UNIVERSITÉ PARIS XI

U.E.R. MATHÉMATIQUE

91-ORSAY (FRANCE)

N° 30

LIMA ω

PAR DENIS FELDMANN C.N.R.S.

ET PATRICK MERISSERT-COFFINIERES C.N.R.S.

Note ELSTASM N° 3

0 4

N° 30

LIMA ω

PAR DENIS FELDMANN C.N.R.S.

ET PATRICK MERISSERT-COFFINIERES C.N.R.S.

Note ELSTASM N° 3 .

Université PARIS-SUD
Centre d'Orsay
91 - ORSAY

Le,27 janvier 1973

23078

FACULTE DES SCIENCES - ORSAY
BIBLIOTHEQUE
MATHEMATIQUES

NOTE N° 3

LIMA ω

par Denis FELDMANN C.N.R.S.

et Patrick MERISSERT-COFFINIERES C.N.R.S.

SOMMAIRE

I - Introduction

II- Le domaine des ordinaux transfinis

III- Les objectifs de LIMA ω

IV- Conventions linguistiques

V - Techniques utilisées

VI - Une "session" LIMA ω

VII- Conclusion

BIBLIOGRAPHIE

Appendice I - Rappels sur les fonctions d'Ackermann

Appendice II- Analyse de quelques fonctions typiques.

I- INTRODUCTION

Le travail qui est décrit dans ce rapport se situe dans le cadre d'un projet général consacré à la manipulation des symboles dans le cas où cette manipulation correspond à une activité de mathématiques pures.

Il s'agit, pour une série de disciplines, ou de fragments de discipline, de mettre au point des "packages", c'est à dire des groupes de programmes donnant à l'utilisateur la possibilité de faire effectuer automatiquement des calculs qui nécessiteraient sans cela un effort combinatoire considérable.

On est donc dans le domaine du calcul automatique, éventuellement de la simplification automatique, de la vérification automatique, mais certainement pas celui de la démonstration automatique.

L'effort d'organisation des données et de mise au point des algorithmes n'en est cependant pas négligeable et promet de porter des fruits lorqu'on abordera des entreprises plus audacieuses.

Le projet global a été décrit par P. BRAFFORT dans une note antérieure [1]. C'est ainsi que, dans une première phase actuellement en cours, il est prévu de développer un certain nombre de systèmes d'ambition limitée, permettant de mettre au point les techniques et d'en vérifier l'efficacité. Ces systèmes portent le nom général de LIMA (langages intéractifs de manipulation algorithmique).

LIMA ω est, parmi ces systèmes, celui qui se propose de manipuler les ordinaux transfinis (ou tout au moins certains d'entre eux). En fait on se propose dans LIMA ω, de constituer l'équivalent de ce que serait une "calculatrice de bureau" dont les arguments ne se contenteraient pas de parcourir les nombres entiers mais poursuivraient leur excursion beaucoup plus loin dans le transfini.

La réalisation qui est proposée ici possède des caractéristiques qui lui sont propres, tant sur le plan du domaine d'application choisi que sur celui des techniques mise en oeuvre.

Elle possède cependant un point essentiel qui est en commun avec les autres systèmes LIMA déjà réalisés où en voie de développement; c'est le choix du "support informatique". Nous avons en effet choisi d'utiliser la notation D'IVERSON [2]

et le langage de programmation, "APL", où cette notation s'est incarnée. Ce langage, notamment sous sa forme récente, APL/360 [3], nous offre à la fois :

- la précision d'un système de notation rationnel
- l'élégance et la concision d'un langage algorithme avancé
- Les commodités d'une organisation conversationnelle bien conçue

II- LE DOMAINE DES ORDINAUX TRANSFINIS.-

A) rappel historique

En 1872, les travaux de G. CANTOR sur les séries trigonométriques l'amènent à un premier essai de classification des ensembles "exceptionnels" qui se présentent dans cette théorie, au moyen de la notion d'ensemble "dérivé". Ayant réussi à obtenir des résultats nouveaux et surprenants (comme celui de l'équipotence de R et de R^n), il se consacre entièrement à la théorie des ensembles, et, entre 1878 et 1884, aborde dans une série de mémoires [4] les problèmes d'équipotence et la théorie des ensembles ordonnés ; en 1882, sa découverte des ensembles bien ordonnés lui permet enfin d'itérer "transfiniment" la formation des ensembles dérivés, et d'aborder l'étude détaillée des nombres cardinaux.

B) Ordres et types d'ordre.

La définition de CANTOR d'un type d'ordre est "ce qui reste d'un ensemble quand on ne considère que l'ordre dans lequel sont disposés les éléments, sans tenir compte de leurs autres propriétés" (Ceci constitue le premier niveau d'abstraction", noté par le surlignage d'une barre ; le second est obtenu en faisant abstraction également de l'ordre, et il ne reste plus alors que le "nombre" (transfini) d'éléments de l'ensemble).

En langage moderne, on définit un ordre par la donnée d'une relation transitive et antisymétrique ; deux ensemble ordonnés ont même type d'ordre s'il existe une bijection conservant l'ordre entre eux ; le type d'ordre est la "classe d'équivalence" pour la relation : avoir même type d'ordre. Ainsi, le type d'ordre de N se note ω , on note ω^* celui de N muni de l'ordre inverse, et par extension des opérations qui seront définies plus bas sur les ordinaux, le type d'ordre de Z est $\omega^* + \omega$.

C) Bons ordres et ordinaux

Le problème d'une itération "au delà de l'infini" d'un processus quelcon-

que demande plus qu'un simple éclaircissement du concept d'ordre. Il faut pouvoir prolonger la strucutre des entiers naturels, seule à permettre le concept de récurrence. Une analyse précise des propriétés demandées montre qu'une extension de l'axiome de Peano est suffisante : on dit qu'un ordre est un <u>bon ordre</u> si tout sous ensemble non vide possède un plus petit élément pour cet ordre (c'est à dire, en langage axiomatique : E est bien ordonné par $<$ $\Longleftrightarrow (\forall A \subset E)(A \neq \emptyset \Longrightarrow (\exists x \in A)(\forall y \in A)(x < y)))$.

On appelle <u>ordinal</u> le type d'ordre d'un ensemble bien ordonné.

Exemples : d'après les axiomes de Peano, N est bien ordonné par l'ordre usuel ; l'ordre "défini par : $3 < 4 < 5 < 6 \ldots < 1 < 2$ est aussi un bon ordre sur N (d'ordinal $\omega+2$.)Z n'est pas bien ordonné par l'ordre usuel, car il n'a pas de plus petit élément.

D) <u>Propriétés élémentaires des ensembles bien ordonnés.</u>

On appelle segment d'un ensemble ordonné E toute partie S de E telle que $x \in S$, $y \in E$ et $> y$ entrainent $y \in S$. Tout segment d'un ensemble bien ordonné est cet ensemble, ou de la forme $]\leftarrow, \alpha[$, c'est à dire l'ensemble des y strictement plus petits que α . On montre alors qu'étant donnés deux ensembles d'ordinaux distincts, l'un est isomorphe à un segment de l'autre, et que ceci est une relation d'ordre (total) entre ordinaux, qui est même une relation de bon ordre sur un ensemble d'ordinaux.

On pourrait espérer ordonner ainsi l'ensemble de tous les ordinaux, mais on arrive alors à une contradiction : le type d'ordre de cet ensemble est encore un ordinal, et on voit qu'il est plus grand que tout ceux de l'ensemble, or il en fait partie !. Ce paradoxe, remarqué en 1897 par BURALI-FORTI, était le premier de ceux qui allaient, au début du 20ème siècle, menacer les fondements des mathématiques et amener une crise qui ne se résoudrait provisoirment qu'après les axiomatisations précises de ZERMELO-

FRAENKEL, GODEL-BERNAYS et VON NEUMANN dans les années 30 ; dans ces systèmes, les paradoxes sont évités en restreignant la notion d'ensemble, et l'ensemble "paradoxal" de tous les ordinaux "n'existe pas".

