C4. **si ca = 0 vers ei**
C5. **vers ej**

Je compléterai ultérieurement cette machine au moyen d'une extension illimitée de mémoire et d'un jeu d'instructions qui permette de l'utiliser. Pour l'instant nous allons nous pencher sur le travail dans la mémoire centrale. Je définis alors la sémantique, le sens, des instructions dont je peux disposer. D'abord je précise que pour une case quelconque dont le nom est **a**, la notation **ca** représente le contenu de cette case **a**.

Le cas C1: a ou b désigne n'importe laquelle des cases de la mémoire. Je prends pour exemple c3 := 0 quel que soit le contenu de la case 3 avant application de cette instruction, il prend de force la valeur 0 après application de l'instruction. Il en serait de même pour une case 12 en déroulant une instruction c12 := 0 .

Le cas C2: je désire dérouler l'instruction c5 := c9 et je suppose que la case 9 contient 4, alors, après application de l'instruction la case 5 contiendra aussi la valeur 4. Cette instruction est une duplication. Elle copie dans la case de gauche ce que contient la case qui est à droite du ":=" , sans en effacer pour autant le contenu.

Le cas C3: c'est la seule instruction capable de faire évoluer un contenu de case. Pour imaginer ce qu'elle fait, je raisonne sur la suite des valeurs possibles qu'on peut enregistrer dans une case. Je choisis K, par exemple: K=7.

0, 1, 2, 3, 4, 5, 6

sont des représentations des seules valeurs qu'on puisse alors rencontrer dans une case. Je vais tenter de définir une arithmétique locale dont il me faudra montrer qu'elle a les propriétés de l'arithmétique tout court. Pour cela je dirai que cette suite est ordonnée, et je me donne un opérateur de

succession. De la sorte, 1 est le successeur de 0, 2 le successeur de 1, etc. 6 est le successeur de 5. Quant au successeur de 6, je l'écris 10, mais dans la machine c'est le 0 qui succède au 6 dans la case a, et il faut faire apparaître le 1 quelque part. Ce sera dans la case w. Ce qui définit en même temps le rôle de cette case particulière. Donc en résumé, pour une instruction:

$c4 := c4 +_K 1$ le résultat de l'application est le suivant:

avant application	après application
si $c4 \neq K-1$	cw = 0 et c4 = c4 + 1
si c4 = K-1	cw = 1 et c4 = 0

Le cas C4 et le cas C5. Pour expliciter ces instructions il me faut définir la structure de l'algorithme. Je dirai donc qu'un algorithme se compose d'une suite d'instructions dont la forme de chacune est choisie exclusivement parmi l'une des formes présentées ci-dessus. Le début en est repéré par une marque **début** et la fin par une marque **fin** . Et entre, à partir de début jusqu'à fin, les instructions sont présentées dans l'ordre où elles devront être déroulées. L'exception provient précisément des cas C4 et C5. Dérouler une instruction C5 revient à prendre pour instruction suivante, non pas celle qui suit dans l'ordre d'écriture, mais celle qui porte la même étiquette.

L'étiquette est un identificateur, un mot formé avec des lettres, et même des chiffres, pourvu qu'il ne commence pas par un chiffre. Dans le langage de la machine de Nolin, l'étiquette sert à établir un lien entre une instruction du type C4 ou C5: affectation d'étiquette, et une instruction quelconque de l'algorithme: déclaration de l'étiquette.

$$E : \; ca := ca +_K 1$$

Pour l'instruction C4, il y a un choix possible. Si la condition "ca = 0" est vérifiée, alors l'instruction suivante à dérouler est celle qui est indiquée par l'étiquette. Sinon c'est l'instruction écrite à la suite qui est prise comme la suivante.

Il faut rajouter des remarques sur deux types particuliers d'instructions:

cw := ca ca:=cw

en effet la case w ne peut contenir que deux valeurs différentes soit 0 soit 1. Ces instructions n'ont d'autre intérêt que de mettre en réserve puis récupérer le contenu de w. Sinon, transférer en w une valeur quelconque de a obligerait à une convention spécifique, sauf dans le cas d'une représentation binaire de K en mémoire où seul le bit de poids le plus faible serait alors recopié.

QUELQUES EXEMPLES.

On se pose la question de savoir comment faire apparaître une constante différente de zéro, dans une case, par exemple: mettre 3 dans la case 5. Dans le jeu qu'on s'est donné, il n'existe pas d'instruction qui fasse cela tout de suite. On cherche donc un algorithme qui ait cette propriété. La solution est simple:

début
c5 := 0
c5 := c5 +$_K$ 1
c5 := c5 +$_K$ 1
c5 := c5 +$_K$ 1
fin

On remarque toutefois que ceci fonctionne si K>3. Faire apparaître une constante qui serait plus grande que la base K ne présente pas davantage de difficulté. Par exemple supposons que K=7. Je veux écrire la valeur 9 (base 10) dans la case 4. Il faut bien convenir que la valeur ne peut tenir dans une seule case, car elle s'écrit 12 (base 7),j'en prends donc au moins deux. Admettons 3 et 4. Je décide de placer le 1 en case 3 et le 2 en case 4:

début
c3 := 0
c4 := 0
c4 := c4 +$_K$ 1
c4 := c4 +$_K$ 1
c3 := c3 +$_K$ 1
fin

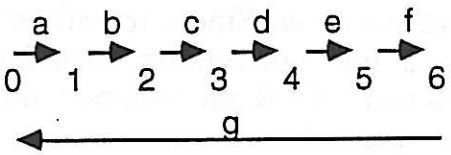
Ces deux exemples montrent bien qu'il est toujours possible de construire un algorithme capable de faire apparaître n'importe quelle valeur en mémoire.

La soustraction du 1.

Si l'on observe notre jeu d'instructions on s'aperçoit que finalement une seule d'entre elles est susceptible de faire évoluer une valeur, comme j'en ai déjà fait la remarque plus haut:

cx := cx +$_K$ 1

En prenant un cas particulier on va réfléchir sur l'action que peut présenter cette opération qui s'applique sur les diverses valeurs possibles:

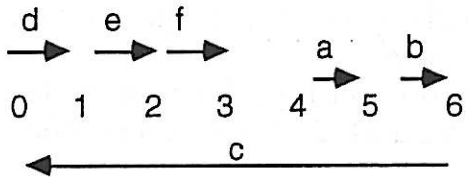


Je m'intéresse à une case quelconque x , et si K=7 j'ai, ci-dessus, la liste des seules valeurs qu'on peut rencontrer dans x. De plus, sont résumées dans ce schéma les diverses actions de notre addition de 1. Les applications de cette opération que j'ai appelées a, b, c, d, e, f, permettent toutes d'obtenir le successeur de la valeur initiale. Mais alors que se passe-t-il lors de l'application g? De la même manière que dans une numération à base K, le successeur s'écrit 10, mais ici le 1 saute automatiquement en case w, grâce au mécanisme de la machine, et c'est le 0 qui succède au 6. Ce processus est

valable bien entendu quel que soit K. En toute généralité, le successeur de l'élément K-1 sera toujours 0, le 1 du débordement apparaissant dans la case w.

