BULLETIN D'INFORMATIQUE APPROFONDIE ET APPLICATIONS

SCIENCE DE L'INFORMATION

N° 51 - DECEMBRE 1998

REDACTION

Université de Provence
Equipe Hermès. Case 33
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: (0)4 91 10 62 30
Télécopie : (0)4 91 50 91 10

DEPOSITAIRE

Université de Provence
Bibliothèque Vniversitaire
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: (0)4 91 62 44 16
Télécopie : (0)4 91 95 75 57

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

Publication trimestrielle, gratuite, de l'Université de Provence.

Edition 1999 ISSN 0291 - 5413

BULLETIN D'INFORMATIQUE APPROFONDIE ET APPLICATIONS

SCIENCE DE L'INFORMATION

N° 51 - DECEMBRE 1998

REDACTION

Université de Provence
Equipe Hermès. Case 33
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: (0)4 91 10 62 30
Télécopie : (0)4 91 50 91 10

DEPOSITAIRE

Université de Provence
Bibliothèque Vniversitaire
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: (0)4 91 62 44 16
Télécopie : (0)4 91 95 75 57

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

Publication trimestrielle, gratuite, de l'Université de Provence.

Edition 1999 ISSN 0291 - 5413

EDITORIAL,

De l'an mil à l'an deux mil.
Ou le crétinisme à travers les âges.
Ou l'informatique prétexte à l'imbécillité.

Une brève qui tombe: le doyen d'Echallens en Suisse, qui vient de fêter ses 105 ans, a été prié de se préparer à rejoindre les bancs de l'école primaire ... Le "programme informatique" négligeant le troisième chiffre lui attribuait 5 ans d'âge. Nouvelle amusante à la fois par son contenu et par le charabia journalistique qui pétille quand on parle d'informatique.

Une sottise du même calibre est à l'origine de la menace terrifiante qui pèse sur le prochain franchissement de millénaire. A l'approche de l'an mil, les moines parcouraient les rues en psalmodiant et en criant:

«Repentez vous, Dieu va vous fracasser ! »

ou quelque chose d'équivalent. Sur quels indices ces pitres sinistres s'étaient-ils basés pour annoncer la fin du monde? Ou plutôt comment cette terreur répandue pouvait-elle profiter aux petites affaires de l'église? Le tout sur un fond de psychose soigneusement entretenu.

A l'orée du troisième millénaire, ces vieilles choses sont un peu démodées, alors on a trouvé mieux. Un fait demeure, tout de même étrange, c'est le pouvoir mystérieux du zéro. il a fallu longtemps à la pensée humaine pour créer cette notion, somme toute, très abstraite. De quelle nature est donc le pouvoir magique des nombres qui ne comportent que des zéros à droite du premier chiffre? On se crée ainsi des sortes de bornes que l'on a visiblement beaucoup de mal à franchir. Cela n'est pourtant qu'une péripétie due au choix parfaitement arbitraire de la notation décimale. Si, pour des raisons de société, une toute autre base numérique avait été choisie, les bornes à barder de zéros eussent été d'une toute autre valeur.

On reste rêveur devant tant d'inconséquence.

Ou encore les voies du Seigneur, c'est-à-dire du pouvoir et de l'argent, restent impénétrables au vulgaire, c'est-à-dire au non initié.

Chaque seconde qui nous sépare du premier janvier 2000 est ponctuée d'un puissant coup de grosse caisse, pour bien nous faire sentir combien notre destinée fragile est liée au sort de "l'informatique". En d'autres termes combien profondément le sort des entreprises dépend de "l'informatique" toute puissante. Le nouveau Dieu tient prêtes ses foudres. Et ce n'est que grâce à d'importantes actions de grâces, et de contritions sincères, que l'on pourra, peut-être, éviter le pire.

Bien entendu toutes ces actions (de grâce, naturellement) forment le fonds de commerce d'une nouvelle industrie juteuse, mais chut ... respectons les arcanes.

Si les entreprises sont menacées, ce qui pourrait laisser de marbre la vile plèbe, il faut tout de même bien voir que l'Entreprise est devenue la marque de l'Aristocratie moderne. L'argument salvateur. Le sort de nous tous a été déclaré étroitement lié à l'entreprise, au profit, à l'investissement, bref, le cœur de la Nation bat dans l'entreprise.

(Oui, d'accord ... mais quid de l'Europe ? ...)

(Allons, allons, taisez-vous, ne mélangez pas tout !)

On peut alors raisonnablement se demander quelles sortes de catastrophes vont effectivement s'abattre sur nous.

Pour ma part j'en vois deux, d'abord l'une, venant de la Gauche, c'est-à-dire les Socialistes, spécialistes en la matière. On va nous concocter une nouvelle race d'impôt, d'abord assez limité, puis qui se révélera merveilleusement extensible par la suite, dans le genre de la CSG. Ensuite l'autre, préparée par la Droite qui fera tout pour juguler la liberté de la Presse, en commençant par les publications jugées licencieuses, afin de protéger au mieux "la famille". Il semble en effet tout à fait naturel de sauvegarder un "vivier" de futurs bons soldats, afin de pouvoir envoyer à travers le monde d'ardents défenseurs des intérêts du pétrole et du nucléaire. Et sur ce point gauche et droite sont bien d'accord.

Cette fin de siècle doublée d'une fin de millénaire, phénomène qui ne se sera jamais produit qu'environ seize millions de fois depuis que la terre semble exister, aura été marquée par quelques vilenies remarquables. L'une des plus belles, fut récemment commise par Tête à Gifles N°2, lorsqu'il a promis des papiers aux sans-papiers, à condition qu'ils se fassent recenser. Pour avoir le plaisir, les tenant sous sa patte de flic, de les expulser avec encore plus d'efficacité que Pasqua. C'est plus facile, et ça semble électoralement plus fructueux que d'essayer d'enrayer la fuite des capitaux dans les paradis fiscaux.

Allez savoir pourquoi se produisit en même temps un curieux phénomène: ayant confondu une banale opération, avec un charter vers l'au-delà, il fut moins une qu'on ne ramena point la dépouille faisandée de notre ami Tête à Gifles N°2, Dieu pourrait peut-être bien exister.

Je ne sais pas ce que sera le troisième millénaire, mystique comme le prétendait le crétin visionnaire de la cinquième, ou simplement encore plus commercialement libéral, c'est à dire encore plus ignoble, mais j'estime qu'il n'y a pas grand chose à regretter du second.

Edmond Bianco

CHAPITRE 5

LA MACHINE UNIVERSELLE

Edmond Bianco

Jusqu'à présent tout calcul se définissait avec un couple mécanique composé d'une mémoire et d'un algorithme adjoint convenablement câblé pour pouvoir faire évoluer le contenu de cette mémoire. Il fallait charger la mémoire par un moyen quelconque, puis lancer l'application de l'algorithme. Le calcul achevé, on lisait le résultat dans la mémoire. Ceci décrit un processus standard de matérialisation de calcul automatique.

L'inconvénient majeur de cette façon de faire apparaît clairement sur une simple remarque. Pour chaque calcul envisagé je suis obligé de câbler un algorithme. C'est le cas des petites calculatrices à main qui permettent d'obtenir quelques fonctions usuelles: + , - , * , / , sinus, cosinus, etc. Ceci ne résout pas le cas du programme quelconque, chaque utilisateur ne peut se permettre de faire câbler son problème, quand on sait qu'il y a des applications qui comportent des millions d'instructions et en plus, qui sont évolutives. La réalisation matérielle d'un câblage est coûteuse et ne peut être envisagée raisonnablement que dans le cas d'algorithmes d'utilité véritablement générale. C'est donc le cas le plus large que nous allons envisager.

Un nouveau problème se pose alors, dont la solution permet, comme nous le verrons, d'apporter une réponse à des questions laissées en suspens. Pour la commodité de l'exposé, j'écris ainsi un calcul:

A: C1 -> C2

qui se lit: l'algorithme A appliqué à la configuration C1 donne la configuration C2.

Je vais donc essayer de concevoir un algorithme U qui serait capable de lire chacun des éléments de A et au fur et à mesure, de faire évoluer C1 jusqu'à l'obtension de C2.

Imaginons que U existe et appelons-le algorithme universel, et envisageons les contraintes auxquelles il est soumis. A l'instar de tout algorithme, cet algorithme universel travaille sur une configuration contenue en mémoire, ceci implique qu'il puisse disposer d'une représentation de l'algorithme à lire et d'une représentation de la configuration qui va avec. Partant ainsi d'une représentation en mémoire, je dirai un codage, du couple formé par mon algorithme quelconque A et sa configuration C1, l'algorithme universel interprèterait le code de l'algorithme objet pour modifier la

configuration représentée, exactement comme l'algorithme lui-même l'aurait traitée. En mémoire, on va devoir retrouver à la fois l'algorithme sous une forme codée, et la configuration initiale. Les deux, placés au même niveau, mais de toute évidence non permutables.

Pour atteindre à cet état, quelques conditions doivent être remplies. D'abord il faut trouver un bon codage qui s'adapte à n'importe quel algorithme. Là une restriction s'impose tout naturellement, ceci n'est pas possible en toute généralité mais peut parfaitement se concevoir si l'on se limite à une classe d'algorithmes décrits formellement à l'aide d'un même langage.