Le but de la définition des ensembles bien ordonnés était de pouvoir généraliser les raisonnements par récurrence ; cela est obtenu par les deux "principes de récurrence tranfinie" suivants (dont la démonstration est élémentaire) :

1.- Si R est une relation sur l'ensemble bien ordonné E telle que "R vraie sur tous les éléments plus petits que x "entraine" R vraie sur x" , R est vraie sur tout E .

2.- Si F est un autre ensemble, g une application de $P(F)$ dans F , il existe une application f et une seule de E dans F telle que pour tout $x \in E$, $f(x) = g(\{f(y) \mid y<x\})$. (définition de f par récurrence transfinie.).

En pratique, on applique ces deux principes en distinguant trois cas :

- Vérification pour le plus petit élément de E
- Vérification pour les "successeurs" : x tels qu'il existe un plus grand y strictement plus petit qu'eux ; dans ce cas, la vérification est en général dépendante uniquement de y .
- Vérification pour les autres éléments de E , dits "limites" ; dans ce cas on doit en général utiliser les propriétés de tout le segment des éléments plus petits.

E) <u>L'ensemble des ordinaux dénombrables</u> .-

Le théorème de ZERMELO (1904), permet de mettre un bon ordre sur tout ensemble ; on obtient ainsi, en partant par exemple de R , un ordinal non dénombrable, et on voit aisément que tout ordinal dénombrable est plus petit ; l'ensemble des ordinaux dénombrables est donc légal (mais non dénombrables) ; on le note souvent Ω_1 . Dans cet ensemble, on peut (à l'aide du deuxième principe de récurrence transfinie) définir une arithmétique des ordinaux, qui se généraliserait d'ailleurs aux ordinaux non dénombrables.

Tout d'abord, tout ordinal a un successeur, le plus petit des ordinaux qui soient plus grands que lui. On note $\alpha+1$ le successeur de α. Si on met "bout à bout" les ordinaux α et β, on obtient un ensemble bien ordonné, dont on note l'ordinal $\alpha+\beta$; cette addition est associative, mais non commutative, ainsi $1+\omega=\omega$, mais $\omega+1>\omega$. Si $\alpha_1<\alpha_2<\alpha_3<\ldots\ldots$ est une suite croissante d'ordinaux, il y a un plus petit ordinal plus grand que que tous les éléments de la suite ; cet ordinal est dit limite de la suite α_n. L'addition est "continue" à droite (mais pas à gauche) :

$$\lim_{n\to\infty}(\alpha+\beta_n)=\alpha+\lim_{n\to\infty}\beta_n.$$

Cette remarque permet, comme dans N, de définir une gamme de fonctions arithmétiques par récurrence transfinie : on définira la multiplication par les relations : $\alpha\times 0=0$; $\alpha\times(\beta+1)=(\alpha\times\beta)+\alpha$, et $\alpha\times(\lim_{n\to\infty}\beta_n)=\lim_{n\to\infty}(\alpha\times\beta_n)$ (dont on voit qu'elles couvrent bien les trois cas du paragraphe précédent). On définit de même l'exponentiation par $\alpha^0=1$, $\alpha^{\beta+1}=\alpha^\beta\times\alpha$ et $\alpha^{\lim_{n\to\infty}\beta_n}=\lim_{n\to\infty}(\alpha^{\beta_n})$.

Toutefois, l'arithmétique des ordinaux (dénombrables ou non) a un comportement inhabituel ; les techniques de calcul en sont d'autant compliquées. Par exemple,

$\alpha\times(\beta+\gamma)=(\alpha\times\beta)+(\alpha\times\gamma)$, mais en général $(\alpha+\beta)\times\gamma\neq(\alpha\times\gamma)+$
$+(\beta\times\gamma)$, $\alpha\beta\neq\beta\alpha$; le comportement de l'exponentiation est très complexe [5].

F) Les ordinaux constructifs

Pour manipuler commodément ces ordinaux, et pouvoir calculer au sens algorithmique du terme, il faut en avoir une représentation en entiers, c'est à dire un codage (on sait depuis GODEL passer d'une formule (chaine de caractères) à un entier et réciproquement ; il suffit donc de représenter un ordinal par une formule caractéristique).

Les ordinaux pour lesquels il existe un tel système de représentation (les couvrant eux et les ordinaux plus petits) sont dits constructifs, une notion introduite par

CHURCH et KLEENE [6].

Un exemple d'un tel système de notation est fourni par LIMA ω ; chaque ordinal est représenté sous la forme d'un polynôme exponentiel en ω. (La justification en sera donnée au chapitre suivant). La théorie développée par KLEENE montre qu'on peut aller beaucoup plus loin (en particulier en utilisant des fonctions d'ACKERMANN généralisées) mais qu'il n'est pas possible de noter tous les ordinaux dénombrables (ce qui est évident si on songe qu'il n'y a qu'un nombre dénombrable de notations disponibles, et une infinité non dénombrable d'ordinaux à noter) ; il existe néanmoins un système uniforme de notation de tous les ordinaux constructifs.

FACULTE DES SCIENCES BIBLIOTHEQUE MATHEMATIQUES ORSAY

III- OBJECTIFS DE LIMA ω

LIMA ω donne à l'utilisateur les moyens d'effectuer des calculs et des manipulations, analogues à celles qu'autorise un langage de programmation tel que A.P.L., mais sur des nombres ordinaux tranfinis suffisamment "petits". Que veut dire ici "suffisamment petits" ? Il s'agit d'ordinaux résultant de l'évaluation d'expressions comprtant un nombre fini de signes et où figurent les entiers, la lettre ω, les parenthèses, et les symboles $+$, $\times$, $*$, c'est à dire, en fait, les trois premières fonctions d'ACKERMANN (cf annexe 1) des ordinaux. On sait qu'il existe une forme canonique, dite forme normale de CANTOR, définie de la façon suivante Pour tout ordinal α, il existe de façon unique une suite finie d'ordinaux $\beta_1, \ldots$ β_k et des entiers $n_1, \ldots, n_k$ tels que

(i) $$\alpha = (\omega^{\beta_1} \times n_1) + \ldots + (\omega^{\beta_k} \times n_k)$$

(ii) $$\beta_1 > \beta_2 \ldots \beta_k$$

(iii) $$n_i > 0 \quad \forall i$$

Bien sur, on a la relation $\alpha \geqslant \beta_1$, mais l'inégalité stricte, comme nous le verrons, n'est pas toujours vraie.

L'existence de cette forme découle de l'existence d'une division euclidienne dans les ordinaux (voir plus loin).

Donnons l'allure de la démonstration :

Soit α un ordinal. L'ensemble des ordinaux β tels que $\alpha \geqslant \omega^{\beta}$ possède un plus grand élément β_1. L'existence d'une division donne :

$$\alpha = \omega^{\beta_1} \times q + r_1 \qquad \text{avec} \quad r_1 < \omega^{\beta_1}$$

q est un entier (si $q > \omega$, $\alpha \geqslant \omega^{\beta_1 + 1}$, contradiction)

si r_1 est non nul on peut recommencer :

$$\alpha = \omega^{\beta_1} \times n_1 \ldots + \omega^{\beta_\ell} \times n_\ell + r_\ell \qquad r_\ell < \omega^{\beta_\ell} \leqslant r_{\ell-1} .$$

Il reste à prouver que r_ℓ est nul pour un ℓ donné, ce qui est évident puisqu'il ne peut pas y avoir de suite finie strictement décroissante d'ordinaux.