Donc, et ceci est très important, dans notre machine nous traitons des représentations de nombres et non pas les nombres eux-mêmes. En conséquence les algorithmes qui permettent de calculer dessus, sont ceux qui sont issus du choix de la représentation.

Sur un exemple on va voir comment avec seulement une addition on fait apparaître la soustraction.



Si je pars d'un élément quelconque, par exemple 4, la suite des opérations: a, b, c, d, e, f m'amène à l'élément qui précède: le 3. Ceci revient en fait à additionner 6 fois le 1. Plus généralement on ajoute K-1 fois 1. Donc on ajoute K-1. On s'aperçoit ainsi que, quel que soit l'élément dont on part, lui ajouter K-1 revient en fait à lui retrancher 1. Bien entendu en arithmétique modulaire. Ceci montre bien que le prédécesseur du 0 est le 6. Plus généralement le prédécesseur du 0 est le K-1.

ca := ca -$_K$ 1

Cet algorithme s'écrit de plusieurs façons, la plus simple, à partir du moment où on a défini la machine, c'est-à-dire qu'on connaît K, par exemple K=7 prend la forme:

début
R1 ca := ca +$_K$ 1
R2 ca := ca +$_K$ 1
R3 ca := ca +$_K$ 1
R4 ca := ca +$_K$ 1
R5 ca := ca +$_K$ 1
R6 ca := ca +$_K$ 1
fin

Cases	w	a	Règle
	x	4	R1
	0	5	R2
	0	6	R3
	1	0	R4
	0	1	R5
	0	2	R6
	0	3	

Le tableau de droite donne un exemple de déroulement. Sur la première ligne on a n'importe quoi en w, et par exemple 4 dans la case a. L'application de la règle R1 donne 5 en a et 0 en w. On constate que l'application de R3 provoque un débordement, et après l'application de R6, dernière règle, on obtient le résultat: 3.
On se propose alors de chercher un algorithme qui agisse indépendamment de la valeur de K. Il s'agit en fait de trouver un moyen d'ajouter K-1 quel que soit ce K.

On peut se servir du débordement. Ajouter K-1 signifie ajouter K-1 fois 1. Ceci est possible en prenant par exemple une case de plus dans laquelle on fait apparaître une valeur de référence, chaque fois que j'ajoute 1 dans la case de référence j'ajoute également 1 dans la case a. Il est clair que si je pars de la valeur 1 dans la case de référence, un débordement y apparaît après addition de K-1 et en parallèle j'aurai ajouté K-1 dans a. Il suffit d'arrêter là le calcul. Cela donne l'algorithme qui suit. Je choisis la case 0 pour jouer le rôle de la case de référence. De la sorte, la case a peut être n'importe quelle case sauf bien sur cette case 0.
Pour un essai je choisis K=5, ce qui me donne droit aux chiffres 0, 1, 2, 3, et 4.
Les deux premières instructions donnent la valeur de référence au départ, soit 1.
Ensuite la boucle ajoute 1 à chaque tour en case a et 0, et on contrôle le débordement sur W.
Chaque instruction du type C3 modifie le contenu de w, c'est pourquoi l'addition sur la case 0 doit venir après celle qui agit sur la case a.

```
MU : début
R1        c0 := 0
R2        c0 := c0 +K 1
R3  E :   ca := ca +K 1
R4        c0 := c0 +K 1
R5        si cw = 0 vers E
      fin
```

Cases	w	0	a	Règle
	x	x	3	R1
		0		R2
	0	1		R3
	0		4	R4
	0	2		R5
				R3
	1		0	R4
	0	3		R5
				R3
	0		1	R4
	0	4		R5
				R3
	0		2	R4
	1	0		R5

Comme on le constate, la dernière application de R5 renvoie sur fin ce qui achève le calcul, et nous laisse avec le bon résultat dans a. Ce déroulement pour cette valeur de K pourrait être repris pour n'importe quelle valeur du modulo. Cela met en évidence le degré de généralité de cette forme d'algorithme. Il s'agit bien d'une forme d'algorithme car les instructions R2, R3, R4 portent la marque du K. Changer K revient à réécrire l'algorithme avec les nouvelles formes de R2, R3, R4 sans retrancher ni ajouter une ligne.
Je dispose désormais d'un nouveau moyen de calcul. Un algorithme me permet de soustraire 1. Je vais essayer de calculer l'addition des contenus de deux cases dans une troisième.

L'addition.

$$cm := cn +_K cp$$

Les notations m, n et p représentent des numéros quelconques de cases. La première question que je me pose c'est: le résultat tient-il dans une seule case ?
A question question et demi: quelles sont les plus grandes valeurs qu'on puisse rencontrer dans n et m ? réponse: K-1. Donc le plus grand résultat possible c'est 2K-2. Or il suffit d'observer que 2K-2= (K)+(K-2). Le tout est d'interpréter cette forme.
N'oublions pas que dans nos cases nous n'aurons jamais que des représentations de nombres. Ainsi si je considère que chaque case peut contenir un symbole de chiffre, bien évidemment je me place dans une numération à base K. Il devient évident que la représentation de la valeur K s'écrit 10, cela explique que 2K-2 s'écrive alors: 1 suivi de (K-2) qui est la somme de 10 et de K-2, lequel K-2 lui, tient dans une seule case.
En conséquence, quel que soit le résultat de l'addition, la case w plus une autre case suffisent pour le noter .
Le principe de l'algorithme de l'addition est simple, je transfère le contenu de n par exemple, dans m puis j'ajouterai 1 dans m chaque fois que je pourrai retrancher 1 du contenu de p.

```
ADD :   début
R1      cm := cn
R2      c1 := 0
R3  A : si cp = 0 vers FIN
R4
        cp := cp -K 1
R5      cm := cm +K 1
R6      si cw = 0 vers A
R7      c1 := cw
R8      vers A
R9  FIN : cw := c1
        fin
```

Cases	w	0	1	m	n	p	Règle
	x	x	x	x	4	2	R1
				4			R2
			0				R3
							R4
	x	x				1	R5
	1			0			R6
							R7
			1				R8
							R3
							R4
						0	R5
	0			1			R6
							R3
							R9
	1						fin

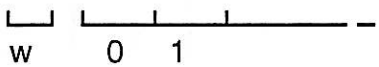
Pour faire cet exemple de déroulement, j'ai choisi K=5. Mais on constate que cet algorithme fonctionne quel que soit la valeur de K.
Il faut remarquer que la ligne R4 n'est qu'une écriture abrégée qui remplace en fait le texte complet de l'algorithme ca := ca $-_K$ 1 réécrit pour la case p. A partir de là je vais faire quelques remarques qui vont servir à orienter nos prochaines investigations. Les deux algorithmes que nous venons d'écrire sont simples, en eux-mêmes ils ne nous apprendront rien de plus. Par contre, l'observation de ce qu'il a fallu faire pour les réaliser, ainsi que la gesticulation nécessaire pour les utiliser peut alimenter notre réflexion.