Je dirai que l'aspect formel de la machine de Nolin constitue un langage, la procédure formelle en constitue un autre. On retombe sur une propriété bien connue. En effet nous avons montré que la machine de Nolin permet de construire toute l'arithmétique des nombres entiers, il resterait à montrer qu'il n'existe pas d'autre **calcul effectif** que celui qu'on peut réaliser avec une machine de Nolin. Je vais admettre cette propriété. Je présenterai alors le projet sous la forme pratique qui s'impose: je me place dans le cas où tout algorithme considéré est écrit en langage de la machine de Nolin.

On peut également représenter les choses de la façon suivante, si je désigne par R1 une représentation d'algorithme et R2 une représentation de configuration, partant d'un calcul quelconque:

A : C1 -> C2

A est l'algorithme, C1 la configuration initiale et C2 la configuration finale, résultat du calcul. Tout calcul peut évidemment se mettre sous cette forme. Je reprends U l'algorithme universel, alors le calcul défini par l'application de U se décrit ainsi:

U : (R1(A) , R2(C1)) -> R1(A) , R2(C2)

dans la mesure où je connais les représentations R1 et R2, la configuration sur laquelle s'applique U est déterminée. Dans la configuration résultat du calcul de U, seule la partie qui correspond à la représentation de la configuration a pu évoluer.

Si je réussis à construire U, ce qui signifie que R1 existe et est pertinente, alors il suffira de le câbler avec une mémoire pour pouvoir simuler le déroulement de n'importe quel calcul, donc, pouvoir simuler l'application de n'importe quel algorithme sur une configuration correspondante.

Observons le langage de la machine de Nolin:

```
S0   début
S1   ca := 0
S2   ca := cb
S3   ca := ca +K 1
S4   si ca = 0 vers E
S5   vers F
S6   si cw = 0 vers P
S7   cw := ca
S8   ca := cw
S9   fin
```

Je rajoute une condition qui fait intervenir explicitement la case w, ceci m'est imposé par le mode de codage que je choisis, comme on va le voir. Bien que cela ne soit pas encore utile pour l'instant, je rajoute début et fin considérés comme des instructions.

Le langage de l'algorithme quelconque que j'ai appelé A sera celui de la machine de Nolin. Pour décrire U je préfère choisir tout de suite un langage plus commode, en l'occurrence la Procédure Formelle.

Coder une instruction ayant l'une ou l'autre des formes présentées ci-dessus, implique que l'on note dans des cases de mémoire ce qui doit permettre à un algorithme de simuler chaque fois le calcul correspondant. Pour pouvoir définir R2 la représentation de la configuration, il faut que je définisse une part de mémoire qui joue le rôle de la mémoire de la machine de Nolin considérée.

Pour simplifier, je vais supposer que les contenus des cases de la machine de Nolin considérée, sont identiques aux contenus de cases de la machine Formelle. S'il n'en était pas ainsi, la traduction n'en serait pas compliquée.

Cette part de mémoire, destinée à contenir la valeur de C1 et de C2, je la repère avec une adresse d, par exemple. De telle sorte que la case d'adresse d joue le rôle de la case 0, la case d'adresse d+1 celui de la case 1 , la case d'adresse d+a celui de la case a.

Les techniques de codage sont multiples selon l'effet désiré. Dans le cas présent, en tenant compte du faible nombre de formes différentes, je peux considérer chacune de ces formes comme un opérateur que je code à l'aide d'un entier. Ainsi pour la forme S1 l'opérateur consiste à affecter zéro à la case désignée. Pour la forme S2, l'opérateur consiste à copier le contenu de la case désignée à droite dans la case désignée à gauche.

Je représente le premier opérateur par ':=0' , et le second par ':=' . Notation destinée à l'exposé qui me permet de rappeler le type de traitement. Et à chacune de ces valeurs j'affecte un code quelconque mais dont je ne change plus après l'avoir choisi. Par exemple 0 pour ':=0' , 1 pour ':=' etc.

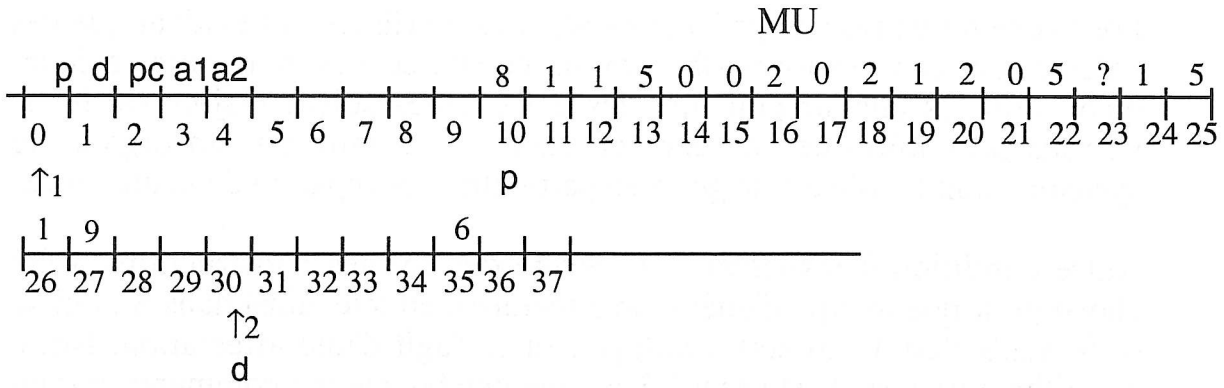
J'observe aussi que chaque opération est paramétrée selon les noms des cases sur lesquelles il faut agir, et également selon les étiquettes. Pour coder une forme je noterai donc le code de l'opérateur, qui tiendra dans une case et dans les cases suivantes je noterai les numéros des cases à traiter, avec le code de l'étiquette quand il y a lieu. Je constitue un tableau où sont reportées toutes ces informations avec un exemple pour illustrer.

Formes	opér.	code	exemples	
ca := 0	':=0'	0,a	c4 := 0	0,4
ca := cb	':='	1,a,b	c6 := c12	1,6,12
ca := ca +K 1	':=+'	2,a	c5 := c5 +K1	2,5
si ca = 0 vers E	'si='	3,a,codeE	si c2=0 vers Z	3,2,code Z
vers F	'vers'	4,code F		
si cw = 0 vers P	'siw'	5,code P		
cw := ca	'wa'	6,a	cw := c15	6,15
ca := cw	'aw'	7,a	c21 := cw	7,21
début	'deb'	8		
fin	'fin'	9		

A titre d'exemple je reprends un algorithme simple, je le code et je le mets en place en mémoire. Je décide, par exemple d'implanter le code à partir de la case 10.

Algorithme	code	N° de case
début	8	10
c1 := c5	1,1,5	11
c0 := 0	0,0	14
c0 := c0 $+_K$ 1	2,0	16
MU : c1 := c1 $+_K$ 1	2,1	18
c0 := c0 $+_K$ 1	2,0	20
si cw = 0 vers MU	5,code MU	22
c5 := c1	1,5,1	24
fin	9	27

Je me place maintenant dans le cadre de la machine formelle, le contenu de la case I me définit une origine, et c'est par rapport à cette origine que je place le code de l'algorithme. De la même manière je mets en place une configuration sur laquelle est susceptible de s'appliquer cet algorithme. Cette configuration est constituée des cases 0, 1, 5 et w. Arbitrairement je décide de l'implanter à partir de la case 30. Ainsi cette case 30 représente la case 0 de l'algorithme, la case 31 en représente la case 1, et 35 la case 5.



La ↑ indique l'origine que donne le contenu de I. La ↑2 repère l'origine de la configuration représentée.

Je vais construire l'algorithme qui va lire le code enregistré à partir de la case 10, et modifier en conséquence la configuration image traitée à partir de la case 30.

D'abord il me faut résoudre le problème du contenu de la case 23. En 22 le code 5 indique qu'il s'agit de la condition sur w. Ainsi si le contenu de w est différent de 0 il faut dérouler l'instruction suivante dont le code commence en case 24. Mais s'il est égal à 0 alors il faut dérouler l'instruction étiquetée par MU et dont le code commence en case18. Je choisis comme code de MU un entier qui compte le nombre de cases qui séparent la case 22 de la case18 pour une raison que je justifierai tout-à-l'heure. En 23 je mets donc -4, car je compte négativement en remontant dans la mémoire, et positivement en descendant. Or, je ne sais pas écrire -4 dans une case, mais par contre je sais écrire K-4, et cela présente le même sens pour ce que je veux faire.

Les codes d'étiquettes seront donc toujours traités de la même manière. Qu'il s'agisse d'une condition comme pour S4 et S6 ou d'une commutation systématique comme pour S5.

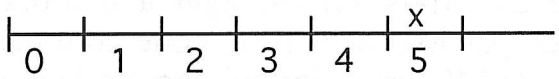
Dans la case réservée au code de l'étiquette, je place un entier qui mesure en nombre de cases la distance qui sépare la case qui contient le code opération de l'instruction de commutation de celle qui contient le code opération de l'instruction étiquetée. Je compte négativement si je me déplace vers des cases de numéros plus petits et positivement si je me déplace vers des cases de numéros plus grands.

Je vais donc poser le problème de la rédaction de l'algorithme de la machine universelle. Cet algorithme a, par définition, besoin de deux paramètres, l'un est l'accès au code de l'algorithme à dérouler, l'autre est l'accès à la configuration de cet algorithme, en toute généralité ce sont p et d ces deux adresses, et je les place dans les cases 0 et 1 de la configuration de la machine universelle.