Il est bien sûr, possible de recommencer l'opération sur les β_i non nuls. Si

cette récurrence s'arrête au bout d'un nombre fini d'itérations, l'ordinal considéré est bien "suffisamment petit" au sens cherché. Une source évidente d'ennuis apparaît avec les ordinaux ε tels que $\varepsilon = \omega^{\varepsilon}$. Soit ε_0 l'ordinal le plus petit ayant cette propriétés. Pour un ordinal $\alpha < \varepsilon_0$, la condition (ii) peut s'écrire

$$\alpha > \beta_1 > \beta_2 \ldots\ldots\ldots\ldots > \beta_k \ .$$

Si un tel ordinal α, 'était pas "suffisamment petit" il, existerait une suite infinie (α_i) telle que :

$$\alpha_0 = \alpha$$

α_{i+1} apparaît dans la forme normale de cantor de α_i, ce qui entraine $\alpha_i > \omega^{\alpha_{i+1}}$

Mais pour $\alpha < \varepsilon_0$, ceci entraîne $\alpha_i > \alpha_{i+1}$, et une telle suite infinie strictement décroissante ne peut exister. Nous avons maintenant déterminé sur quels ordinaux nous travaillons. Il reste à savoir quels calculs seront effectués. Les comparaisons sont bien sûr obligatoires, mais le principal reste le calcul arithmétique.

Addition, multiplication, et exponentiation, sont définies par récurrence tranfinie ; la soustraction par

$$\forall \lambda < \mu \ , \exists ! \ \nu \ \text{ tel que } \ \lambda + \nu = \mu$$

(l'existence se démontre aisément par la considération des α tels que $\lambda + \alpha \leqslant \mu$, et l'unicité vient de ce que l'addition est régulière <u>à droite</u>)

La division par :

$$\forall \lambda , \mu > 0 \ \exists ! \ \eta, \xi \ \text{ tels que } \ \lambda = (\mu \times \eta) + \xi \ \text{ et } \ \xi < \mu \ .$$

Là encore, la considération des α tels que $\mu \times \alpha \leqslant \lambda$ fournit la démonstration.

<u>Propriétés des opérations sur les ordinaux</u>

Soient :

$$\alpha = \omega^{\beta_1} n_1 + \ldots\ldots + \omega^{\beta_k} n_k ,$$

$$\alpha' = \omega^{\beta'_1} n'_1 + \ldots\ldots + \omega^{\beta'_{k'}} n'_{k'}$$

Pour simplifier l'e xpression des algorithmes qui suivent, le dernier terme de la forme de CANTOR représentera toujours la partie entière, même si elle est nulle.

La condition $n_L \neq 0$ pour $1 < L < k$

devient donc $n_L \neq 0$ pour $1 < L < k$

a) Addition

$$\lambda + (\mu + \nu) = (\lambda + \mu) + \nu \quad ; \quad \lambda + 0 = 0 + \lambda = \lambda$$

Soit i le plus grand indice tel que $\beta_i > \beta'_1$, 0 s'il n'y en a pas (nous conviendrons qu'une somme indexée sur l'ensemble vide est nulle). Alors :

$$\alpha + \alpha' = \omega^{\beta_1} n_1 + \ldots\ldots + \omega^{\beta_i} n_i + \omega^{\beta'_1} n'_1 + \ldots\ldots + \omega^{\beta'_{k'}} n'_{k'} \quad \text{si } \beta_i > \beta'_1$$

$$\omega^{\beta_1} n_1 + \ldots\ldots + \omega^{\beta_i}(n_i + n'_1) + \omega^{\beta'_2} n'_2 + \ldots + \omega^{\beta'_{k'}} n'_{k'}$$

$$\text{si } \beta_i = \beta'_1$$

b) Multiplication

$$0 \times \lambda = \lambda \times 0 = 0 \quad ; \quad 1 \times \lambda = \lambda \times 1 = \lambda \quad ;$$

$$\lambda(\mu\nu) = (\lambda\mu)\nu \quad ; \quad \lambda(\mu + \nu) = \lambda\mu + \lambda\nu$$

Si $\alpha > 0$:

$$\alpha \times \alpha' = \omega^{\beta_1 + \beta'_1} n'_1 + \ldots\ldots + \omega^{\beta_1 + \beta'_{k'-1}} n'_{k'-1} + \omega^{\beta_1} n_1 n'_{k'} + \omega^{\beta_2} n_2 + \ldots\ldots$$

$$+ \omega^{\beta_k} n_k \quad \text{si } n'_{k'}$$

$$\omega^{\beta_1 + \beta'_1} n'_1 + \ldots\ldots + \omega^{\beta_1 + \beta'_{k'-1}} n'_{k'-1} \quad \text{si } n'_{k'} = 0$$

c) Exponentiation :

$$\lambda^0 = 1 \quad ; \quad 1^\lambda = 1 \quad ; \quad 0^\lambda = 0 \quad \text{si } \lambda > 0 \quad ; \quad \lambda^1 = \lambda \quad ;$$

$$\lambda^\mu . \lambda^\nu = \lambda^{\mu+\nu} \quad ; \quad (\lambda^\mu)^\nu \quad . \text{ Si } \alpha > 1 \quad , \quad \alpha' > 0 \text{ on a :}$$

$$\alpha^{\alpha'} = \omega^{(\omega^{(-1)+\beta'_1} n'_1 + \ldots\ldots + \omega^{(-1)+\beta'_{k'-1}} n'_{k'-1})} n_{k'} \quad \text{si } \beta_1 = 0$$

$$\omega^{\beta_1 \omega^{\beta'_1} n'_1 + \ldots\ldots + \beta_1 \omega^{\beta'_{k'-1}} n'_{k'-1}} \alpha^{n'_{k'}} \quad \text{si } \beta_1 > 0$$

$$\text{où } \alpha^{n'_{k'}} = \omega^{\beta_1 n'_{k'}} n_1 + \omega^{\beta_1(n'_{k'-1})+\beta_2} n_2 + \ldots + \omega^{\beta_1(n'_{k'-1})+\beta_{k-1}} n_{k-1} + \omega^{\beta_1(n'_{k'-1})} n_1 n_k$$

$$+ \omega^{\beta_1(n'_{k'-2}) + \beta_2} n_2 + \ldots\ldots + \omega^{\beta_{k-1}} n_{k-1} + \omega^{\beta_k(n'_{k'-1})} n_k \text{ . Si } n_k > C$$

$$\omega^{\beta_1 n'_{k'}} n_1 + \omega^{\beta_1(n'_{k'-1}) + \beta_2} n_2 + \ldots\ldots + \omega^{\beta_1(n'_{k'-1}) + \beta_{k-1}} n_{k-1} \quad \text{si } n_k = C$$

d) Soustraction

Si $\lambda < \mu$ il existe un unique ν tel que

$$\lambda + \nu = \mu$$

Ce ν est noté $(-\lambda) + \mu$. On a $(-\mu) + \mu = 0$; On suppose que $\alpha' < \alpha$. Soit i le plus grand indice tel que $(\beta'_i , n'_i) < (\beta_i, n_i)$. On a alors

$$(-\alpha') + \alpha = \omega^{\beta_i} n_i + \omega^{\beta_{i+1}} n_{i+1} + \ldots + \omega^{\beta_k} n_k \quad \text{si } \beta'_i < \beta_i$$

$$\omega^{\beta_i}(n_i - n'_i) + \omega^{\beta_{i+1}} n_{i+1} + \ldots + \omega^{\beta_k} n_k \quad \text{si } \beta'_i = \beta_i$$

e) Division euclidienne

Soit $\mu > 0$. Il existe η, ξ tels que

$$\lambda = \mu\eta + \xi \text{ avec } \xi < \mu$$

On pose $\nu = [\lambda/\mu]$, $\xi = \mu|\lambda$

$$[0/\mu] = 0 \; ; \; \mu|0 = 0 \; ; \; [\lambda/1] = \lambda \; ; \; 1|\lambda = 0 \; ;$$

si $\lambda < \mu$ on a $[\lambda/\mu] = 0$, $\mu|\lambda = \lambda$

Supposons $0 < \alpha' < \alpha$ et écrivons

$$\alpha = \alpha'\eta + \xi$$

Comme dans a) soit i le plus grand indice tel que $\beta_i > \beta'_1$.