L'algorithme MU utilise une case de manœuvre, j'ai choisi la case 0 car il en fallait bien une. Mais l'algorithme ADD inclut MU, en dehors de celui-ci il est évident qu'il est interdit d'utiliser encore 0 sous peine de mélanges de calculs donc de fautes. ADD a également besoin d'une case j'ai donc pris 1, qui sert à noter le débordement s'il y en a un. C'est le cas dans l'exemple traité.

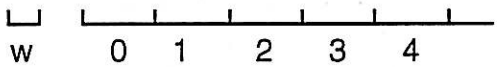
On remarque également que ADD détruit l'un des facteurs qui lui sont fournis ce qui est à éviter, car dans un calcul il est toujours possible d'avoir besoin plusieurs fois d'une même valeur.

Autre remarque, chaque fois que j'utilise MU je suis obligé de le réécrire en remplaçant le numéro de la case sur laquelle il calcule par celui de la case que j'ai besoin de décrémenter. A partir de là je vais systématiser l'écriture de ces algorithmes afin de pallier au mieux aux défauts soulevés.
D'abord quelques observations sur le rôle joué par les variables dans nos algorithmes. Dans le MU, la case nommée **a** contient une valeur qui existe avant application de cet algorithme, puis contient une valeur qu'il a calculée et qui doit subsister après son application. Par opposition, la case 0 elle, ne sert que pendant son calcul; peu importe son contenu avant application, et après application le contenu en a perdu toute importance. A charge pour l'algorithme d'en contrôler et utiliser ce contenu. Je dirai donc que **a** est le paramètre de MU et que 0 en est une variable locale. Ici, **w** en est l'autre.

Quand on applique le même raisonnement sur l'addition, il apparaît quatre paramètres: m, n, p et w et une variable locale 1. Une différence surgit alors dans les paramètres: il y a deux données ce sont les contenus de n et p, et deux résultats qui sont les contenus de m et w. Pour MU la case a jouait les deux rôles.
Je reprends alors l'écriture de ces deux algorithmes en détachant bien les rôles de chacun de ces types de variables.



début
c1 := **ca**
c0 := 0
c0 := c0 $+_K$ 1
E : c1 := c1 $+_K$ 1
c0 := c0 $+_K$ 1
si cw = 0 vers E
ca := c1
fin



début
c2 := **cn**
c3 := **cp**
c4 := 0
A : si c3 = 0 vers F
c3 := c3 $-_K$ 1
c2 := c2 $+_K$ 1
si cw = 0 vers A
c4 := cw
vers A
F : **cm** := c2
cw := c4
fin

Il est vivement conseillé de faire tourner ces algorithmes sur de petits exemples comme on l'a déja fait sur les cas précédents, en choisissant une valeur de K petite. Ce qui est suffisant pour montrer, de proche en proche que ça marche pour toute valeur de K.
Quelques remarques s'imposent. Dans cette nouvelle version, on voit clairement que les paramètres, notés en gras n'apparaissent plus qu'en tête et en fin d'algorithme, les données en tête, et les résultats à la fin. Fait encore plus important, aucun paramètre n'est plus détruit. Les variables locales occupent davantage de place. Tout se paye.

Mais notre réflexion va nous amener à chercher le moyen de récupérer la place des variables locales. Pour l'instant, la construction de l'addition interdit de récupérer la place utilisée par le MU.

La soustraction.

$cm := cn -_{K} cp$

Je suppose d'abord que cp < cn et on verra après ce qui se passe dans le cas contraire. Le principe de l'algorithme est simple: si je peux retrancher 1 dans l'une des données, alors je retranche 1 dans l'autre. En répétant cette opération jusqu'à épuisement de la première donnée, la valeur qui reste dans le deuxième opérande est le résultat. J'utilise les cases 2 et 3 comme variables locales.

début
R1 c2 := cn
R2 c3 := cp
R3 S : si c3 = 0 vers FS
R4 $c3 := c3 -_{K} 1$
R5 $c2 := c2 -_{K} 1$
R6 vers S
R7 FS : cm := c2
fin

w	2	3	m	n	p	Règles
x	x	x	x	4	2	R1
	4					R2
		2				R3
						R4
		1				R5
	3					R6
						R3
						R4
		0				R5
	2					R6
						R3
						R7
			2			

On est tenté de se demander alors ce qui peut se passer si cp > cn. Il suffit de dérouler un exemple comme on vient de le faire pour voir. Ainsi prenons K=9, puis essayons de calculer cm si cn=3 et cp=5. Les séquences de soustraction vont donner:

3 5
2 4
1 3
0 2
8 1
7 0

Il est évident qu'on s'attend au résultat -2, mais sous une telle forme il ne peut apparaître dans une case. On peut toujours essayer de voir ce que donne (7+2) modulo 9. Et là on constate qu'on obtient 10. Donc 7 est bien la représentation du -2 en modulo 9, car seul le zéro reste dans la case. Cette remarque va servir. On peut utiliser directement un tel résultat, car son addition à un contenu de case quelconque agira bien comme un -2. Mais alors comment différencier ce 7 d'un 7 simple valeur positive. Il est possible d'utiliser de toute évidence la case w. On choisit alors une convention, par exemple si le résultat est positif on sort de l'algorithme avec cw=0 et s'il est négatif complémenté à K, on sort avec w=1.

Il faut, à la sortie de l'algorithme, sonder le contenu de w avant d'utiliser le résultat. On conseille toujours aussi vivement au lecteur de réécrire la soustraction en rajoutant ce qu'il faut pour que w puisse ainsi évoluer. On remarque que c'est lorsque le contenu de la case 2 s'annule avant celui de 3 qu'on a un résultat négatif, et aussi que le résultat n'est pas toujours négatif.

Multiplication.

$cm := cn *_K cp$

Combien de cases faut-il pour noter le produit de deux cases? Question préliminaire. Le maximum pour cn et pour cp est K-1. Le produit des deux est (K-1) * (K-1) c'est-à-dire:

$K^2 - 2K + 1$, pour évaluer la place occupée, il ne faut pas oublier qu'un contenu de mémoire est comparable à une numération à base K. Ainsi, en deux cases le plus grand nombre qu'on puisse écrire se représente par: $K^1(K-1) + K^0(K-1)$, soit

$K^2 - K + (K-1)$ ou encore $K^2 - 1$. Et là on constate que $K^2 - 1$ pour toutes valeurs de K est plus grand que $K^2 - 2K + 1$, sauf pour K=1 pour lequel il y a égalité et K=0 qui n'a pas de sens.