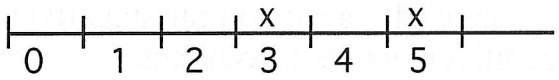
Pour l'exemple présenté p=10 et d=30. Il est parfaitement évident que ces valeurs sont choisies tout-à-fait arbitrairement, et il est non moins évident qu'on doit les choisir pour que les deux zones soient disjointes. Nous verrons plus loin que ce genre de choix est à faire par un organe, le système, dont le rôle est de gérer en particulier l'occupation de la mémoire.

Autre condition à respecter, Le codage des formes d'instructions a été choisi pour que le type d'opération apparaisse en tête, ainsi dans S2 qui se code 1,a,b c'est 1 qui sert à indiquer qu'il s'agit d'une affectation. Notre algorithme universel est organisé comme une boucle qui commence par un aiguillage. Et l'aiguillage a pour rôle de déterminer le type d'opération à dérouler, l'ayant détecté il faut alors calculer les opérandes pour pouvoir réaliser l'opération, puis se mettre en position pour permettre à l'aiguillage d'analyser le code de l'opération suivante.
C'est l'adresse pc contenue dans la case 2 qui sert à explorer le code de l'algorithme à dérouler, cette valeur est initialisée à partir du paramètre p.

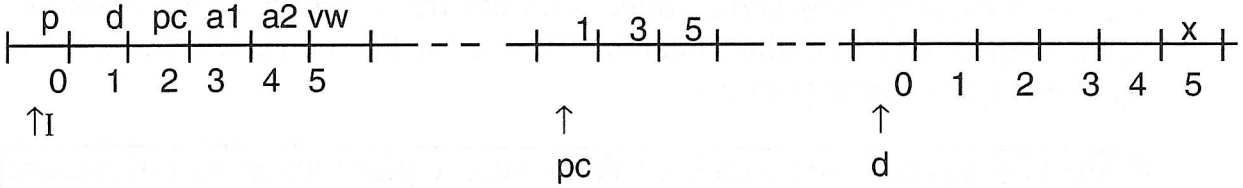
Je vais expliquer sur un exemple le principe du traitement, imaginons qu'on dispose d'une instruction c3 := c5
destinée à s'appliquer sur une configuration:



après application on doit obtenir:



Après avoir codé j'obtiens une configuration qui prend la forme:



et où ce qui se trouve en adresse d est la configuration image.

L'algorithme que je vais construire ne connaît par définition que l'origine ↑I , il constate qu'en adresse pc par rapport à cette origine il y a 1 c'est-à-dire le code ':=' , il lui faut atteindre à la case 5 par rapport à d, puis à la case 3 toujours par rapport à d.

Il va lui falloir construire deux adresses a1 et a2 qui vont lui permettre d'atteindre à ces deux cases par rapport à son origine ↑I . De cela on déduit que:

a1 = d + 3 et que a2 = d + 5

or, le d se trouve à l'adresse 1 et le 3 et le 5 respectivement en adresse pc+1 et pc+2, on dispose ainsi de suffisamment d'information pour calculer.

Pour ce qui est de l'exemple qui précède, après avoir calculé a1 et a2 il suffit d'écrire:

$$[_{3,0} := [_{4,0}$$

pour que la valeur x contenue dans la case 5 de la configuration image soit dupliquée dans la case 3 de cette même configuration.

Après qu'on ait pu déterminer le type d'opération à appliquer, on calcule la ou les adresses des opérandes, il ne reste plus qu'à appliquer l'opération, c'est ce que je vais montrer dans l'algorithme qui suit.

```
début
[2 := [0                          Initialisation de pc.

Entrée : si [2,0 = ':=0' vers Z   Aiguillage sur le type d'opération.
         si [2,0 = ':=' vers DPE
         si [2,0 = ':=+' vers PLUN
         si [2,0 = 'si=' vers COND
         si [2,0 = 'vers' vers VERS
         si [2,0 = 'siw' vers SIOM
         si [2,0 = 'wa' vers AOM
         si [2,0 = 'aw' vers OMA
         si [2,0 = 'deb' vers DEB
         si [2,0 = 'fin' vers FIN

Z :   [3 := [1 + [2,1             d+a, adresse de la case à mettre à
                                  zéro.
      [3,0 := 0                   Mise à zéro.
      [2 := [2 + 2                Le code de cette opération tient
                                  en deux
      vers Entrée                 cases, calcul de pc pour le code
                                  suivant.

DPE :[3 := [1 + [2,1              d+a, adresse du premier
                                  opérande,
      [4 := [1 + [2,2             d+b, adresse du second opérande,
      [3,0 := [4,0                recopie de la seconde case dans la
                                  première.
      [2 := [2 + 3                Le code occupe trois cases.
      vers Entrée
```

```
PLUN : [3 := [1 + [2,1          d+a, adresse de la case dont le
       [3,0 := [3,0 + 1         contenu doit être incrémenté, puis
                                addition de 1.
       [5 := [w                 Réservation du débordement.
       [2 := [2 + 2             Code à deux cases.
       vers Entrée

COND : [3 := [1 + [2,1          d+a, adresse de la case dont le
                                contenu est à sonder.
       si [3,0 = 0 vers CD1     Puis sondage.
       [2 := [2 + 3             S'il n'y a pas 0 calcul du pc de
       vers Entrée              l'instruction qui suit.

CD1 : [2 := [2 + [2,2           Addition du code étiquette à la
                                valeur courante de pc,
       vers Entrée              pour sauter à l'instruction
                                étiquettée.

VERS : [2 := [2 + [2,1          Addition du code étiquette à la
       vers Entrée              valeur de  pc.

SIOM : si [5 = 0 vers SIO1      Exploration du contenu de l'image
                                de w.
       [2 := [2 + 2             S'il est non nul: instruction
                                suivante.
       vers Entrée

SIO1 : [2 := [2 + [2,1          S'il est nul: instruction étiquettée.
       vers Entrée

AOM :  [3 := [1 + [2,1          d+a, adresse de la case à recopier
                                en w.
       [5 := [3,0               Recopie.
       [2 := [2 + 2
       vers Entrée              Instruction suivante.

OMA :  [3 := [1 + [2,1          d+a, adresse de la case dans
                                laquelle on
       [3,0 := [5               recopie w. Puis recopie.
       [2 := [2 + 2
       vers Entrée              Instruction suivante.
```

```
DEB :   [2 := [2 + 1
        vers Entrée          Instruction suivante.

FIN :   vers Entrée          Blocage sur l'instruction fin.
```

Remarque 1

A titre de remarque à propos du code de l'étiquette, en imaginant qu'on applique cette machine sur l'exemple précédent, et que pc ait pour valeur 22, on constate que la sixième instruction de l'aiguillage renvoie à l'étiquette SIOM. Et là deux cas se présentent:
-la case w contient 0, alors en SIO1 on ajoute le contenu de la case 2 à celui de la case en pc+1, soit pc+(K-4), ou encore 22-4=18 . Et on retourne en Entrée où s'analysera l'instruction dont le code est en case 18. C'est bien le code de l'instruction qui porte l'étiquette MU;
-la case w ne contient pas 0, alors à pc on ajoute 2 soit 22+2=24 adresse qui indique le code de l'instruction suivante.

Remarque 2

Lorsque la machine universelle rencontre un code de fin elle se met à tourner en rond sur ce code. Il y a là visiblement problème. Il suffit d'élargir le débat pour saisir de quoi il s'agit. Ayant achevé le déroulement d'un algorithme, à quoi peut-on s'attendre sinon à engager un autre calcul. Qui peut d'ailleurs être le même, mais portant sur d'autres données, soit carrément différent, soit encore on peut souhaiter consulter les résultats obtenus, etc. Dans tous ces cas il faut pouvoir fournir une information supplémentaire à la machine universelle, voire entamer une opération de transfert mémoire centrale à mémoire externe, ou l'inverse. Je classerai toutes ces opérations dans une catégorie qualifiée de **'système de gestion de l'ordinateur'**.

Exercice

Appliquer cet algorithme de machine universelle sur la totalité de l'exemple.

EXTENSION DU LANGAGE

Pour définir cette machine universelle je suis parti de la machine de Nolin la plus élémentaire. Mais nous avons pu observer que rapidement s'est imposée à nous la nécessité de construire des algorithmes d'usage courant indispensables pour effectuer le moindre calcul. Ainsi pour le 'moins-un' , l'addition, la multiplication, la comparaison à d'autres valeurs que 0, voire la comparaison entre deux cases, etc...

La question est: "si nous incluons ces formes dans le langage que devient la machine universelle?".

Je vais me fixer quelques instructions de plus à titre d'exemple, l'addition, une comparaison entre deux cases, mais la réalisation de la soustraction, de la multiplication, des autres opérateurs de comparaison, etc, est proposée comme exercice au lecteur.