Alors :

e_1) Si $\beta_i > \beta'_1$ on a $\eta = \omega^{(-\beta'_1)+\beta_1} n_1 + \ldots\ldots + \omega^{(-\beta'_1)+\beta_i} n_i$

$$\xi = \omega^{\beta_{i+1}} n_{i+1} + \ldots + \omega^{\beta_k} n_k$$

e_2) Si $\beta_i = \beta'_1$. Faisant la division euclidienne ordinaire de n_i par n'_1 on peut écrire

$$n_i = n'_1 q_i + r_i \quad \text{où} \quad 0 < r_i < n'_1$$

On a alors

$e_{2.1}$) Si $r_i > 0$ ou si $Q_i = 0$ on a

$$\eta = \omega^{(-\beta'_1)+\beta_1} n_1 + \ldots\ldots + \omega^{(-\beta'_1)+\beta_{i-1}} n_{i-1} + \omega^{(-\beta'_1)+\beta_i} q_i$$

$$\xi = \omega^{\beta_i} r_i + \omega^{\beta_{i+1}} n_{i+1} + \ldots\ldots + \omega^{\beta_k} n_k$$

$e_{2.2.}$) Si $r_i = 0$, $q_i \neq 0$ et $(\omega^{\beta'_2} n'_2 + \ldots\ldots + \omega^{\beta'_{k'}} n'_{k'}) = (\omega^{\beta_{i+1}} n_{i+1} + \ldots + \omega^{\beta_k} n_k)$

alors $\eta = \omega^{(-\beta'_1)+\beta_1} n_1 + \ldots\ldots + \omega^{(-\beta'_1)+\beta_{i-1}} n_{i-1} + \omega^{(-\beta'_1)+\beta_i} nq_i$

$$\xi = (-(\omega^{\beta'_2} n'_2 + \ldots + \omega^{\beta'_{k'}} n'_{k'})) + \omega^{\beta_{i+1}} n_{i+1} + \ldots + \omega^{\beta_k} n_k)$$

$e_{2.3}$) Si $r_i = 0$, $q_i \neq 0$ et $(\omega^{\beta'_2} n'_2 + \ldots\ldots + \omega^{\beta'_{k'}} n'_{k'}) > (\omega^{\beta_{i+1}} n_{i+1} + \ldots + \omega^{\beta_k} n_k)$

alors

$$\eta = \omega^{(-\beta'_1)+\beta_1} n_1 + \ldots\ldots + \omega^{(-\beta'_1)+\beta_{i-1}} n_{i-1} + \omega^{(-\beta'_1)+\beta_i} (q_i - 1)$$

$$\xi = \omega^{\beta_i} n'_1 + \omega^{\beta_{i+1}} n_{i+1} + \ldots + \omega^{\beta_k} n_k .$$

Dan les formules précédentes de e_2) on peut évidemment ommettre $\omega^{(-\beta'_1)+\beta_i}$ puisque $\beta_i = \beta'_1$. Nous l'avons conservé pour mieux montrer la structure du résultat

IV.- LES CONVENTIONS LINGUISTIQUES

a) l'alphabet.

L'alphabet de LIMA ω n'utilise que le symbole supplémentaire ω (qui figure déjà sur le clavier standard APL), et outre les lettres (non soulignées) et les chiffres, les symbolrs opératoires suivants :

+ ⊖ × ⍟*⍳ = ≠>< ; les deux affectations ← et : ; le ⎕ ; et les parenthèses. (Tout autre symbole provoque le diagnostic LIMA : CHARACTER ERROR .).

b) Lexique.

L'écriture en APL, d'un compilateur traitant des noms de variables arbitraires est rendu presque impossible par le fait qu'on ne peut modifier une instruction en APL (tant que "dequote" n'est pas mis en service).

Nous nous sommes donc restreints, pour LIMA ω , à des noms de variables ne comportant qu'une seule lettre ; on verra dans l'appendice que même avec cette restriction, les programmes manipulant les valeurs des variables (STORE et VVALUE) utilisent beaucoup de place en mémoire.

C) Syntaxe.

En accord avec l'orientation générale du projet LIMA, la syntaxe de LIMA est identique à celle d'APL, c'est à dire qu'il n'y a pas de hiérarchie des opérateurs, et que l'analyse se fait de la droite vers la gauche. L'analyseur syntaxique correspondant est très simple, mais le traitement des parenthèses nécessitant la construction de listes (files d'attente), qu'APL manipule très mal, on a préféré sur ce point appeler récursivement l'analyseur. L'évaluation des expressions est d'ailleurs faite silmutanément, puisque l'analyse est "linéaire".

Néanmoins, il a semblé utile de donner la possibilité à l'utilisateur de mettre ces ordinaux sous forme normalisée directement. Il en avertit le système par le symbole : (par exemple S : ω (ω(2) + ω × 3+1)) et l'ordinal n'est pas calculé, mais directement codé et rangé.

La syntaxe de ces messages spéciaux n'est pas la syntaxe APL ; et un petit

analyseur se charge de la vérifier (Par contre, si la forme entrée n'était pas en fait normalisée, le système ne s'en rend pas compte).

d) Sémantique

La sémantique de LIMA ω est analogue à celle d'APL : les variables ont des valeurs (sinon leur emploi donne le diagnostic VVALUE ERROR) qui sont des ordinaux ; les chiffres désignent les entiers (qui sont aussi des ordinaux), et ω désigne ω!. Les symboles opératoires sont clairs, sauf peut être A|B, représentant le reste dans la division de B par A . ⎕ est utilisé comme en APL, en entrée pour réclamer une valeur (le système imprime ⎕ :, et une nouvelle expression doit être entrée); en sortie, pour faire imprimer un résultat intermédiaire.

e) Pragmatique

On désigne ainsi la partie du langage qui permet à l'utilisateur de communiquer avec l'organisation interne du système. Comme en APL, cela est obtenu en LIMA ω au moyen de "commandes-systèmes", reconnues par l'analyseur à ce qu'elles commencent par une parenthèses fermante (par exemple)VARS). Les commandes-système de LIMA ω sont)VARS, qui donne la liste des noms de variables utilisés,)CLEAR, qui vide cette liste, et)ERASE suivi de noms de variables, qui ne supprime que ces dernières.

V.- TECHNIQUES UTILISEES

Nous avons reporté en Annexe II, les listings commentés correspondants à certaines des fonctions essentielles qui composent le "workspace" LIMA ω. Nous voulons ici préciser l'organisation générale qui régit l'articulation de ces fonctions et préciser certaines techniques que nous avons utilisées.

L'esprit général est le suivant : comme dans le système APL, et d'une façon plus générale, dans les sytèmes conversationnels, on doit disposer d'un programme-maître jouant le rôle de "superviseur", qui se préoccupe des questions d'intendance" et appelle des programmes esclaves.