Pour K=2 on peut noter au maximum 4-1=3, alors qu'on a besoin seulement de 4-4+1=1.

Donc notre résultat exige deux cases pour être noté, mais deux cases suffisent largement.

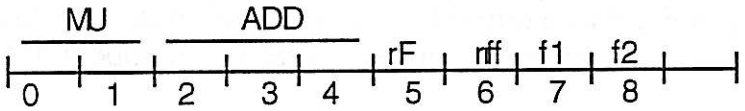
En utilisant ce dont on dispose, la technique de calcul peut être celle-ci: chaque fois que je peux soustraire 1 à l'un des facteurs, j'ajoute l'autre facteur au résultat. Le résultat, je le dispose en poids faible sur une case et poids fort sur une autre case. L'addition du deuxième facteur se fait sur le poids faible, et chaque fois que cette addition provoque un débordement j'ajoute 1 sur le poids fort. Comme on l'a vu sur le poids fort il ne peut jamais y avoir débordement.

Par exemple pour K=7 si je veux faire le produit de 3 par 4, j'obtiens la suite des calculs:

00	3	4
03		3
06		2
12		1
15		0

Il faut mettre le résultat à zéro au préalable, résultat que je note en deux cases qui se jouxtent, le poids fort se trouvant dans la case d'adresse la plus faible.

La configuration se présente ainsi:



début
c7 := cn
c8 := cp
c5 := 0
c6 := 0
M : si c8 = 0 vers FM
c8 := c8 -K 1
c6 := c6 +K c7
si cw = 0 vers M
c5 := c5 +K 1
vers M
FM : cm := c5
c(m+1) := c6
fin

Pour la multiplication j'ai ainsi besoin de l'addition et du moins-un, il me faut donc protéger les configurations de ces deux algorithmes, les cases 0 et 1 puis 2, 3, 4 me sont interdites je placerai donc le résultat en 5 et 6 et les deux facteurs en 7 et 8.

Je n'ai pas construit la division car c'est un exercice que je laisse à l'initiative du lecteur. Une réflexion peut être menée à son propos sur la représentation approchée des nombres fractionnaires, car il faut bien arrêter à un moment ou à un autre une division dont le résultat non seulement n'est pas entier mais encore menace d'exiger une suite illimitée de décimales.

La conclusion de ce travail se résume ainsi, la machine de Nolin permet de réaliser toute l'arithmétique. Ce sera pour nous le système formel de référence auquel on se rapportera un peu comme à une sorte d'étalon sémantique. D'autres préféreront les descriptions qu'on trouve en logique formelle des fonctions récursives, d'autres encore préféreront la machine de Turing. De toute façon on montre l'équivalence sémantique entre tous ces systèmes, sans oublier les systèmes de Post.

Pour nous la machine de Nolin rassemble deux qualités: elle est de structure simple, d'un coup d'œil on peut embrasser ses propriétés, et elle est proche à la fois de ces machines réelles qu'on appelle les ordinateurs.

L'existence de cette forme d'arithmétique étant ainsi prouvée il faut aller plus loin, et c'est l'observation des inconvénients rencontrés qui va nous y mener. La réflexion porte tour à tour sur l'algorithme et sur la configuration. Sur l'algorithme on constate deux sortes d'obligations pénibles. Observons nos deux algorithmes: l'addition et la multiplication insèrent le moins-un, l'une pour agir sur la case 3, l'autre sur la case 8. Là où j'ai encadré le moins-un, il faut donc remplacer cette notation par le texte de cet algorithme dont on a enlevé les début, fin mais qu'il faut en outre modifier en changeant le numéro de la case paramètre.

Quant à la configuration, on conçoit qu'il est impossible de récupérer aisément la place des variables locales des algorithmes dont on se sert, ainsi, dans la multiplication les cases de 0 à 4 demeurent intouchables.

Je vais donc essayer de construire un système dans lequel ces inconvénients disparaissent. Ce ne sera pas possible en un seul coup. En un premier temps on va trouver un moyen de récupérer la place des variables locales et insérer ou réécrire les algorithmes sans toucher à leur forme.

Je définis à cet usage une nouvelle machine dotée d'un nouveau langage. Auparavant, je vais compléter la structure de la machine de Nolin, en complétant, ce que j'avais annoncé et qui est indispensable à la puissance de la machine.

CHAPITRE 3. COMPLEMENTS A LA MACHINE DE NOLIN,.

Edmond Bianco

PUISSANCE D'UNE MACHINE.

Le mot 'puissance' sert à quantité d'usages dans les divers domaines de la connaissance, aussi quelques explications sont-elles nécessaires pour le cas présent. Il s'agit dans la présente théorie d'un moyen de comparaison entre les propriétés de divers modèles. C'est ainsi que je dirai, par exemple que le modèle 'machine de Turing' a même puissance que le modèle 'machine de Nolin' dans la mesure où je suis à même de montrer qu'il n'existe pas d'algorithme de l'un d'entre eux qui n'ait pas au moins un algorithme équivalent dans l'autre modèle.
En général on montre d'ailleurs que pour deux modèles "équipuissants", chercher l'équivalent d'un algorithme de l'un, revient à en trouver dans l'autre, non pas un seul mais en fait une classe qu'on ne peut réduire qu'en rajoutant des critères. Ceci n'est qu'une illustration de la richesse du champ informatique.
La puissance n'est pas ici définie par rapport au temps comme une unité mécanique, mais par rapport à l'absolu : elle peut ou ne peut pas faire tel calcul.
Dans le champ du calculable on peut encore subdiviser. En effet un problème peut être calculable sans être effectivement calculable. Par exemple la division est toujours effectivement calculable sauf dans une division par 0, qui est calculable sans être effectivement calculable.
Dans effectivement calculable, il existe des calculs qui ne sont pas réellement calculables. En exemple le calcul:

$$10^{30} / 1$$

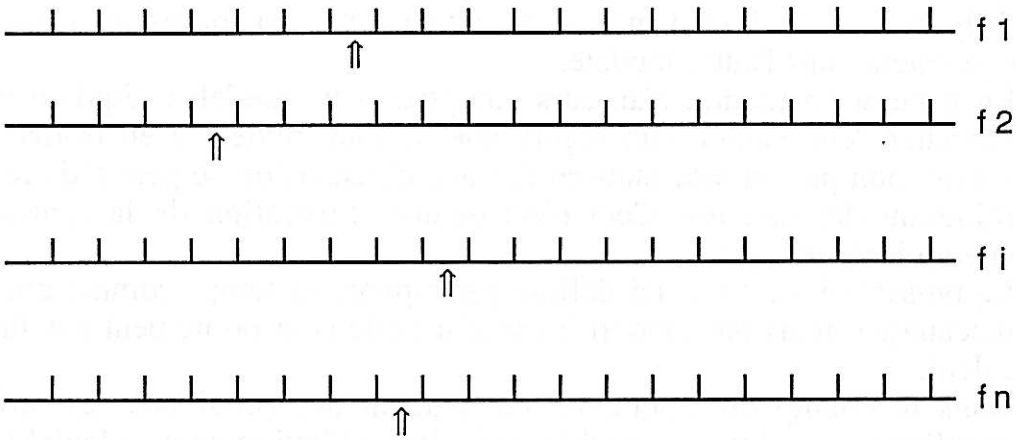
dont le résultat évident est 10^{30} mais qui, calculé par la méthode des soustractions successives demanderait de l'ordre de 3.10^{13} millénaires pour être réalisé sur un ordinateur normal.
En conclusion, pour nous la puissance d'une machine représente sa capacité à réaliser un type de calcul déterminé. De cette façon, on imagine que la machine de Nolin non complétée par un moyen adéquat ne pourrait absolument pas traiter des problèmes qui exigeraient une information trop volumineuse pour tenir sur sa mémoire centrale, dont le maximum qu'elle puisse contenir, nous l'avons vu est de $2. K^k$.

FILES EXTERNES.

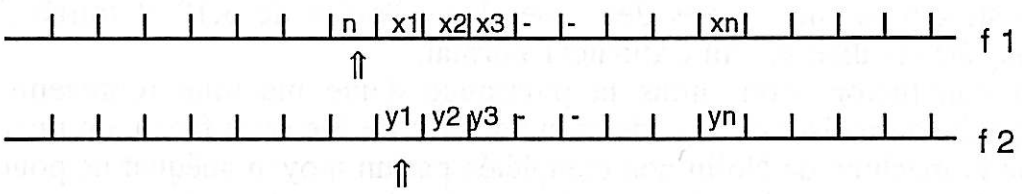
On adjoint à la mémoire centrale un jeu de files externes. Une file externe est constituée d'une suite illimitée de cases. Chacune de ces cases est identique aux cases de la mémoire centrale (à l'exception de w, bien

entendu). Une sorte de tête de lecture-écriture est calée devant l'une de ces cases. Il y a une tête par file externe.
Un ensemble d'instructions permet de travailler avec ces files, en fait d'échanger de l'information entre l'une d'entre elles, au choix, et la mémoire centrale.

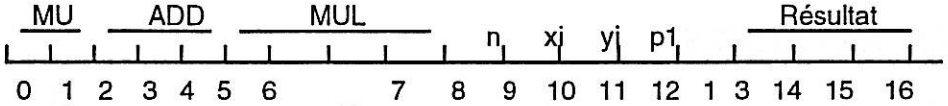
ca := cfi	Envoie en case "a" le contenu de la case de file "i" marquée par la tête de lecture.
cfi := ca	Envoie sur la file un contenu de case de la mémoire centrale.
Av fi	Déplace la tête de lecture-écriture de la file i d'une case à droite.
Ar fi	Déplace la tête à gauche.



Pour illustrer l'utilisation de ces files on peut traiter un petit exemple. On va calculer un produit scalaire de deux vecteurs. Je vais supposer placées les composantes x1, x2, x3, ... xn du premier vecteur dans la file f1, et les composantes y1, y2, y3, ... yn dans la file f2. En tête de l'enregistrement en f1 se trouve la valeur n, dimension des deux vecteurs.



J'utilise bien entendu les algorithmes de l'addition, de la multiplication et du moins-un. Donc, je réserve les cases qui leur correspondent.



Je suppose pour simplifier que le résultat tient en trois cases: 14, 15 et 16. En 12 et 13 se placent les produits de paires de composantes.

```
début
        c14 := 0
        c15 := 0
        c16 := 0
        c9 := cf1
        Av f1
D :     si c9 =0 vers FPS
        c10 := cf1
        c11 := cf2
        ┌──────────────────┐
        │ c12 := c10 *K c11 │
        └──────────────────┘
        ┌──────────────────┐
        │ c16 := c16 +K c12 │
        └──────────────────┘
        si cw =0 vers S1
        c15 := c15 +K 1
        si cw =0 vers S1
        c14 := c14 +K 1
S1 :
        ┌──────────────────┐
        │ c9 := c9 -K 1     │
        └──────────────────┘
S2:     Av f1
        Av f2
        vers D
FPS :   fin
```

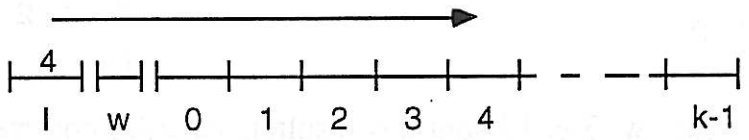
Par ailleurs, il est bon de connaître certains résultats théoriques qui complètent les propriétés des files. D'abord, on montre qu'une seule file illimitée est aussi puissante qu'un nombre quelconque de files illimitées. Et enfin qu'une demi-file, c'est-à-dire une file bornée d'un côté, est aussi puissante qu'une file illimitée dans les deux sens. Par "puissant" ou "puissante", j'entends que quelle que soit l'information notée d'un côté sur une sorte de file, elle peut l'être également notée sur l'autre sorte de file, et y être retrouvée sans ambiguïté.

CHAPITRE 4. LA MACHINE FORMELLE,

Edmond Bianco

LA PROCEDURE FORMELLE.

La procédure formelle c'est le langage, le moyen d'expression des algorithmes de la machine formelle. Pour le définir il me faut rajouter un tout petit mécanisme à la machine. C'est une case supplémentaire, qu'on désigne par **I**, et unique comme la case w. Pour l'instant je dis que le contenu de cette case est de même nature que celui des cases qui portent les noms 0, 1, 2, etc. (k-1). Mais il est interprété autrement. Je mets en relation le contenu de **I** avec le nom d'une case. Par exemple si le contenu de **I** est 21, cela désigne la case 21, s'il est 197, cela désigne la case 197.
La case désignée par le contenu de I est ainsi placée à l'origine d'une configuration translatable, puisqu'on peut recalculer le contenu de **I**. Et tout se passe comme si les cases à partir de cette origine étaient renumérotées: 0 pour l'origine, 1 pour la suivante, puis 2, 3, etc. C'est cette renumérotation relative qui va servir pour la description des opérandes.
La mémoire se représente ainsi:



Dans cet exemple $[_I = 4$, donc l'origine est sur la case 4.