Je prends en considération les deux formes:

ca := cb $+_K$ ce

si ca > cb vers E

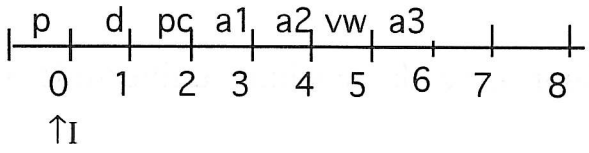
Dans le tableau qui me donne les codes des opérateurs, je rajoute deux lignes pour ces instructions supplémentaires.

forme	opérateur	code	exemples	
ca := cb $+_K$ ce	':=add'	10	c67 := c23 $+_K$ c33	10,67,23,33
si ca > cb vers E	'si>'	11	si c44 > c9 vers G6	11,44,9,codeG6

Dans l'aiguillage qui détermine le type d'opérateur rencontré, je rajoute aussi deux lignes:

```
Entrée : si [2,0 = ':=0' vers Z
         - - - - - - - - - - - -
         - - - - - - - - - - - -
         si [2,0 = ':=add' vers ADD
         si [2,0 = 'si>' vers SPG
Z:       - - - - - - - - - - - -
```

Il me faut d'abord compléter la configuration de la machine universelle. En effet, pour l'addition il va me falloir calculer trois adresses. Je réserve donc une case de plus à cet usage, ce sera la case 6:



Donc:

```
ADD :[3 := [1 + [2,1        d+a, adresse du premier  opérande,
     [4 := [1 + [2,2        d+b, adresse du second opérande,
     [6 := [1 + [2,3        d+e, adresse du troisième opérande
     [3,0 := [4,0 + [6,0
     [5 := [w               Calcul de l'addition.
     [2 := [2 + 4           Le code occupe quatre cases.
     vers Entrée
```

```
SPG :  [3 := [1 + [2,1        d+a, adresse de la première case
       [4 := [1 + [2,2        d+b, adresse de la seconde case.
       [7 := [3,0             Appel des valeurs en cases 7 et 8,
       [8 := [4,0             pour pouvoir les détruire.
SPG3 : si [8 = 0 vers SPG1
        si [7 = 0 vers SPG2
       [7 := [7 - 1           Epluchage des deux valeurs.
       [8 := [8 - 1
       vers SPG3
SPG1 : si [7 = 0 vers SPG2
       [2 := [2 + [2,3        La première valeur est
       vers Entrée            effectivement plus grande que la
                              seconde.

SPG2 : [2 := [2 + 4           La comparaison n'est pas vérifiée.
       vers Entrée
```

On pourrait de la sorte étendre considérablement le jeu des instructions dont on peut disposer dans le langage. Je conseille vivement au lecteur de compléter la machine pour le cas de la soustraction et de la multiplication, comme je le proposais plus haut.

Désormais j'appellerai **programme** un algorithme codé destiné à être déroulé sous le contrôle d'une machine universelle, après avoir été introduit en mémoire.

Je fais référence à la précédente remarque 2, l'observation de notre machine universelle va nous obliger à poser un problème supplémentaire. En effet quand on considère l'instruction fin on s'aperçoit que la machine telle que je l'ai écrite, tourne en rond à partir du moment où elle l'a rencontrée. Cela signifie que cet ordinateur, si ordinateur il y a, déroule un programme et se met en attente.

La conception et la réalisation d'une machine universelle représente un investissement qui ne peut véritablement prendre tout son intérêt que s'il devient possible de la mettre dans une situation où il est commode de lui soumettre autant de programmes qu'on le désire. Or, pour l'instant d'une part rien ne permet d'arrêter le déroulement de la machine, et d'autre part rien encore ne permet d'enchaîner deux ou plusieurs déroulements de programmes. De plus je n'ai pas encore montré comment mettre la machine en situation de travailler, c'est-à-dire comment introduire les divers codes.

Se donner les moyens d'enchaîner les tâches les unes aux autres, c'est construire ce qu'on appelle généralement un système de gestion de l'ordinateur, plus brièvement un système.

CODES

Le système est en quelque sorte l'organe qui va mettre l'intérieur de l'ordinateur en contact avec l'extérieur, c'est-à-dire nous, les utilisateurs. Or, pour nous, le contenu d'une case selon son aspect, est un élément d'un langage bien ésotérique. Pour pouvoir l'interpréter il se révèle commode voire indispensable de transformer ce contenu en un signe qui corresponde mieux à nos habitudes. Ainsi même si je parviens à visualiser une valeur binaire, j'aurai du mal à la lire, le décimal m'est plus agréable.

Et puis l'ordinateur ne servira pas qu'à faire des calculs arithmétiques, les traitements de textes, les systèmes de gestion des bases de données et autres sont la manifestation d'une foule de possibilités de services non encore toutes épuisées

.

D'un point de vue extérieur, pour nous permettre de communiquer avec l'ordinateur les machines à écrire ou imprimantes d'usage traditionnel,se révèleraient bien utiles

.

Là des contraintes techniques apparaissent, le nombre de touches du clavier est forcément borné même si chaque touche peut avoir jusqu'à trois usages différents. Il en est de même pour les polices des imprimantes. De toute manière pour écrire un nombre suffisamment grand il faut bien aligner une suite de chiffres.

On a donc été amenés à choisir des codes destinés à être mis en correspondance avec chacun des caractères utilisés couramment dans la graphie des textes tels que les alphabets, les jeux de chiffres, les caractères de ponctuation, et divers signes spécialisés.

On a pu disposer sur les premiers ordinateurs d'alphabets pauvres de 36 signes, puis la puissance des ordinateurs croissant ainsi que la variété de leurs utilisations, les constructeurs visiblement avec regret ont fini par enrichir les polices de caractères. Je donne deux exemples parmi les plus courants de tels alphabets ainsi que les codes qui vont avec.

Dans la première colonne se trouve la lettre telle qu'elle apparaît sur la touche du clavier, ou imprimée sur le papier, et dans la colonne de droite et en face on voit le code exprimé en hexadécimal. Ainsi le "!" a pour représentation: 21, ou encore: 0010 0001 en binaire.

C'est le cas du code dit ASCII (ici 7 bits):

SP	20	0	30	@	40	P	50	`	60	p	70
!	21	1	31	A	41	Q	51	a	61	q	71
"	22	2	32	B	42	R	52	b	62	r	72
#	23	3	33	C	43	S	53	c	63	s	73
$	24	4	34	D	44	T	54	d	64	t	74
%	25	5	35	E	45	U	55	e	65	u	75
&	26	6	36	F	46	V	56	f	66	v	76
'	27	7	37	G	47	W	57	g	67	w	77
(	28	8	38	H	48	X	58	h	68	x	78
)	29	9	39	I	49	Y	59	i	69	y	79
*	2A	:	3A	J	4A	Z	5A	j	6A	z	7A
+	2B	;	3B	K	4B	[	5B	k	6B	{	7B
,	2C	<	3C	L	4C		5C	l	6C	:	7C
-	2D	=	3D	M	4D	]	5D	m	6D	}	7D
.	2E	>	3E	N	4E	^	5E	n	6E	~	7E
/	2F	?	3F	O	4F	_	5F	o	6F	DEL	7F

Je ne donne pas le code au complet car nombre de valeurs servent de caractères de contrôle, à usage interne, et de toute manière elles ne sont pas visualisables. Il en traîne encore deux, ce sont SP et DEL. L'autre code est toutefois moins courant, c'est l'EBCDIC:

a	81	j	91		A1	A	C1	J	D1		E1
b	82	k	92	s	A2	B	C2	K	D2	S	E2
c	83	l	93	t	A3	C	C3	L	D3	T	E3
d	84	m	94	u	A4	D	C4	M	D4	U	E4
e	85	n	95	v	A5	E	C5	N	D5	V	E5
f	86	o	96	w	A6	F	C6	O	D6	W	E6
g	87	p	97	x	A7	G	C7	P	D7	X	E7
h	88	q	98	y	A8	H	C8	Q	D8	Y	E8
i	89	r	99	z	A9	I	C9	R	D9	Z	E9
0	F0	œ	4A	!	5A		6A	:	7A		
1	F1	.	4B	$	5B	,	6B	#	7B		
2	F2	<	4C	*	5C	%	6C	@	7C		
3	F3	(	4D	)	5D	-	6D	'	7D		
4	F4	+	4E	;	5E	>	6E	=	7E		
5	F5	|	4F	¬	5F	?	6F	"	7F		
6	F6	7	F7	8	F8	9	F9	&	50	_	60

Là encore je ne donne que l'essentiel, mais on peut constater que d'un code sur l'autre les représentations sont on ne peut plus différentes. La même lettre A a pour code C1 en EBCDIC et 41 en ASCII. Et dans les deux cas les chiffres ne sont pas représentés par leur valeur binaire, par exemple le chiffre "1" est codé 31 EN ASCII, soit 0011 0001 en binaire et F1 en EBCDIC, soit 0001 1111 en binaire.

Le caractère est introduit en mémoire sous la forme de son code, de telle sorte que pour construire des nombres par l'introduction d'une suite de chiffres il faut par algorithme transformer le code introduit en la valeur effective qui est censée lui correspondre. C'est ce problème que nous allons essayer de résoudre dans le chapitre du système.

CHAPITRE 6

NOTION DE SYSTEME

Edmond Bianco

Le code de l'algorithme à dérouler doit par définition se trouver en mémoire pour être déroulé. Mais je n'ai pas encore dit comment l'y introduire, ni comment introduire les données qui lui seront soumises. Or, nous avons vu que pour être complète, une machine de Nolin doit posséder des files externes dont la principale propriété est d'être illimitées en nombre de cases. Je vais donc me donner deux de ces files externes, dont je définis ainsi les propriétés.

L'une fonctionne comme un clavier de machine à écrire, et permet d'introduire un code en mémoire chaque fois qu'on appuie sur une touche, par exemple un code ASCII.

L'autre fonctionne comme un écran cathodique, et visualise sur des lignes, des codes qui sont extraits de la mémoire.