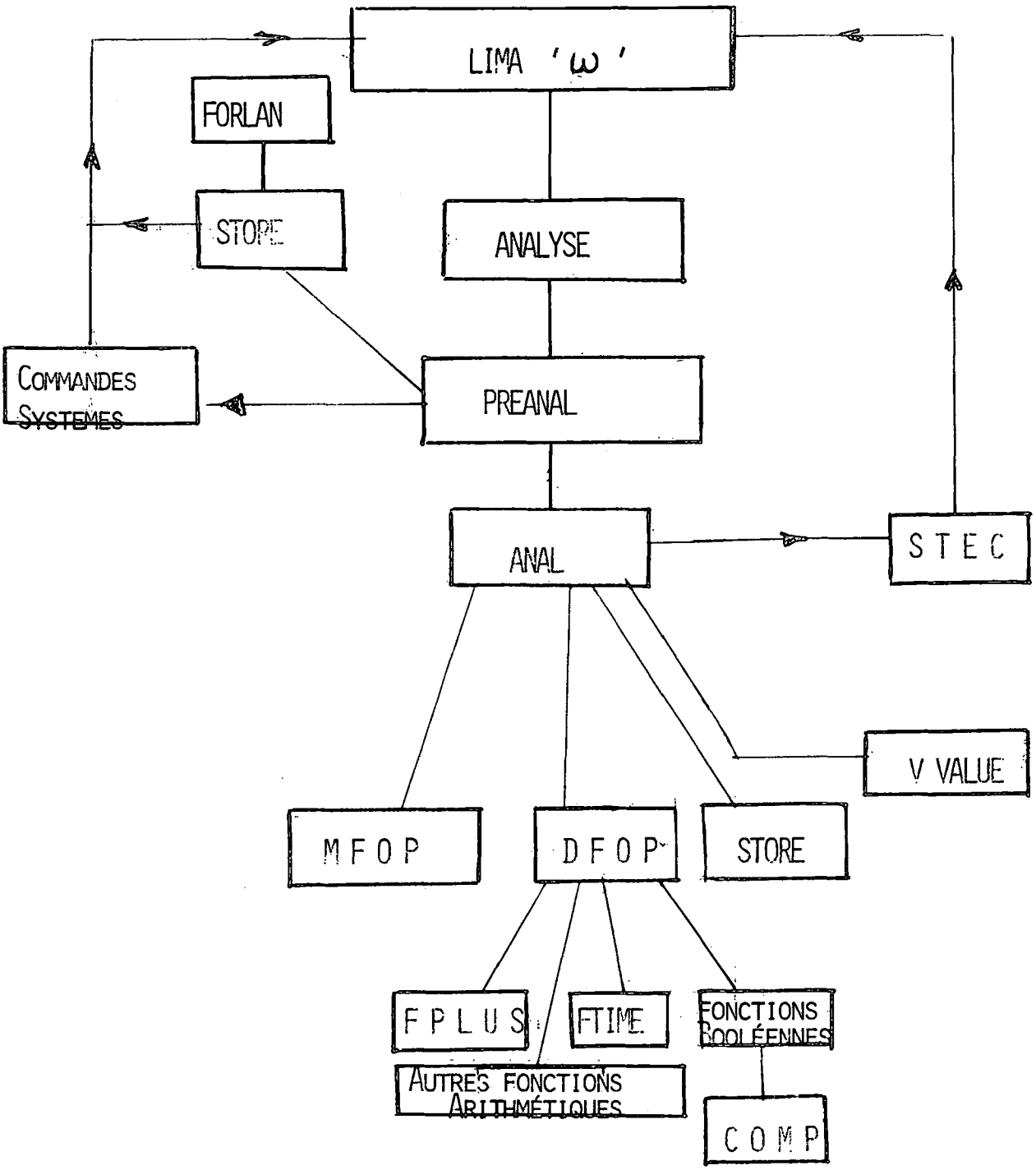
Parmi ceux-ci, il en est évidemment un qui joue le rôle périodique, c'est le programme "d'analyse".

Ce programme procède à l'analyse syntaxique des "lignes" c'est à dire des expressions introduites par l'utilisateur, donc dans le cas qui nous occupe ici d'expressions arithmétique comportant des ordinaux transfinis.

Puis l'analyse effectuée, on passe la main à des "routines sémantiques" qui font le travail d'évaluation, en tenant compte, évidemment des particularités de l'arithmétique transfinie.

Enfin, ce travail achevé, le contrôle repasse au superviseur après, éventuellement, l'impression des résultats, ou la quête d'informations complémentaires.

Bien entendu, toute action incorrecte, ou toute erreur d'écriture de l'utilisateur donne lieu à des messages d'erreurs.



PREANAL	traite les cas particuliers : Commandes systeme, rentrée sous forme normale de CANTOR (FORLAN)
MFOP	traite(rait) les opérateurs monadiques
DFOP	traite(rait) les opérateurs dyadiques
STORE	intervient sur chaque affectation et insère une valeur dans la table de variables (ex $\underline{A} \leftarrow \omega+1$)
VVALUE	va chercher une valeur dans cette table, chaque fois qu'on appelle une variable (ex.: $\underline{A}$ +2)
STEC	est le programme d'impression.

L'organigramme ci-contre donne une idée de la structure interne du système :

<u>ANALYSE</u> est le superviseur, qui s'empare de la ligne, élimine les blancs en trop, puis appelle :
<u>PREANAL</u>, qui recherche les lignes spéciales :

a) COMMANDES SYTEME, identifiées par)
PREANAL appelle alors une routine spéciale pour chaque COMMANDE SYSTEME

b) Commentaires : Identifiés par ρ, et qui ne sont évidemment pas traités.

c) l'affectation spéciale ':' pour la forme normale de CANTOR, dont nous avons déja parlé
PREANAL appelle alors FORLAN, qui code directement l'ordinal

<u>ANAL</u> . C'est un programme qui lit la ligne caractère par caractère à partir de la droite, et appelle, suivant le caractère rencontré, des sous programmes dont les principaux sont :

DFOP intervient avec les opérateurs binaires : c'est simplement un programme de branchement sur l'une des (actuellement 12) fonctions correspondant aux dits opérateurs ($+ \times \ominus$ ÷ $*$ | $\rangle\langle = \neq \rangle \langle$) . Elles sont de 2 types : arithmétiques (F PLUS, F TIME, etc....) ou la comparaison (à valeurs Booléennes) le sous-programme de base de celles-ci est <u>COMP</u>

MFOP joue éventuellement le même rôle pour les opérateurs monadiques.

STORE intervient avec toute affectation (← <u>et</u> : , il est donc éventuellement appelé par PREANAL), il rentre une valeur dans la table de variables

VVALUE intervient chaque fois qu'on tape un nom de variable non suivi d'une affectation : il va chercher la valeur correspondante dans la table des variables.

Après la sortie d'ANAL, ANALYSE appelle éventuellement STEC, programme d'impression. Les listings des programmes dont les noms sont soulignés, sont donnés en appendice. Il est indispensable, pour permettre de les suivre, de donner quelques renseignements, sur la représentation interne, ainsi qu'une description de certains des algorithmes utilisés :

a) Représentation interne :

Elle suit l'écriture de la forme normale de CANTOR :

1) 0 est représenté par le vecteur : ,0

2) $\omega^{\alpha} \times n$ pour $n > 0$ est représenté par (A,S,B,n) si S est la représentation de α ; A et B étant deux nombres choisis une fois pour toutes, en dehors du domaines des entiers admis, et représentant respectivement les parenthèses ouvrante et fermante.