Je reprends les mêmes opérations que celles de la machine de Nolin et j'intègre les algorithmes construits, dans ma liste d'opérations élémentaires. Le langage prend alors la structure suivante:

x := y	x	représente	$[_{a,b}$, $[_a$, $[_I$, $[_w$;
x := y **A** y	y	représente	x ou Const;
si y **R** y vers E	**A**	représente	+ , - , * ,/ ;
vers F	**R**	représente	≠ , = , > , < , ≤ , ≥ ;

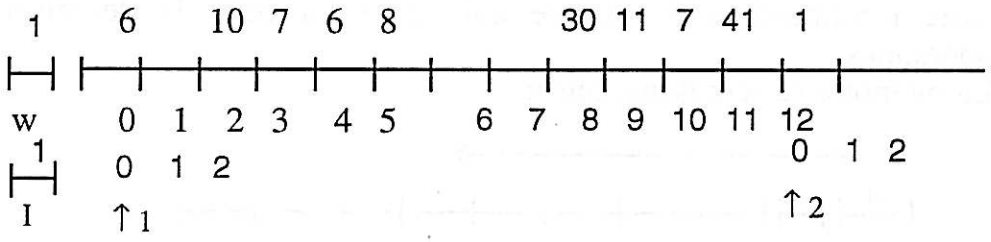
J'admets désormais, puisque nous avons montré leur existence, que les quatre opérateurs arithmétiques +, -, *, /, font partie des opérations élémentaires. Pour la même raison, j'intègre également les six opérateurs de relation dans les conditions.
La différence entre un opérande de forme x et de forme y tient dans la possibilité d'une constante à droite du :=. On peut ainsi se donner quelques exemples d'instructions dont j'expliciterai la signification après.

E1 $[_3 := [_{0,4}$ E2 $[_{2,1} := [_{3,2} + [_{5,0}$

E3 $[_W := [_{2,2}$ E4 si $[_3 \geq [_{4;2}$ vers HJ

E5 si $21 \neq [_8$ vers PL E6 $[_I := [_I + 11$

L'opérande à un seul indice tel que $[_3$ signifie qu'on s'intéresse au contenu de la case qui est comptée numéro 3 par rapport à l'origine définie par le contenu de I. Désormais je ne me préoccuperai plus du numéro réel de la case, cela c'est l'affaire de la case I. Par contre je superpose aux vrais numéros de case une nouvelle numérotation qui est relative à l'origine désignée par le $[_I$, la case ainsi repérée porte le numéro 0, la suivante à droite le numéro 1, puis le 2, etc. C'est à cette numérotation que je m'intéresse dans l'écriture des instructions.
L'interprétation d'un opérande à deux indices, de la forme $[_{0,4}$ comporte une étape de plus dans le calcul de l'adresse. Le premier indice: 0 signifie qu'il faut prendre le contenu de la case 0, considéré comme une adresse relative à l'origine définie par: $[_I$ et cette adresse est elle-même corrigée par le deuxième indice, ici 4. A titre d'exemple je me donne une configuration ainsi choisie:



Les contenus des cases w, 3 et 11 sont des résultats calculés comme on va le voir.
D'abord l'instruction E1:l'opérande de droite se calcule ainsi, dans la case 0 je trouve 6, je lui ajoute le 4 du deuxième indice, soit 10 et je considère le contenu de la case 10 en l'occurrence 7. C'est cette valeur 7 que je transfère dans la case 3, comme l'indique l'opérande de gauche.
Ensuite l'instruction E2, le premier opérande à droite du := prend son adresse en case 3, soit 7, valeur à laquelle j'ajoute le 2 du second indice et cela me renvoie à la case 9 dont le contenu 11 est la valeur désignée. Le second opérande prend son adresse en case 5, soit 8, corrigée de 0, la valeur désignée est ici 30, en case 8. La somme de 11 et de 30, soit 41 est envoyée à l'adresse précisée par l'opérande à gauche du := c'est-à-dire celui qui prend son adresse en case 2, soit 10, corrigée de 1 soit 11. C'est en 11 que se retrouve 41, quand on se place dans le cas où K >41.
L'instruction E3, qui prend en case 2 l'adresse 10, à laquelle s'ajoute le deuxième indice 2, ce qui donne l'adresse 12, saisit le 1 qui est en case 12 et le place en case w.
Pour l'instruction E4, on constate que le contenu de la case 3, soit 7 n'est pas plus grand ou égal à 30, contenu de la case 8 dont le nom est obtenu par l'addition de 6, contenu de la case 4 et de l'indice 2. En l'occurrence cette instruction ne renvoie pas à l'étiquette HJ.

L'instruction E5 renvoie à l'étiquette PL car 21 est effectivement différent de 30 qui est le contenu de la case 8.
Enfin l'instruction E6 rajoute 11 au contenu de I ce qui a pour conséquence de faire passer l'origine de 1 à 2 . Continuer à calculer reviendrait alors à condidérer comme case 0 l'ancienne case 12, comme case 1 l'ancienne case 13 etc.
Pour l'interprétation des instructions E4 et E5, je conserve la même définition de la notion d'algorithme que dans la machine de Nolin.

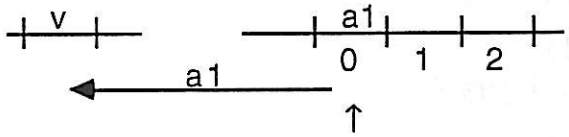
NOTION D'INSERTION.

Gestion de la mémoire.

Avec ce nouvel outil je vais essayer de résoudre au moins une partie des problèmes posés à propos de la machine de Nolin. Pour la configuration, comment récupérer la place tenue par les variables locales, pour l'algorithme, comment ne plus avoir à en modifier le texte avant de le reproduire là où on l'insère, et si possible ne plus avoir à le dupliquer chaque fois qu'on l'insère. On verra que pour résoudre ce dernier problème, une étape supplémentaire devra être franchie.
Pour illustrer tout cela on va reprendre les algorithmes bien connus désormais de la multiplication, de l'addition, et du moins-un. Réécris tels quels en procédure formelle. Ce qu'on va dire à propos de ces trois algorithmes sera évidemment valable pour n'importe quel autre jeu d'algorithmes.

Le moins-un.

D'abord la configuration. L'algorithme travaille avec un seul paramètre, je le place en case 0, en case 1 et 2 je place la variable de référence et la valeur à incrémenter.