Ces deux files sont spécialisées, l'une seulement en entrée, elle permet l'introduction en mémoire des codes, l'autre seulement en sortie d'information, c'est grâce à elle qu'on va pouvoir consulter la mémoire. Il me faut deux instructions d'échanges pour pouvoir programmer l'utilisation de ces files.

introd $[_a$	introd $[_{a,b}$
extract $[_a$	extract $[_{a,b}$

Ce sont là des formes particulières pour les formes connues:
ca := cfi et **cfj := ca** .

Les deux premières: "introd", ont pour rôle d'introduire à partir du clavier, des valeurs binaires qui vont se retrouver soit en case "a", soit en case dont l'adresse est en "a", corrigée de la valeur "b". Les deux instructions "extract" font l'inverse, à partir d'une valeur binaire située en "a" ou en adresse contenue en "a", corrigée de "b", elles vont visualiser sur l'écran, le caractère qui correspond à cette valeur binaire. Consulter les tableaux qui donnent les codes ASCII ou EBCDIC.

Je vais construire un algorithme qui va jouer le rôle du système, et qui fonctionne en deux phases. Pendant la première, il attend une injonction, dès que l'injonction est reçue il la reconnaît et entre dès lors dans la phase

seconde correspondante dont il ne sort plus que sur la reconnaissance de la fin de cette phase, qui le renvoie sur la première phase.

Cet algorithme est destiné à piloter le fonctionnement complet de l'ordinateur, tout doit donc y être prévu. A partir de l'instant où la machine est mise sous tension c'est cet algorithme qui commence à se dérouler. Tout de suite au début il faut prévoir la répartition de la place en mémoire. Or, trois emplacements sont nécessaires, l'un pour la configuration de la machine universelle et du système, l'autre pour placer le code-programme à dérouler, le troisième pour les données dans la configuration du programme. Je me place dans un cas simple tel qu'un seul programme se déroule à la fois. Je vais donc mettre tout-à-fait arbitrairement la configuration du système en adresse réelle 1000, ce qui me définit l'origine, puis l'emplacement du code à partir de la case relative 20, et enfin l'emplacement des données à partir de 3000 par rapport à l'origine.

LE LANGAGE DU SYSTEME

Si le système a pour rôle d'organiser le travail dans l'ordinateur, il ne conserve une initiative que dans des limites étroites, pour l'essentiel, il agit sous l'effet d'injonctions qui lui sont fournies de l'extérieur. En fait il a à traiter deux sortes d'informations, des injonctions pour l'organisation, et des suites de données destinées à décrire algorithmes et configurations.

Toute cette information qu'il doit traiter lui sera présentée sous forme d'un langage de communication. Je le choisis très simple pour l'exemple simple présenté ici.

D'abord une liste d'injonctions.

Je veux introduire un programme	:	code 'P',
introduire des données	:	-"- 'D',
extraire des données	:	-"- 'L',
lancer un calcul	:	-"- 'C'.

Chacun des caractères 'P', 'D' ou 'L', indique au système que la liste des entiers qui suivent représentent soit du programme, soit des données à charger en mémoire ou à extraire sur l'écran. La dernière injonction: 'C' ne doit être utilisée que si un programme et une configuration ont déjà été introduites.

Je veux introduire le petit programme:

début c7 := c3 +K c4 fin.

Je vais en conséquence présenter au système tel qu'il est décrit plus bas la phrase:

P 8 , 10 , 7 , 3 , 4 , 9 ;

Le système lit: P à partir de cet instant il s'attend à une suite d'entiers qui seront séparés par des virgules, la fin de la phrase est marquée par un point-virgule.

Ce petit programme travaille sur une configuration qui comporte huit cases:

0	0	0	18	21	0	0	0
0	1	2	3	4	5	6	7

L'application de ce programme sur cette configuration aura pour but de faire apparaître la somme de 18 et 21 en case 7. Pour cela il me faut introduire la phrase:

D 0 , 0 , 0 , 18 , 21 , 0 , 0 , 0 ;

Présenter alors le caractère L à ce système aurait pour conséquence de faire apparaître sur l'écran la suite:

0 , 0 , 0 , 18 , 21 , 0 , 0 , 0 ;

Il suffirait alors de présenter le caractère C au système pour que le programme se déroule, et une nouvelle présentation du caractère L ferait apparaître sur l'écran:

0 , 0 , 0 , 18 , 21 , 0 , 0 , 39 ;

CONFIGURATION DU SYSTEME

C'est dans l'espace des 1000 premières cases que je place le code du système lui-même, et je découpe tout-à-fait arbitrairement la mémoire pour placer les diverses configurations. On envisagera des systèmes plus complexes, capables de gérer eux-mêmes la mémoire en fonction des êtres qui l'occupent. Pour l'instant, limitons-nous à une sorte de structure minimum, dont nous verrons qu'il est relativement facile de la compléter et de la perfectionner.
Le système est construit en se basant sur les hypothèses suivantes:
a) le code du système lui-même est pré-enregistré en mémoire à partir de la case 0.
b) la configuration du système commence à la case 1000, donc dès le départ on charge 1000 en [I ;
c) c'est à partir de 1020 qu'on stocke le code du programme à dérouler, en 1019 on place "n" le nombre de cases qu'occupe le programme, "p" est l'adresse courante de son code;

d) la configuration du programme commence à 3000 en adresse relative, donc, en l'occurence, à l'adresse effective 4000;

La case 3999 contient le nombre de cases occupées par la configuration.
e) les caractères sont lus dans la case 7 de la configuration du système;
f) en case 8 on construit les nombres.

```
1           début
            [I := 1000
            [0 := 20
            [1 :=3000
Phase1 :    introd [7
6           si [7 = 'P' vers IP
            si [7 = 'D' vers ID
            si [7 = 'L' vers SD
            si [7 = 'C' vers C

IP :    [2 := [0
11      [10 := [0 - 1          Réservation de la première case de la
                               file du programme.
        vers ID0

ID :    [2 := [1
15      [10 := [0 - 1          Réservation de la première case de la
                               configuration.
ID0 :   [9 := 0                On va compter le nombre d'éléments
                               introduits.
ID1 :   [8 := 0
SID :   introd [7              Lecture d'une lettre au clavier.
19      si [7 = ',' vers VIRG  C' est une virgule, le nombre est
                               complet.
20      si [7 = ';' vers PVG   C'est un ';' le nombre en cours est
                               complet, la liste est achevée.
        [8 := [8 * 10          La valeur précédente est multipliée par
                               10.
        si [7 = '0' vers SID
        [8 := [8 + 1           On ajoute la valeur du chiffre.
        si [7 = '1' vers SID
25      [8 := [8 + 1           ↓
        si [7 = '2' vers SID
        [8 := [8 + 1           ↓
```

```
        si [7 = '3' vers SID
        [8 := [8 + 1                    ↓
30      si [7 = '4' vers SID
        [8 := [8 + 1                    ↓
        si [7 = '5' vers SID
        [8 := [8 + 1                    ↓
        si [7 = '6' vers SID
35      [8 := [8 + 1                    ↓
        si [7 = '7' vers SID
        [8 := [8 + 1                    ↓
        si [7 = '8' vers SID
        [8 := [8 + 1                    ↓
40      si [7 = '9' vers SID

VIRG :  [2,0 := [8                      Le nombre est construit, on l'expédie
        [2 := [2 + 1                    dans la configuration.
        [9 := [9 + 1
        vers ID1

PVG :   [2,0 := [8                      La liste est achevée, on expédie le
                                        nombre et on
46      [9 := [9 + 1                    retourne en attente des injonctions.
        [10,0 := [9                     On note en tête, le nombre
                                        d'éléments de la file.
        vers Phase1

SD :    [2 := [1                        Extraction des données.
50      [10 := [1 - 1
        [9 := [10,0                     Calcul du nombre de valeurs à
                                        extraire.
SD1 :si [9 = 0 vers FSD                 Boucle sur le nombre de valeurs.
        [11 := 10
        [8 := [2,0
CP :    si [8 < [11 vers ED             Calcul de la puissance de 10 qui borne
                                        le nombre.
56      [11 := [11 * 10
        vers CP
ED :    si [11 = 1 vers suite           Boucle d'extraction des facteurs des
                                        différentes
59      [11 := [11 ÷ 10                 puissances de 10 dans le nombre.
```

```
60      [7 := [8 ÷ [11
        vers SORT                    Emission de l'entier trouvé.
ED1:    [7 := [7 * [11
        [8 := [8 - [7
        vers ED
suite : [7 := ','                    Impression d'une ',' entre chaque
                                     nombre.
66      extract [7
        [9 := [9 - 1
        [2 := [2 + 1
        vers SD1

FSD :           [7 := ';'            Impression d'un ';' à la fin de la liste.
71      extract [7
        vers Phase1                  Retour au choix de l'injonction.

SORT : si [7 ≠ 1 vers S1             Emission du code qui correspond à la
                                     valeur du chiffre, afin que ce soit
        [7 := '1'                    l'image du chiffre
75      vers EMIS                    qui soit effectivement imprimée.
S1 :    si [7 ≠ 2 vers S2
        [7 := '2'
        vers EMIS
S2 :    si [7 ≠ 3 vers S3
80      [7 := '3'
        vers EMIS
S3 :    si [7 ≠ 4 vers S4
        [7 := '4'
        vers EMIS
S4 :    si [7 ≠ 5 vers S5
86      [7 := '5'
        vers EMIS
S5 :    si [7 ≠ 6 vers S6
        [7 := '6'
90      vers EMIS
S6 :    si [7 ≠ 7 vers S7
        [7 := '7'
        vers EMIS
S7 :    si [7 ≠ 8 vers S8
95      [7 := '8'
        vers EMIS
```

```
S8 :   si [7 ≠ 9 vers S9
       [7 := '9'
99     vers EMIS
S9 :   [7 := '0'
EMIS : extract [7
102    vers ED1

C  :   vers MUNIV          Lancement du calcul en faisant appel à
                           la machine universelle.
retour : vers Phase1
```