3) l'ordinal $\underbrace{\omega^{\beta_1} \times n_1 + \ldots\ldots + \omega^{\beta_k} \times n_k}_{\alpha} + \underbrace{\omega^{\gamma_1} \times m_1 + \ldots + \omega^{\gamma_\ell} \times m_\ell}_{\beta}$

avec $\beta_1 > \beta_2 \ldots > \beta_k > \gamma_1 \ldots\ldots > \gamma_\ell$ et n_i , $n_j \neq 0$

est représenté par (S,T) si α est représenté par S, et β par T .

Exemples : un entier n , considéré comme $\omega^0 \times n$, est représenté par $(A,0,B,n)$. ω est représenté par $(A,1,B,1)$.

Choix de A et B : Une propriété importante de la représentation interne est qu'elle doit permettre de comparer rapidement deux ordinaux. Pour que l'ordre des ordinaux coïncide avec l'ordre lexicographique sur les représentations, il suffit de prendre pour A un nombre M plus grand que tout entier admis par LIMA ω , et pour B, -1 (plus petit que tout le reste).

b) Algorithmes utilisés pour le calcul arithmétique des deux premières opérations

I Addition :

La remarque de base utilisée pour l'addition est :

$$(\omega^{\alpha} \times n) + (\omega^{\beta} \times m) = \omega^{\alpha} \times n + \omega^{\beta} \times m \quad \text{si} \quad \alpha > \beta$$
$$= \omega^{\alpha} \times (n+m) \quad \text{si} \quad \alpha > \beta$$
$$= \omega^{\beta} \times m \quad \text{si} \quad \alpha < \beta$$

Il résulte alors de l'associativité de l'addition, et de l'unicité de la forme normale de CANTOR que l'addition de deux ordinaux sous cette forme s'obtient en justaposant les formes, et en "oubliant" les termes de la première d'exposants

inférieure au plus grand exposant de la seconde

II.- Mutiplication

$$\alpha = \omega^{\beta_1} \times n_1 + \ldots\ldots\ldots\ldots + \omega^{\beta_k} \times n_k$$

$$\alpha' = \omega^{\beta'_1} \times n'_1 + \ldots\ldots\ldots\ldots + \omega^{\beta'_\ell} + n'_\ell$$

On utilise la distributivité à droite pour se ramener aux problèmes de $\alpha \times N$ (partie entière éventuelle de α') et $\alpha \times \omega^{\beta}$ $(\beta \neq 0)$

Par associativité, on se ramène à $\alpha \times \omega$. Ceci est la limite (récurrence transfinie de

$$\underbrace{\alpha + \alpha \ldots\ldots\ldots\ldots + \alpha}_{N \text{ fois}} \quad \text{quand } N \text{ "tend vers" } \omega$$

c'est à dire $(\omega^{\beta_1} \times n_1) \times \omega + \omega^{\beta_2} \times n_2 \ldots\ldots\ldots\ldots + \omega^{\beta_k} \times n_k$

et bien sûr $\omega^{\beta_1} \times \omega = \omega^{\beta_1 + 1}$

Le résultat final est :

$$\omega^{\beta_1 + \beta'_1} \times n'_1 \ldots\ldots\ldots + \omega^{\beta_1 + \beta'_{\ell-1}} \times n'_{\ell-1} + \omega^{\beta_1} \times n_1 n'_\ell + \omega^{\beta_2} \times n_2 \ldots + \omega^{\beta_k} \times n_k$$

si $\beta_\ell = 0$, c'est à dire α' a une partie entière

$\omega^{\beta_1 + \beta'_1} \times n'_1 \ldots\ldots + \omega^{\beta_1 + \beta'_\ell} \times n'_\ell$ si $\beta_\ell \neq 0$, c'est à dire α' n'a pas de partie entière)

Les algorithmes des trois autres opérations arithmétiques sont insuffisamment "parlants" pour mériter d'être exposés ici.

C) Problèmes liés aux entrées-sorties

Quand l'utilisateur tape un ordinal, que ce soit en système APL, ou sous la forme canonique, la représentation exponentielle serait la plus agréable, mais elle est impraticable, les claviers ne permettant pas de remonter les lignes, de revenir en arrière, etc...

Par contre, l'ordinateur, lui, peut très bien commencer par imprimer tous les exposants puis les ω. C'est ce format, d'ailleurs assez spectaculaire en action, qui a été adopté en sortie (en fait on ne fait pas imprimer des lignes à la suite les unes des autres, mais des matrices de caractère).

VI.- UNE SESSION LIMA ω .-

```
⍝ EXEMPLE DE SESSION LIMA ω
ω
ω
ω+1
ω+1
1+ω
ω
⍝JUSQU'ICI,CHAQUE LIGNE TAPEE PAR L'UTILISATEUR EST SUIVIE
⍝D'UNE REPONSE DE LIMA ω; MAIS, COMME POUR APL,L'AFFECTATION N'EST
⍝PAS SUIVIE D'UNE REPONSE DU SYSTEME:
A←ω+1
A×A
 2
ω  +ω+1
⍝CE QUI PRECEDE ILLUSTRE NOTRE TECHNIQUE DE REPRESENTATION DE
⍝L'ELEVATION A UNE PUISSANCE.POUR DEFINIR DIRECTEMENT
⍝LA MEME VALEUR,ON POURRAIT ECRIRE SOIT(NOTATION APL):
A←(ω*2)+ω+1
⍝EN EFFET:
A
 2
ω  +ω+1
⍝                                        SOIT(FORME DE CANTOR):
A:ω(2)+ω+1
⍝EN EFFET:
A
 2
ω  +ω+1
B←A×A
A>B
0
A<B
1
A*B
 4   3   2        4   3   2        4   3   2
ω  +ω  +ω  +ω+2  ω  +ω  +ω  +ω+1  ω  +ω  +ω  +ω
ω                +ω                +ω
)VARS
A B W
⍝W EST UNE VARIABLE RESTEE DE LA DERNIERE SESSION.QUE VAUT+IL?
W
ω
⍝NOUS N'EN AVONS PAS BESOIN,EFFACONS LE:
)ERASE W
)VARS
A B
⍝POUR TOUT EFFACER:
)CLEAR
)VARS
```

```
A←⎕*⎕
  ⎕:
ω
  ⎕:
ω+1
  A
 ω
ω
  ⍝CECI ETAIT UN EXEMPLE D'UTILISATION DU QUAD D'ENTREE;
  ⍝ILLUSTRONS LE QUAD DE SORTIE:
  B←⎕←A*A+1
 ω
  ω +ω
ω
  ⍝CALCULS DIVERS:
ω+ω=ω
  ω+1
ω|(ω*2)+1
  1
((ω+2
    ∨
  *2)+1)÷ω
  ω
  ⍝EXEMPLES D'ERREURS:
  ⍝(CHAQUE DIAGNOSTIC NOUS FAIT SORTIR DE LIMA)
D
  VALUE ERROR
D
  ∧
      LIMA 'ω'
  ω∧1
  SYMBOL ERROR:
  ω∧1
   ∧
      LIMA 'ω'
ω*(W+
    ∨
   ω+1))
  PARENTHESIS ERROR:
ω*(ω+1))
       ∧
      LIMA'ω'
 ←ω+1
  AFFECTATION ERROR:
 ←ω+1
  ∧
```

```
      LIMA 'ω'
A←ω+1
B←(ω*2)+1
A⊖1
ω+1
B⊖A
 2
ω  +1
A⊖ω
1
⍝EXEMPLE DE MANIPULATIONS D'ORDINAUX 'ENCOMBRANTS':
B←
      ∨
    :ω(ω(ω(ω(ω×8+2)×8+ω(ω8+
                          ∨
                          ×8+1)×8+ω(ω×8)×8)×8)×8)×8
B
        ω×8+2       ω×8+1       ω×8
      ω        ×8+ω        ×8+ω        ×8
    ω                                  ×8
   ω                                      ×8
  ω                                          ×8
A←ω*ω*ω
B*A
        ω×8+2       ω×8+1       ω×8
      ω        ×8+ω        ×8+ω        ×8        ω
    ω                                  ×8+ω
   ω
  ω                                          ×8


B*B
        ω×8+2       ω×8+1       ω×8
      ω        ×8+ω        ×8+ω        ×8
    ω                                  ×8
   ω                                      ×8
  ω                                          ×8
 ω                                              ×8
```

VII.- CONCLUSION.-

Le système que nous venons de décrire, comme nous l'avions déjà indiqué dans l'introduction, transforme un terminal APL en machine à calculer de bureau pour les ordinaux transfinis, (une machine à calculer un peu plus puissante qu'à l'accoutumée, puisqu'en plus de calculs immédiats, il est possible d'afficher des valeurs à des variables).