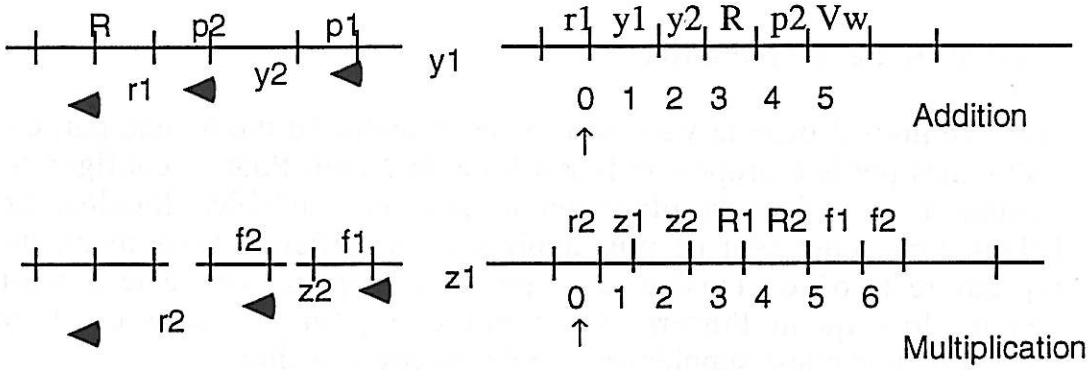
La flèche verticale indique l'origine, laquelle est fournie par le contenu de I, et v est la valeur à décrémenter. L'algorithme se décrit ainsi en faisant comme hypothèse qu'avant de l'appliquer, la case 0 contient l'adresse du paramètre calculée par rapport à l'origine de sa configuration, ici c'est a1 qui permet d'atteindre v.

```
début
      [2 := [0,0
      [1 := 0
      [1 := [1 +K 1
MU :  [2 := [2 +K 1
      [1 := [1 +K 1
      si [w = 0 vers MU
      [0,0 := [2
fin
```

Cet algorithme pourrait être écrit de manière plus condensée avec les moyens de la procédure formelle, mais peu importe, il ne s'agit plus du calcul de moins un, cela on sait désormais que ça existe, on veut maintenant mettre en évidence des propriétés d'insertion, tout algorithme fait l'affaire, alors pourquoi pas celui-là puisque nous le connaissons bien.

L'addition et la multiplication.

Dans ce cas trois paramètres sont nécessaires, j'utilise à leur usage les cases 0, 1 et 2, les cases 3 et 4 vont servir pour le résultat et la donnée à détruire, la case 5 pour sauvegarder le contenu de w, pour l'addition. Pour la multiplication, il y a également trois paramètres, placés en 0, 1 et 2, les cases 3 et 4 servent au résultat, et 5 et 6 aux deux facteurs. Le schéma des deux configurations est organisé en conséquence.



Désormais je ne préciserai plus le modulo-K dans les instructions puisque nous savons que toutes les opérations se réalisent ainsi.

début
    [3 := [1,0
    [4 := [2,0
    [5 := 0
A : si [4 = 0 vers FA
    [4 := [4 - 1
    [3 := [3 + 1
    si [w = 0 vers A
    [5 := [w
    vers A
FA : [0,0 := [3
    [w := [5
fin

début
    [5 := [1,0
    [6 := [2,0
    [3 := 0
    [4 := 0
M : si [6 = 0 vers FM
    [6 := [6 - 1
    [4 := [4 + [5
    si [w = 0 vers M
    [3 := [3 + 1
    vers M
FM : [0,0 := [3
    [0,1 := [4
fin

Je vais considérer les instructions en gras comme des programmes à insérer. L'intérêt de procéder ainsi c'est de porter notre attention sur ce qui se passe pour réaliser effectivement une insertion. Insertion au sens journalistique: remplacer l'instruction en gras par le texte de l'algorithme qui lui correspond.
Je rappelle les hypothèses:

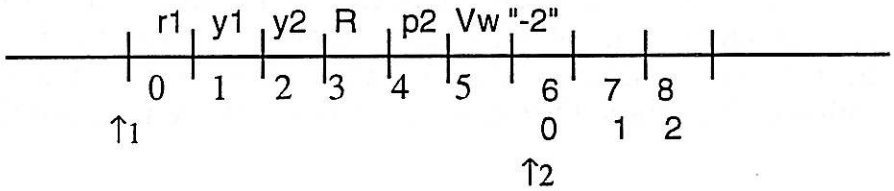
Pour que de tels algorithmes puissent fonctionner il faut et il suffit que le contenu de I indique la case origine de la configuration qui lui est ainsi préparée, les premières cases de cette configuration contiennent dans l'ordre, les paramètres effectifs, c'est-à-dire les adresses par rapport à cette origine des valeurs données et des valeurs résultats.

L'addition, par exemple a besoin de calculer un moins-un sur sa case 4. Que faut-il faire pour utiliser l'algorithme prévu pour cela?
Il faut lui fournir une configuration qui lui convienne. Où la mettre ? on peut décider arbitrairement de la créer à l'extrémité droite de la configuration de l'addition. Or, cette configuration qui occupe six cases laisse à sa droite la case 6, première case libre. C'est là qu'on va mettre le paramètre du moins-un. C'est cette case 6 qui va devenir l'origine de la configuration du moins-un. La valeur du paramètre, l'adresse de la quantité à décrémenter, doit être calculée par rapport à la case 6 pour atteindre la case 4 ; puisqu'on veut dans l'addition calculer :

$$[_4 := [_4 - 1$$

Si la case 6 est la nouvelle origine pour le moins-un, atteindre à l'ancienne case 4 exige de disposer d'une adresse qui ait la signification de "-2", or cette valeur se représente par K-2 avec $[_W := 1$, c'est donc la valeur qui doit être placée en case 6 avant de changer d'origine.
Dans l'algorithme de l'addition je remplace $[_4 := [_4 - 1$ par le texte suivant

L1	$[_6 := K-2$	Place l'adresse en case 6,
L2	$[_I := [_I + 6$	Change d'origine qui passe de ↑1 à ↑2,
	texte du moins-un	Texte recopié tel quel,
L3	$[_I := [_I - 6$	retour de l'origine à ↑1



Après déroulement de la ligne L2 l'origine est ↑2 la première instruction à dérouler dans le moins-un:

$$[_2 := [_{0,0}$$

place bien dans la case 2 la valeur dont l'adresse est en 0 et corrigée de 0 c'est-à-dire K-2 par rapport à...↑2, soit p2. Après déroulement de L3, l'origine revient en ↑1 ce qui libère les cases 6, 7 et 8, de nouveau disponibles.

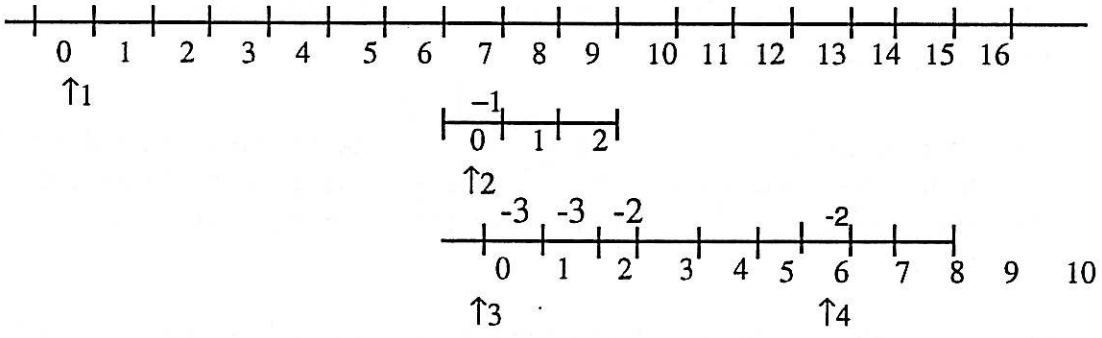
Observons alors le déroulement de la multiplication. Les deux expressions:

$$[_6 := [_6 - 1$$
$$[_4 := [_4 + [_5$$

sont remplacées par l'insertion successive du moins-un et de l'addition. Or, c'est la case 7 la première case disponible dans la configuration de la multiplication. C'est dans cette case que doit apparaître le paramètre effectif du moins-un, pour le compte de la case 6, soit K-1. Le travail du moins-un achevé, revenu à l'origine de la configuration de la multiplication, il faut préparer la configuration de l'addition qui comporte trois paramètres lesquels coïncident dans l'ordre avec les accès aux cases respectivement: 4, 4, et 5. Ce qui par rapport à la case 7 donne les adresses K-3, K-3, et K-2. Cela s'écrit:

```
[7  := K-1
[I  := [I + 7
Insérer le texte du moins-un
[I  := [I - 7
[7  := K-3
[8  := K-3
[9  := K-2
[I  := [I + 7
Insérer le texte de l'addition
[I  := [I - 7
```

Faire dérouler la multiplication a pour conséquence ce qu'on appellera une **gestion dynamique de la mémoire.** Cela se présente de la sorte: à la première rencontre du moins-un, en 7, 8, et 9 se crée sa configuration, l'origine vient en case 7. Lorsque le moins-un a terminé, l'origine revient au départ en ↑1. A la rencontre de l'addition, on crée sa configuration sur les cases 7 à 12. L'origine passe sur la case 7. L'addition à son tour qui fait appel au moins-un, est amenée, comme on l'a vu, à créér la configuration de ce dernier qui apparaît ainsi à la case 6 de la configuration de l'addition. Où? A la case 13 de celle de la multiplication.



L'insertion du moins-un dans la multiplication provoque la création de la configuration en ↑2, puis il y a retour en ↑1. L'insertion de l'addition à la suite provoque la création de la configuration en ↑3. Or dans l'addition on

fait appel au moins-un et à cette occasion se crée la configuration en ↑4, de là se fait le retour en ↑3 quand le moins-un a terminé, puis le retour sur ↑1 quand l'addition a terminé. On voit que ce processus se répète pour chaque tour dans la boucle M de la multiplication.

Cela montre bien qu'il y a réellement récupération de la place occupée par les variables locales de l'algorithme. En contrepartie et selon toutes apparences chaque configuration occupe davantage de place que dans le cas de la machine de Nolin. En effet il faut noter les paramètres en plus et nous verrons qu'il faudra noter encore bien d'autres choses pour parvenir à nos fins. Seulement il faut également considérer qu'en toute généralité, pour chaque variable on note le nom, et que ce nom unique correspond à un ensemble de valeurs qui peut être grand. Le rapport qui existe entre le nombre de noms et le nombre de valeurs est généralement faible, l'essentiel de la place est occupée par les valeurs donc la rentabilité d'un tel système demeure certaine.

A titre de conclusion, côté algorithme l'insertion n'est désormais plus qu'une simple recopie sans modification du texte et côté configuration la place occupée par les variables locales n'est plus nécessaire que pendant le déroulement de l'algorithme.

Je vais alors poser une dernière question afin de parachever ce travail:

« Comment faire pour ne plus avoir non plus à réécrire l'algorithme dans l'insertion ?»

VOUZZAVEDIBIZAR,

Les aventures de Savate Premier et son Ordinateur Chevelu.

Le temps des voyages.

Savate Premier avait un ami de trente ans. Trente ans n'étaient pas l'âge de l'ami, non, c'était celui de leur amitié.

Malgré les promesses de prompt rétablissement, promesses électorales, la santé des entreprises "Gaullicoques" mettaient beaucoup de temps à se rétablir. Aussi notre homme, homme d'action, décida-t-il de prendre le taureau par les cornes.

« Je vais aller exporter et vendre la Nouvelle "Gaullicoquie" à l'étranger ! ». S'écria-t-il dans un bel élan et avec un très beau mouvement du menton qu'il avait splendide.

Savate Premier, défavorisé au départ, après une courte mais brillante lutte l'avait emporté contre son ami de trente ans qui s'en était trouvé très affecté. Aussi notre héros, constatant ainsi une glorieuse victoire à la fois sur tous les fronts de la bataille politique, se sentit-il investi d'une mission surnaturelle, voire miraculeuse, pour la rénovation du Bon Vieux Pays.

Grand Général à nous deux !
Et il partit.

Il emmena dans ses bagages, quelques représentants chevronnés des plus grosses entreprises "Gaullicoquiennes". Tout ce Staff, pour parler le "new- européan" langage, faisait fort bel effet. Et fit un fort bel effet, quand il se présenta en ces républiques insolentes d'outre Atlantique. Et ce n'est pas parce que les sourires de ces cow-boys aux bottes encore pleines de foin, étaient quelque peu légers et désinvoltes que la charge ne fut pas une réussite. Puis le temps passa pendant lequel l'Ordinateur Chevelu fut brillant comme à l'accoutumée, les "Gaullicoques" toujours aussi remuants et fermés aux progrès nouveaux, à l'exception, bien sûr des loyaux affidés, et encore.

Mais l'idée continuait à tarauder Savate Premier. Il fallait aller rendre visite à l'autre puissante contrée. L'Empire du Soleil Levant. Rien que ce nom évoquait en lui d'immenses tas d'or irisés par les rayons de l'astre matinal. Il en frissonnait d'une joie intérieure mal contenue.

Et il partit.

Ce fut un magnifique bain de sourires énigmatiques, comme seuls les sourires asiatiques savent l'être. Ce qui n'empêcha pas l'un d'entre eux de tourner très légèrement ironique en osant demander pourquoi ce n'était pas le Ministre du commerce extérieur qui venait exposer ces questions. Notre homme alors de se rengorger et de faire un magnifique panégyrique de son Ordinateur Chevelu, en étalant tous ses mérites hélas pas toujours très bien reconnus en "Gaullicoquie".

Un autre sourire énigmatique se pencha alors vers une oreille du même tonneau, pour lui glisser que ces éloges passaient bien à Tokyo, alors qu'en "Gaullicoquie"... On frisa l'incident diplomatique. Un peu comme l'escarmouche qui l'avait légèrement agacé dans ce ridicule pays de l'Orient Moyen.

En bref c'est un véritable enthousiasme énigmatique qui ponctua la noble prestation du Grand Chef.

Nul n'est prophète en son pays.

Edmond Bianco

Université de Provence
Atelier de Reprographie
Centre Saint Charles
3, place Victor Hugo
F - 13331 Marseille Cedex 3