Machine Universelle

```
MUNIV :  [2 := [0          On reprend l'algorithme de la
                           machine universelle, tel qu'il a été
Entrée : si [2,0 = ':=0' vers Z  construit au chapitre 5 de la machine
                           universelle.
-  -  -  -  -  -  -  -     Sont supprimées simplement les
-  -  -  -  -  -  -  -     instructions début et fin . On modifie le
FIN :       vers retour    traitement du code de l'instruction fin,
                           qui, au lieu de faire tourner en rond la
                           machine universelle, renvoie dans le
                           système, pour reprendre l'analyse des
                           injonctions.
```

TRANSCODAGE DES NOMBRES

Dans cet algorithme, on trouve deux calculs qui n'ont d'autre intérêt que de transposer des codes. Dans un sens transformer la suite des codes qui représentent chacun un chiffre afin d'obtenir un nombre qui tienne dans une case. Dans l'autre sens à partir d'un nombre contenu dans une case émettre la suite des chiffres censée représenter le nombre en décimal.

Construction d'un nombre

Je raisonne à partir d'un exemple: 1993. Je suppose que les chiffres sont introduits dans l'ordre habituel: 1, puis 9, puis 9, puis 3, puis un signe qui indique que le nombre est complet ici une virgule ou bien un point-virgule. Dans une case je mets 0, et je commence la boucle: je multiplie le contenu de cette case par 10. Avec 10 comparaisons je détermine le code du chiffre lu, pour un '9' j'ajoute 9 dans la case, pour un '8' j'ajoute 8 dans la case, etc, ayant interprété ce chiffre j'attends un autre chiffre, dès qu'il arrive je recommence en multipliant le contenu de la case par 10 et en déterminant le code du chiffre lu. Ainsi:

case := 0 ,
case := case * 10 donc case = 0 ,
on lit '1' donc : case := case + 1 , case =1,
case := case * 10 donc case =10 ,
on lit '9' donc case := case + 9 , case = 19,
case := case * 10 donc case = 190 ,
on lit '9' donc case := case + 9 , case = 199 ,
case := case * 10 donc case = 1990 ,
on lit '3' donc case := case + 3 , case = 1993 ,
là on lit ',' le calcul est fini.
La lecture du chiffre ou des ',' et ';' qui ponctuent le nombre, s'effectue à la ligne 18, étiquette SID, du programme du système. Le calcul du nombre commence à la ligne 21 et s'achève à la ligne 40.

Impression d'un nombre

Je pars d'une valeur prise dans une case, et j'essaie de construire la suite des chiffres décimaux qui m'en donne l'équivalent décimal.

Exemple 872. L'extraction des données commence à la ligne 49, étiquette SD.
Je cherche la puissance de 10 immédiatement supérieure, ici c'est 1000. Ce calcul se fait sur les trois lignes 55,56,57. Je la réduis de 1 ce qui fait 100. La valeur divisée par ce nombre donne le premier chiffre:

872 ÷ 100 = 8 , Qui est calculé ligne 60,

j'imprime '8' , l'impression est calculée de la ligne 72, étiquette SORT, à la ligne 101, étiquette EMIS,
puis 872 - (100 * 8) = 72, Calculé aux lignes 62, 63,
la puissance de 10 réduite de 1, ce qui donne 10 sert pour le coup suivant, on revient à la ligne 58, étiquette ED:

72 ÷ 10 = 7 , j'imprime '7' , puis 72 - (10 * 7) = 2 ,

la puissance de 10 réduite de 1, ce qui donne 1 il ne reste plus qu'à imprimer '2'.

INTEGRATION, SYSTEME ET MACHINE UNIVERSELLE

Ici, une petite réflexion s'impose. On a pu constater sur l'exemple que je viens de construire, que finalement, système et machine universelle se situent sur le même plan. j'entends par là que de la manière qu'ils sont écrits, ces deux algorithmes n'en constituent de fait qu'un seul.

Après avoir rencontré l'injonction 'C', lancer un calcul, le système fait appel à un algorithme de déroulement qui n'est autre que la machine universelle présentée plus haut, mais simplement modifiée quant au traitement de l'instruction fin.

Si je déclare que le code du programme dénommé système est enregistré en mémoire dans la partie qui va de 0 à 1000, il est évident que le code de cette machine universelle est enregistré avec.

Or, notre ordinateur fonctionne parce qu'il existe une machine universelle, câblée, qui déroule les programmes enregistrés en mémoire, qui déroule donc ce fameux système dont j'ai fait l'hypothèse, précisément, qu'il est ce programme-orchestre qui organise tout ce qui se passe dans l'ordinateur, et qui déroulera également le code de la machine universelle qui lui est jointe.

INTEGRATION

Plusieurs questions complexes se soulèvent ici. On vient de voir que dans le cas qui nous occupe, une machine universelle câblée va devoir dérouler une machine universelle codée et placée en mémoire. Or, ces deux machines universelles sont précisément identiques de nature. Il y a là gaspillage de travail. Pour résoudre ce nouveau problème, on imagine les situations suivantes:

1) le cas brut tel qu'il est présenté ci-dessus;

2) seule existe la machine universelle câblée, tandis que le système, lui, est codé et placé en mémoire;

3) machine universelle et système sont tous les deux câblés.

La discussion de ces diverses éventualités fera l'objet des chapitres consacrés à la notion de systèmes généraux. Pour l'instant, on peut simplement dire que pour le cas 2) il faut se donner un moyen de communication entre système et machine universelle, ce sera un jeu d'instructions spécifiques, utilisées dans le système et traitées par la machine universelle. Pour le cas 3) le câblage du système lui enlève toute souplesse de modification, il faut donc être sur de sa théorie ou rechercher des applications très spécifiques pour adopter cette solution.

Depuis l'instant de la mise en route de l'ordinateur jusqu'à son arrêt, c'est le système qui contrôle la situation. Il initialise tout processus, gère les échanges avec les files externes, et lance les calculs des utilisateurs. C'est sur ce point que les diverses sortes de systèmes diffèrent le plus. Certains laissent les programmes se dérouler sans autres contrôles que les contrôles de débordement, et certains autres contrôlent rigoureusement les déroulements avec des compilateurs de déroulement qui ne sont rien autre que des machines universelles non câblées.

COMPILATION

Nous allons devoir nous poser un nouveau problème, face à la situation dans laquelle nous nous sommes placés. Nous avons un langage, la procédure formelle, nous avons une machine, la machine formelle et un procédé pratique, l'écriture de programmes.

Le programme est un algorithme mis sous forme de codes, il s'agit donc d'un travail en deux temps, on décrit l'algorithme dans un langage symbolique, ici la procédure formelle, et quand le texte s'en révèle satisfaisant, on procède alors à sa traduction en code.
Une double remarque nous amène au cœur de ce nouveau problème, cette transformation est systématique d'une part et elle est parfaitement et inutilement pénible d'autre part. On va donc chercher à l'automatiser, ce qui soulève la question: existe-t-il un algorithme capable de faire cela ?

S'il en existe un nous l'appellerons compilateur, et je vais essayer de montrer comment le construire en toute généralité. Pour cela je fais appel, comme d'habitude, à un exemple.

J'en choisis un, qui présente l'intérêt d'introduire quelque chose de plus, et d'important: la notion de variable symbolique. Je crée un langage, je le nomme LAC pour le désigner, et j'introduis un objet spécifique: l'identificateur.

Pour définir l'identificateur je reprends la définition du rapport ALGOL 60 qui est passée désormais dans la culture informatique, et utilisée dans la presque totalité des langages modernes.

L'objet identificateur est une suite quelconque de lettres et de chiffres qui commence obligatoirement par une lettre. Par lettre, j'entends une lettre de l'alphabet latin, et par chiffre l'un des dix chiffres de la numération décimale.L'identificateur sert à désigner symboliquement tout objet de la programmation, que ce soit une variable au sens mathématique du terme, une ligne de programme, il s'appelle alors étiquette, ou même un programme il est alors le nom de la procédure ou de toute autre construction algorithmique.

LE LAC

Ce langage ne comporte qu'une seule sorte de variables: celles qui prennent leurs valeurs dans l'ensemble des entiers dits naturels. Cela suffit car ajouter d'autres types de variables ne modifie pas le degré de complexité du compilateur il ne fait que multiplier la difficulté. Ceci signifie que rajouter d'autres types de variables n'introduit pas de concepts fondamentalement

différents mais oblige à multiplier les traitements dont l'ordre de difficulté est le même. Je soulignerai ces points au fur et à mesure qu'ils se présenteront.

Pour l'instant, et afin d'illustrer la programmation en LAC je vais donner un exemple. Cet exemple, je l'ai en apparence choisi un peu compliqué, mais il me semble qu'un informaticien même amateur, ne peut pas en ignorer la substance.