Plus précisément on a donc un langage qui fonctionne comme APL, mais où l'ensemble de nombres entiers n'est plus l'ensemble des entiers finis, mais l'ensemble des ordinaux $< \varepsilon_0$.

A vrai dire, nous ne pouvons, dans la version actuelle de LIMA ω , que manipuler des ordinaux scalaires et non pas des vecteurs, des matrices d'ordinaux, etc.....

Cependant cette extension serait possible à peu de frais. Nous nous en sommes abstenus pour gagner du temps, mais aussi parce que ce type de structure a été peu utilisé, jusqu'à présent, par les mathématiciens.

De même, la version actuelle ne prévoit qu'un type de représentation interne pour les ordinaux.

Dans la mesure où il existe aujourd'hui une "clientèle" non négligeable pour la manipulation des ordinaux constructifs,

il pourrait être intéressant de développer d'autres types de représentation interne (KLEENE CHURCH, SCHUITE, TAKENTI, etc...). Dans ce cas, on pourrait ajouter à LIMA ω d'autres fonctions primitives du type (APL) et ("encode" et "décode") permettant le passage systématique d'une représentation à une autre.

Bien entendu, d'autres fonctions primitives "scalaires" d'APL pourraient, sans difficulté, être introduite :

! (factorielle)

$\lfloor$, $\lceil$ (minimum et maximum), etc.

mais dont le sens, pour des ordinaux transfinis, demande évidemment à être précisé.

En un mot, la syntaxe de notre système est claire et définitive : c'est pratiquement celle d'APL. la sémantique, liée à la nature même des ordinaux transfinis pourrait être étendue sans difficultés.

C'est l'aspect pragmatique qui reste inachevé. En particulier toute erreur commise par l'utilisateur fait sortir du superviseur LIMA ω et l'oblige à une réinitialisation du système.

Dans l'ensemble, tant que les tâches demandées ne nécessitent pas d'intervention humaine, ne serait-ce que parce que le programme peut simplifier les expressions qu'on lui soumet de façon optimale, un langage de type APL est parfaitement adapté à la partie calcul du projet LIMA. Cette expérience ne procure par contre rien en ce qui concerne les problèmes de vérifications de démonstrations.
Le problème des entrées sorties, sans être grave, montre néanmoins les limites du clavier. Il faudra sans doute envisager l'utilisation d'écrans cathodiques.

Enfin, les temps de calcul sont énormément allongés par le passage à travers deux compilateurs, ce qui s'aggravera encore pour des systèmes LIMA comprenant un mode définition.

La réécriture générale prévue dans la phrase 3 en langage machine peut donc d'ores et déjà être considérée comme indispensable.

BIBLIOGRAPHIE

[1] P. BRAFFORT.- Déclarations et initialisations.- Note ECSTASM N°1 Orsay 1972

[2] K. IVERSON.- A programming language .- John WILEY 1962

[3] A.D. FAIKOFF and K.IVERSON.- APL/360 User's manuel.- IBM H 2060683-1962

[4] G. CANTOR .- Contributions to the foundation of the theory of transfinite numbers.- DOVER 1955(reed)

[5] W. SIERPINSKI.- Cardinal and ordinal numbers.- PWN. 1965

[6] A. CHURCH and S. KLEENE.- Fundamente Mathematicae 28 11, 1937

[7] A. ACKERMANN.- Math. Annalen , 99 118; 1928

APPENDICE I.-RAPPEL SUR LES FONCTIONS D'ACKERMANN

L'idée même de fonction d'ACKERMANN" est due en réalité à HILBERT(1925) dans son effort (inachevé) pour démontrer l'hypothèse du continu.

Il s'agit de donner un procédé d'engendrement d'opérations arithmétiques (fonctions primitives dyadiques scalaires -dans le langage Iversonien) et qui soient les "itérées" les unes des autres.

On obtient ainsi une suite de fonctions indéxées par les entiers naturels: Si ACK est le nom générique de cette suite on posera :

ACK [1] identique à + $+$

ACK [2] identique à $\times$

ACK [3] identique à * (exponentiation)

ACK [4] est parfois nommée "tetration". On la représente parfois sous forme d'une "exponentiation gauche"

X ACK [4] Y étant écrit sous la forme ^{Y}X

On a X ACK[n] Y identique à X ACK[n-1] (ACK[n-1] ACK[n-1] X

(avec la convention syntaxique d'APL (associativité vers la gauche)

si X figure Y fois dans la formule.

Il existe une définiton <u>récursive</u> non <u>primitive</u> de <u>la</u> fonction d'ACKERMANN (dans <u>trois</u> variables n , x , y).

$$ACK\ (1,0,y) = y$$

$$ACK(1,x+1,y) = ACK\ (1,x,y)+1$$

$$ACK\ (2,0,y) = 0$$

$$ACK\ (n+1,0,y) = 1$$

$$ACK\ (n+1,x+1,y) = ACK(n,ACK(n+1,x,y),y)$$

Les fonctions d'ACKERMANN permettant d'exprimer simplement des ordinaux transfinis très grands.

C'est ainsi que LIMA ω ne manipule que des ordinaux inférieurs à

$$\varepsilon_0 = \omega^{\omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}}$$

ω fois

On a évidemment $\varepsilon_0 = \omega$ ACK [4] ω

ou encore $\varepsilon_0 = \omega$ ACK[5] 2

APPENDICE II. : ANALYSE

DE QUELQUES FONCTIONS TYPIQUES

```
      ∇ANALYSE[⎕]∇
    ∇ Z←ANALYSE V
[1]   Z←0
[2]   V←,V
[3]   V←(V≠' ')/V
[4]   PREANAL V
[5]   →(U=1)/0
[6]   V←SUPPAR V
[7]   Z←ANAL V
[8]   →((ρV)=1)/PR
[9]   →(V[2]≠'←')/PR
[1]]  →0
[11]  PR:STEC Z
    ∇
```

V est la ligne de caractères tapés par l'utilisateur.

suppression des blancs

traitement des cas spéciaux(voir plus loin)

←V est l'indicateur avertissant que PIANAL à rencontré une situation à période

Suppression des parenthèses inutiles.

Traitement du cas général.

Sortie sans impression si la ligne"débutait" par une affectation.

impression.

```
      ∇PREANAL[⎕]
    ∇ PREANAL V
[1]   U←0
[2]   →((ρV)=1)/TEST
[3]   →(V[2]≠':')/TES
[4]   →(~V[1]∊ALPH)/ERR
[5]   (ALPHιV[1]) STORE FORCAN 2↓V
[6]   U←1
[7]  TEST:V←,V
[8]   →0
[9]  TES:→(T1,T2,0)[')⍳'ιV[1]]
[10] T1:→((ρV)≠5)/CER
[11]  →(~∧/(V=')VARS'))/CES
[12]  VARS
[13]  →0,U←1
[14] CER:→((ρV)≠6)/CET
[15]  →(~∧/(V=')CLEAR'))/CES
[16]  CLEAR
[17]  →0,U←1
[18] CET:→((ρV)≤6)/CES
[19]  →(~∧/((6↑V)=')ERASE'))/CES
[20]  →(~∧/(6↓V)∊ALPH)/CES
[21]  ERASE ALPHιδ↓V
[22]  →0,U←1
[23] CES:' INCORRECT COMMAND'
[24]
[25]  →
[26] T2:U←1
[27]  →0
[28] ERR:'SYNTAX ERROR'
[29]  V
[30]  '∧'
[31]  →
    ∇
[32]
```

Affectation spéciale immédiatement codée et rentrée indicateur allumé.

Sortie en cas normal

COMMANDE SYSTEMS

Indicateur Allumé

Sortie défausse

COMMANDE

Commentaire indicateur allumé.

```
      ∇ANAL[⎕]∇
    ∇ Z←ANAL V;N;VAL;CODE;K
[1]   V←,V
[2]   VAL←0
[3]   N←ρV
[4]   CODE←1                                        comme au dernier
[5]   →LABEL[PERM⍳V[N]]                             caractère
[6]  ERREUR:'SYMBOL ERROR:'                         sortie d'erreur de
[7]   V                                             caractère
[8]   (N-1)ρ' ';'∧'
[9]   →
[10] LETTRE:VAL←(VVALUE ALPH⍳V[N]) DFOP VAL         nom de variable
[11]  →BOUCLE
[12] CHIFFRE:VAL←(N NSEGMCH V) DFOP VAL             variable numérique
[13]  N←UTR
[14]  →BOUCLE
[15] OMEGA:VAL←VALOM DFOP VAL                       'ω'
[16]  →BOUCLE
[17] MONADIC:VAL←V[N] MFOP VAL                      opérateur monadique
[18]  →BOUCLE
[19] FPARENT:N←(K←N) OPAR V                         parenthèse fermée
[20]  VAL←(ANAL N+(K-1)↑V) DFOP VAL                 appel récursif
[21]  →BOUCLE
[22] QUAD:→(N=ρV)/INQU                              :
[23]  →(V[N+1]='←')/OUTQU                           quad d'entrée
[24] INQU:'⎕:'
[25]  VAL←(ANAL⎕) DFOP VAL
[26]  →BOUCLE
[27] DYADIC:CODE←CODE⍳V[N]                          opérateur dyadique*
[28] BOUCLE:→(0=N←N-1)/FIN                          fin de boucle
[29]  →5
[30] OUTQU:STEC VAL                                 quad de sortie(impressi
[31]  CODE←1
[32]  →BOUCLE
[33] AFFECT:→(N=1)/ERR                              '→': affectation
[34]  →(V[N-1]='⎕')/AFQU
[35]  →((K←ALPH⍳V[N-1])=27)/ERR                     recherche de la variabl
[36]  K STORE VAL                                   entrée de la valeur dans
[37]  N←N-1                                         la table .
[38]  CODE←0
[39]  →BOUCLE
[40] ERR:'AFFECTATION ERROR:'
[41]  V
[42]  (N-1)ρ' ';'∧'
[43]  →
[44] FIN:Z←VAL
[45]  →(CODE≤1)/0
[46]  'SYNTAX ERROR:'
[47]  V

[49]  →
[50] AFQU:N←N-1                                     quad de sortie
[51]  →OUTQU
    ∇
```

A chacune des étiquettes (lettre, chiffre, etc...) ANAL "met à jour" la valeur courante (VAL) CODE indique que l'opérateur dyadique en valeur (s'il y en a un) ce qui détermine le branchement dans OLOP.

```
      ∇FTIME[⎕]∇
    ∇ R←V FTIME W;A;B;C;N;S;T;X
[1]    →(V[1]≠0)/SU1                    si V=0
[2]    R←,0                                V×W = 0
[3]    →0
[4]   SU1:→(W[1]≠0)/SU2                 si W = 0
[5]    R←,0                                V×W = 0
[6]    →0
[7]   SU2:A←1,V FIN 1                   S:= exposant du terme de plus haut degré de V
[8]    S←A SEL V                        si V est un entier(S=0)
[9]    →(S[1]≠0)/SU3
[10]   R←W
[11]   T←ENT W                          La partie entiere de W  est multipliée par  V
[12]   →(T=0)/SU4                       pour donner V × W
[13]   →((MAX>C←T×V[ρV]))/NO1
[14]   SOS
[15]   →0
[16]  NO1:R[ρR]←C
[17]  SU4:→0
[18]  SU3:N←V[A[2]+1]                   Si  V  n'est pas un entier  on applique
[19]   X←W                              l'algorithme page
[20]   R←¯1
[21]  DEBU:B←1,X FIN 1
[22]   T←R SEL X                              partie irrégulière de l'algorithme
[23]   →(T[1]≠0)/SU5                          correspondant à la partie entière de
[24]   →(MAX>C←X[B[2]+1]×N)/NO2               W
[25]   SOS
[26]   →0
[27]  NO2:R←R,INF,S,¯1,C,(A[2]+1)↓V
[28]   →0
[29]  SU5:R←R,INF,(S FPLUS T),¯1,X[B[2]+1]    partie régulière de l'agorithme
[30]   X←(B[2]+1)↓X
[31]   →((ρX)≠0)×DEBU
    ∇


      ∇FPLUS[⎕]∇
    ∇ R←V FPLUS W;A;B;C;S;T;X
[1]    →(W[1]≠0)/SU1                    si  W = 0
[2]    R←V                                 V × W = V
[3]    →0
[4]   SU1:→(V[1]≠0)/SU2                 si V = 0
[5]    R←W                                 V+W = W
[6]    →0
[7]   SU2:A←1,W FIN 1
[8]    T←A SEL W                        T = exposant du terme de plus
[9]    X←V                              haut degréde W.
[10]   R←W
[11]  DEBU:B←(X DEB(ρX)-1),(ρX)-1       Algoritüme (page  ) : on prend
[12]   C←B SEL X                        chaque terme de  V (par la fin)
[13]   →((C COMP T)= 1 0 ¯1)/(UN,DE,TR) on compare au terme de tête de W
[14]  UN:R←INF,C,¯1,X[ρX],R             plus grand : on juxtapose
[15]   →TR                              égal : on ajoute les coefficient
[16]  DE:→(MAX>S←R[A[2]+1]+X[ρX])/NO
[17]   SOS
[18]   →0
[19]  NO:R[A[2]+1]←S
[20]  TR:X←(B[1]-1)↑X
[21]   →DEBU×((ρX)≠0)                   plus petit : on l'oublie
    ∇
```

⍝ET VOICI LA BASE DES PROGRAMMES DE COMPARAISON(<≤>=≠):

```
      ∇COMP[⎕]∇
    ∇ R←V COMP W;X;Y
[1]   →((×(⍴V)-⍴W)= 1 0 ¯1)/(UN,DE,TR)
[2]  UN:X←V
[3]   Y←(⍴V)↑W
[4]   →FI
[5]  DE:X←V
[6]   Y←W
[7]   →FI
[8]  TR:X←(⍴W)↑V
[9]   Y←W
[10] FI:R←×1↑(X≠Y)/X-Y
    ∇
```

} ramène V et W à la même longueur

compare suivant l'ordre lexicographiques