Je dispose d'une expression arithmétique quelconque, par exemple:

M1 := Z + Q * (HK - F) ou encore

X :=((((A + B) * C) + (D ↑ E)) - F)

La deuxième expression est dite "complètement parenthésée". Pour simplifier mon problème je vais supposer n'avoir affaire qu'à de telles expressions. Ce que je veux c'est les transformer en une suite d'expressions réduites à 3 opérandes seulement, cela donnerait pour la seconde:

a := A + B ; b := a * C ; c := D ↑ E ; d := b + c ; X := d - F ;

On constate sans difficulté que le déroulement de ces cinq calculs amène au même résultat que la deuxième expression, mais en plus correspond à la manière dont on la calculerait si on le faisait "à la main".

Je pourrais ajouter une petite remarque à propos de la première expression, telle qu'elle est construite, il y a une ambigüité dans sa décomposition, qui pourrait prendre les deux formes:

a := Z + Q ; b := HK - F ; M1 := a * b ;
a := HK - F ; b := Q * a ; M1 := Z + b ;

En général ce genre d'ambigüité est levée dans les langages de programmation en admettant des "priorités différentes par classes d'opérateurs arithmétiques: la ≠ étant prioritaire par rapport aux opérateurs multiplicatifs: * et ÷ , et ces derniers prioritaires par rapport aux additifs: + et - . A ce moment-là, seul le second développement serait admissible.
De toutes façons je ne considèrerai que les expressions entièrement parenthésées pour la facilité que cela apporte à l'exemple, outre le fait que cela supprime toute ambigüité.

Je définis 4 tableaux destinés à contenir à raison d'une lettre par case, PH, le premier, la phrase symbolique donnée, un autre RES la phrase résultat, enfin un tableau ID contient une source d'identificateurs suffisante pour les variables intermédiaires a, b, c, etc. Un tableau défini en variable locale, PILE sert de pile pour la construction des expressions réduites. Une variable locale C sert à contrôler une boucle.

```
        procédure ANAL ;
        paramètre PH , ID , RES ;
        index x , p , r , i ;
        var loc PILE(20) , C ;
        début
        x := 1 ;
        p := 1 ;
        r := 1 ;
        i := 1 ;
        RES(r) := ID(i) ;
ACC:    si PH(x) = ')' vers GEN ;
        PILE(p) := PH(x) ;
        p := p + 1 ;
        x := x + 1 ;
        vers ACC ;
GEN:    p := p - 1 ;
        r := r + 4 ;
        C := 3 ;
GEN1:   si C = 0 vers GENS ;
        RES(r) := PILE(p) ;
        r := r - 1 ;
        p := p - 2 ;
        si PILE(p) = ':=' vers FIN ;
        p := p + 1 ;
        C := C - 1 ;
        vers GEN1 ;
GENS:   RES(r) := ':=' ;
        r := r + 4 ;
        RES(r) := ';' ;
        r := r + 1 ;
        PILE(p) := I D(i) ;        Ici l'identificateur remplace la "("
        p := p + 1 ;               dans la pile.
        i := i + 1 ;
        RES(r) := ID(i) ;
        x := x + 1 ;
        vers ACC ;
FIN :   r := r - 1 ;
        p := p - 1 ;
        RES(r) := PILE(p) ;
        r := r + 1 ;
        RES(r) := ":=";
        r:=r+4;
        RES(r) := ";";
        fin
```

Le lecteur avisé, celui qui a compris ce que peut lui apporter ce livre, va tout de suite faire tourner cet algorithme à la main sur un exemple, pour bien voir comment il fonctionne. Le tableau PILE est représenté sur un tableau avec des colonnes pour suivre l'évolution de son contenu.

Le problème

Le problème qu'on est amené à se poser se définit ainsi: je dispose d'une telle phrase qui appartient à un langage dont la structure vient d'être définie, d'abord comment la rendre calculable automatiquement ? Une solution se présente, on peut chercher à lui trouver un équivalent en termes de machine connue. Nous disposons de la machine formelle, on va chercher à construire un programme de procédure formelle qui ait sémantiquement la même signification.

Ensuite, si on montre que cette image est sémantiquement équivalente, et en l'occurence déroulable, alors il faut montrer comment l'obtenir au moyen d'un automate qui n'est autre que le compilateur.

La méthode

On constate que la phrase symbolique se subdivise en deux grandes parties, une déclaration avec des listes de variables dont l'existence est ainsi définie, et un corps qui contient des instructions dont la forme est à choisir dans une liste de formes possibles. La déclaration donne en quelque sorte la structure de la configuration traitée. La suite des instructions décrit l'algorithme.

Partant de la déclaration, je cherche un équivalent en terme de configuration de procédure formelle, ce qui me procure une correspondance entre chaque variable symbolique, et une case de machine formelle. Correspondance qui, matérialisée dans un tableau fournit l'information pour construire les codes des opérandes des instructions-images, c'est-à-dire un code déroulable censé réaliser le même calcul que celui qui est décrit par l'instruction sysmbolique.

Je vais essayer de définir l'image de tout élément de phrase symbolique, et c'est ensuite qu'on essayera de montrer comment construire l'algorithme capable de calculer cette image.

Le langage

Il me faut ici donner quelques précisions quand à la syntaxe et à la sémantique du LAC. L'exemple illustre mieux qu'une longue palabre, et puis je préciserai certains points délicats au cours de la construction du compilateur.

Le LAC permet de construire des procédures. Une procédure se compose d'un "en-tête" et d'un corps, l'en-tête lui-même est subdivisé en la déclaration du nom de la procédure, et en la déclaration de la liste complète de ses variables.

Sur notre exemple: ANAL est le nom de la procédure, et PH, ID, ..., PILE, C. sont ses variables. On classe d'ailleurs ces variables selon leur rôle. Les paramètres représentent des objets qui existent à l'extérieur du domaine de calcul de la procédure. Les variables locales sont spécifiques au calcul de la procédure et n'ont pas d'existence en dehors. Les index jouent le rôle particulier du calcul du nom dans les variables indicées.

Le corps de procédure est constitué d'une liste d'instructions dont la forme est l'une des suivantes possibles:

m := n
m := n **A** n
si n **R** n vers **etiq**
vers **etiq**

m est soit une variable simple comme x, p ou C, ou une variable indicée comme PILE(p), RES(r) ou RES(3).

n prend toutes les formes de m plus celle d'une constante, comme dans: r := r + 4 .

A représente les opérateurs arithmétiques: + , - , * , ÷ .
R représente les opérateurs de relation: = , ≠ , > , < , ≥ , ≤ .
etiq représente une étiquette, qui est un identificateur que l'on attribue à une instruction pour pouvoir la mettre en relation avec une instruction de commutation, comme par exemple: ACC : si PH(x) = ... ; avec vers ACC;.

Je dois ajouter un mot à propos des "index" et des "variables indicées". Quand je décide de définir un tableau, qui est une suite de valeurs à laquelle je n'affecte qu'un seul nom, par exemple PH, il me faut un moyen pour désigner chacune des valeurs.

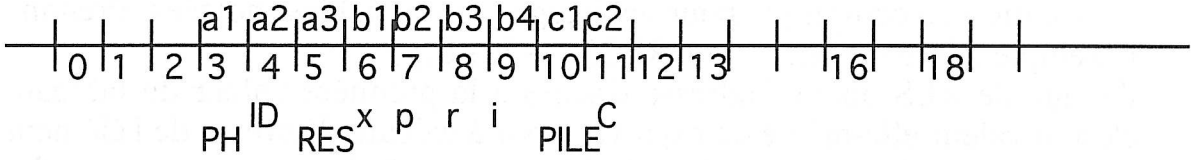
Traditionnellement on adopte la règle suivante:
PH(1) désigne la première valeur, PH(2) désigne la seconde valeur, PH(3) la troisième etc. Plus généralement PH(x) désigne la première valeur si x=1, la deuxième si x=2 etc. La variable déclarée comme index prend comme valeurs un numéro d'élément de la liste des éléments qui composent le tableau qu'elle indice. Ainsi ayant déclaré PILE(20), quand j'écris PILE(p), p ne peut plus prendre que l'une des valeurs 1, 2, 3, ... , 20 et c'est tout.

La réalisation

Quelle que soit la déclaration que j'observe, je considère les variables les unes après les autres, dans l'ordre où elles se présentent. L'exemple cité donne:

PH, ID, RES, x, p, r, i, PILE, C.

or, en procédure formelle, la configuration comporte d'abord 3 cases à usage de la machine universelle, puis ensuite viennent les paramètres etc. J'établis la correspondance la plus simple qui soit:



Au premier identificateur correspond la case 3 dont le contenu, a1 est une adresse adéquate mise en place, puisqu'il s'agit d'un paramètre, par la machine universelle pendant le déroulement de l'insertion. Les identificateurs ID et RES, les deux autres paramètres, sont traités de la même manière, on les fait correspondre aux cases 4 et 5 dont les contenus a2 et a3 sont également chargés pendant l'insertion.

Les variables x, p, r, et i qui jouent le rôle d'index pour le LAC, ont leurs images, les cases 6, 7, 8, et 9 qui sont des variables locales pour la procédure formelle image. Les valeurs b1, b2, b3 et b4 dont on verra plus loin comment les calculer sont mises en place quand la machine universelle déroule l'instruction de déclaration de la procédure image.

Il en est de même pour PILE et C.

Première approche

On peut trouver rapidement une solution simple pour la construction de l'image; J'imagine l'instruction:

S1 C := p + r ;

qui n'a aucun sens pour la procédure ANAL, mais qui est une instruction possible. Quelle pourrait en être l'image ? Or, à C correspond la case 11, à p la case 7 et à r la 8, si donc j'écris:

I1 $[_{11,0} := [_{7,0} + [_{8,0}$;

on constate que si la machine image avant le déroulement de cette instruction est dans le même état que la procédure symbolique juste avant le déroulement de S1, la variable 11 sera affectée de la même valeur que C aurait été affectée dans le déroulement de S1. Par même état j'entends: si valeur de p = x1 alors valeur de $[_{7,0}$ = x1 et si valeur de r = x2 alors valeur de $[_{8,0}$ = x2.
Pour que I1 soit immédiatement utilisable c'est le code correspondant qui doit être fabriqué:

(':=+',1,1,1),(11),(0),(7),(0),(8),(0)

chaque quantité entre-parenthèse représente un contenu de case, conformément à ce que je définirai au chapitre 8 de la machine universelle de la procédure formelle.

Variables indicées

La solution se complique pour les images des variables indicées. Prenons l'exemple d'instruction: RES(r) := PILE(p) ;
l'image de RES donne l'adresse d'accès à la première valeur du tableau, c'est la valeur elle-même de r qui va servir à calculer l'adresse de l'élément désigné. Il me faut pouvoir faire l'addition entre l'adresse qui correspond à RES et la valeur qui correspond à r. Pour réaliser ce calcul j'ai besoin d'une case spéciale. En fait, pour une instruction complète je pourrais avoir à faire au maximum trois calculs de cet ordre, pour une même instruction, puisque le LAC n'admet que des instructions à 3 opérandes au maximum.

Je vais donc choisir dans la configuration image, trois variables index au sens de la procédure formelle. Par exemple les cases 12, 13 et 14 qui sont les premières cases libres. Ce sont ces mêmes trois cases qui vont servir pour chacune des images des instructions symboliques.

Solution complète

En vertu de ces remarques si je prends une instruction qui n'existe pas dans l'exemple donné plus haut et qui n'aurait en plus aucun sens, mais qui représente une forme générale:
S2 PILE(p) := ID(i) + PH(x) ;
je serai amené à lui faire correspondre l'image:

G2

$[_{12}$:= $[_{10}$ + $[_{7,0}$;	
$[_{12}$:= $[_{12}$ - 1 ;	Image de PILE(p)
$[_{13}$:= $[_{4}$ + $[_{9,0}$;	
$[_{13}$:= $[_{13}$ - 1 ;	Image de ID(i)
$[_{14}$:= $[_{3}$ + $[_{6,0}$;	
$[_{14}$:= $[_{14}$ - 1 ;	Image de PH(x)
$[_{12,0}$:= $[_{13,0}$ + $[_{14,0}$;	Image de S2

Pour la clarté de l'exposé je présente les formes symboliques à construire, mais en fait ce sont les codes qui leur correspondent qui seront générés par le compilateur, comme nous allons le voir.

Je propose au lecteur attentif le petit exercice ainsi conçu:
on se donne S2, on se donne un jeu de valeurs pour les variables ID(i) et PH(x), de telle sorte qu'on puisse calculer PILE(p). Dans la configuration image on place dans les cases 3 à 11 des valeurs d'adresses qui permettent d'accéder à toutes les variables dont font aussi partie les tableaux, et on se donne pour les images de ID, de i, de PH, de x et de p les mêmes valeurs que pour S2, on déroule alors le jeu G2, et on vérifie qu'on aboutit à la même valeur pour l'image de PILE(p) que pour PILE(p).

Les constantes

Dans une instruction symbolique une constante apparaît comme une variable de type particulier. C'est un objet qui a la dimension d'une variable, mais qui n'est pas déclarée. Il va falloir choisir une solution qui tient compte de ce fait, donc prévoir une place au moins momentanée pour la valeur.

Ainsi pour les deux instructions de notre exemple:

S3	x := 1 ;
S4	r := r + 4 ;

si les cases 12, 13 et 14 sont prévues pour les trois opérandes, je réserve les deux cases suivantes 15 et 16, pour mettre les valeurs des constantes éventuelles. De cette façon, pour S3, le 1 est placé en case 15 alors que la valeur adresse 15 est placée dans la case 13.

Pour S4, le 4 est placé en 16 pendant que 16 se trouve en case 14.

Exemple de code

Je vais donner l'exemple de code de la procédure ANAL pour les sept premières instructions:

('décl')(4)(A)(N)(A)(L)(3)(4)(2)(20)(1)	proc ANAL; par PH,ID,RES; ...
(':=',0,0,-)(12)(6)	x := 1 ;
(':=',0,4,-)(13)(15)	
(':=',0,4,-)(15)(1)	
(':=',1,1,-)(12)(0)(13)(0)	

(':=',0,0,-)(12)(7) (':=',0,4,-)(13)(15) (':=',0,4,-)(15)(1) (':=',1,1,-)(12)(0)(13)(0)	p := 1 ;
(':=',0,0,-)(12)(8) (':=',0,4,-)(13)(15) (':=',0,4,-)(15)(1) (':=',1,1,-)(12)(0)(13)(0)	r := 1 ;
(':=',0,0,-)(12)(9) (':=',0,4,-)(13)(15) (':=',0,4,-)(15)(1) (':=',1,1,-)(12)(0)(13)(0)	i := 1 ;
(':=+',0,0,1)(12)(5)(8)(0) (':=-',0,0,4)(12)(12)(1) (':=+',0,0,1)(13)(4)(9)(0) (':=-',0,0,4)(13)(13)(1) (':=',1,1,-)(12)(0)(13)(0)	RES(r) := ID(i) ;
(':=+',0,0,1)(12)(3)(6)(0) (':=-',0,0,4)(12)(12)(1) (':=',0,4,-)(13)(16) (':=',0,4,-)(16)(')') ('si=',1,1,-)(12)(0)(13)(0)(code GEN)	ACC : <u>si</u> PH(x) = ')' <u>vers</u> GEN ;

etc...

VOUZZAVEDIBISAR,

Les aventures de Savate Premier et son Ordinateur Chevelu.

La triste fin d'Ordinateur Chevelu.

Depuis la grandissime visite de Jean-Polsky, Savate Premier parlait ouvertement avec Dieu. Peut-être grâce aussi à Bernadette, qui en fut dans un temps à peine lointain, l'interlocutrice privilégiée. De toute manière, notre Prince de Gaullicoquie était de plus en plus sur de son fait. Il avait été dignement élu, ses promesse, il les avait tenues ... enfin comme on tient des promesses électorales. En faisant le contraire de ce qu'on a dit, mais c'est là la règle du jeu, et il s'y était scrupuleusement conformé. Aussi le Royaume contenait-il quelque chose de bien pourri, rien que de très normal. En connaissez-vous, vous, oui vous, des royaumes ou des républiques qui ne soient pas, peu ou prou pourris dans quelque coin? hein? alors!

Mais en son genre, Savate Premier avait à son avantage une réussite appréciable, aussi en était-il très justement fier. Au début de son règne, bien entendu, c'était dans le bas peuple surtout que se localisaient ses détracteurs. Et puis, peu à peu une sorte de grogne s'était mise à l'envelopper à la fois discrètement et indirectement. Il n'était pas la cible des critiques, voire des dénigrements, mais il souffrait que l'on s'attaquât à son Ordinateur Chevelu.

Et puis un jour, à la fois excédé et inspiré, d'aucuns disent même armé du Conseil d'En-Haut, il défit la Chambre des Représentants du Peuple. Sur qu'il était, qu'il allait s'en constituer une autre encore plus à sa botte. Ou peut-être même avec une arrière pensée encore plus diabolique, estimaient d'aucuns qui lui prêtaient des calculs machiavéliques, cherchant comme alternative une cohabitation qui lui permettrait de se mesurer avec son illustre, presque légendaire, prédécesseur. On l'en imaginait se pourléchant et se frottant les mains avec un dégagement de flammes et de vapeurs sulfureuses. On en frissonnait pour Bernadette.

L'appel aux urnes fit un véritable triomphe ... à l'opposition. Bien qu'Ordinateur Chevelu ait promis de rester au pouvoir après les élections. Décidément le Gaullicoque se révélait totalement dénué de reconnaissance. La Majorité, devenu brutalement minorité, et rejetée de facto dans l'opposition, s'y rencogna folle de rage. Quelques membres éminents en vinrent aux mains et se mordirent cruellement dans une ambiance chargée d'électricité. Quand soudain tout le monde se souvenant d'Ordinateur Chevelu, et comprenant que c'est à lui et à lui seul qu'on devait cette déchéance, il y eut volte face suivie d'une belle ruée. Dans un concert de hurlements sauvages on vit des pièces de toutes sortes voler dans l'air, une odeur âcre se dégageait du lieu de lynchage, une fumée noire et blanche jaillissait par saccades ... puis la foule se dispersa, quelques acharnés continuèrent à piétiner des débris de circuits électroniques qui jonchaient le sol. Et puis même ces derniers partirent tristes en courbant le dos. Ils allaient se rassembler de nouveau pour tenter de reformer un nouvel appui solide pour Savate Premier, qui estimaient-ils, en avait grand besoin.

Voila comment périt Ordinateur Chevelu. Sans fleurs ni couronnes. Paix à ses cendres. Décidément la Gaullicoquie n'aimait guère la beauté rigoureuse des chiffres.

Edmond Bianco

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