# Contribution to formal calculation on computer

Yvon Siret

N° d'ordre

# THESES

présentées à

LA FACULTE DES SCIENCES DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR ES SCIENCES APPLIQUEES

par

**Yvon SIRET**

Ingénieur I.E.G.

Première thèse

# CONTRIBUTION AU CALCUL FORMEL SUR ORDINATEUR

Deuxième thèse

PROPOSITIONS DONNEES PAR LA FACULTE

Thèses soutenues le 1er Juillet 1970 devant la commission d'examen

| | | |
|---|---|---|
| MM. | J. KUNTZMANN | Président |
| | N. GASTINEL | |
| | J. CEA | Examinateurs |
| | L. BOLLIET | |

# L I S T E  D E S  P R O F E S S E U R S

Doyen honoraire : Monsieur M. MORET
Doyen : Monsieur E. BONNIER

## PROFESSEURS TITULAIRES

| | | |
|---|---|---|
| MM. | NEEL Louis | Physique Expérimentale |
| | KRAVTCHENKO Julien | Mécanique Rationnelle |
| | CHABAUTY Claude | Calcul différentiel et intégral |
| | BENOIT Jean | Radioélectricité |
| | CHENE Marcel | Chimie Papetière |
| | FELICI Noël | Electrostatique |
| | KUNTZMANN Jean | Mathématiques Appliquées |
| | BARBIER Reynold | Géologie Appliquée |
| | SANTON Lucien | Mécanique des Fluides |
| | OZENDA Paul | Botanique |
| | FALLOT Maurice | Physique Industrielle |
| | KOSZUL Jean-Louis | Mathématiques |
| | GALVANI Octave | Mathématiques |
| | MOUSSA André | Chimie Nucléaire |
| | TRAYNARD Philippe | Chimie Générale |
| | SOUTIF Michel | Physique Générale |
| | CRAYA Antoine | Hydrodynamique |
| | REULOS René | Théorie des Champs |
| | BESSON Jean | Chimie Minérale |
| | AYANT Yves | Physique Approfondie |
| | GALLISSOT François | Mathématiques |
| Melle. | LUTZ Elisabeth | Mathématiques |
| MM. | BLAMBERT Maurice | Mathématiques |
| | BOUCHEZ Robert | Physique Nucléaire |
| | LLIBOUTRY Louis | Géophysique |
| | MICHEL Robert | Minéralogie et pétrographie |
| | BONNIER Etienne | Electrochimie et Electrométallurgie |
| | DESSAUX Georges | Physiologie animale |
| | PILLET Emile | Physique Industrielle-Electrotechnique |
| | YOCCOZ Jean | Physique Nucléaire théorique |
| | DEBELMAS Jacques | Géologie Générale |
| | GERBER Robert | Mathématiques |
| | PAUTHENET René | Electrotechnique |
| | MALGRANGE Bernard | Mathématiques Pures |
| | VAUQUOIS Bernard | Calcul Electronique |
| | BARJON Robert | Physique Nucléaire |

| | | |
|---|---|---|
| MM. | BARBIER Jean-Claude | Physique |
| | SILBER Robert | Mécanique des Fluides |
| | BUYLE-BODIN Maurice | Electronique |
| | DREYFUS Bernard | Thermodynamique |
| | KLEIN Joseph | Mathématiques |
| | VAILLANT François | Zoologie et Hydrobiologie |
| | ARNAUD Paul | Chimie |
| | SENGEL Philippe | Zoologie |
| | BARNOUD Fernand | Biosynthèse de la Cellulose |
| | BRISSONNEAU Pierre | Physique |
| | GAGNAIRE Didier | Chimie Physique |
| Mme. | KOFLER Lucie | Botanique |
| MM. | DEGRANGE Charles | Zoologie |
| | PEBAY-PEROULA Jean-Claude | Physique |
| | RASSAT André | Chimie Systématique |
| | DUCROS Pierre | Cristallographie Physique |
| | DODU Jacques | Mécanique Appliquée I. U. T. |
| | ANGLES D'AURIAC Paul | Mécanique des Fluides |
| | LACAZE Albert | Thermodynamique |
| | GASTINEL Noël | Analyse numérique |
| | GIRAUD Pierre | Géologie |
| | PERRET René | Servo-mécanisme |
| | PAYAN Jean-Jacques | Mathématiques Pures |

PROFESSEURS SANS CHAIRE

| | | |
|---|---|---|
| MM. | GIDON Paul | Géologie |
| Mme. | BARBIER Marie-Jeanne | Electrochimie |
| Mme. | SOUTIF Jeanne | Physique |
| | COHEN Joseph | Electrotechnique |
| | DEPASSEL R. | Mécanique des Fluides |
| | GLENAT René | Chimie |
| | BARRA Jean | Mathématiques Appliquées |
| | COUMES André | Electronique |
| | PERRIAUX Jacques | Géologie et Minéralogie |
| | ROBERT André | Chimie Papetière |
| | BIARREZ Jean | Mécanique Physique |
| | BONNET Georges | Electronique |
| | CAUQUIS Georges | Chimie Générale |
| | BONNETAIN Lucien | Chimie Minérale |
| | DEPOMIER Pierre | Physique Nucléaire-Génie Atomique |
| | HACQUES Gérard | Calcul numérique |
| | POLOUJADOFF Michel | Electrotechnique |
| Mme. | KAHANE Josette | Physique |
| Mme. | BONNIER Jane | Chimie |
| MM. | VALENTIN Jacques | Physique |
| | REBECQ Jacques | Biologie |
| | DEPORTES Charles | Chimie |
| | SARROT-REYNAULD Jean | Géologie |
| | BERTRANDIAS Jean-Paul | Mathématiques Appliquées |
| | AUBERT Guy | Physique |

PROFESSEURS ASSOCIES

| | | |
|---|---|---|
| MM. | RODRIGUES Alexandre | Mathématiques Pures |
| | MORITA Susumu | Physique Nucléaire |
| | RADHAKRISHNA | Thermodynamique |

MAITRES DE CONFERENCES

| | | |
|---|---|---|
| MM. | LANCIA Roland | Physique Atomique |
| Mme. | BOUCHE Liane | Mathématiques |
| MM. | KAHANE André | Physique Générale |
| | DOLIQUE Jean Michel | Electronique |
| | BRIERE Georges | Physique |
| | DESRE Georges | Chimie |
| | LAJZEHOWICZ Joseph | Physique |
| | LAURENT Pierre | Mathématiques Appliquées |
| Mme. | BERTRANDIAS Françoise | Mathématiques Pures |
| MM. | LONGEQUEUE Jean-Pierre | Physique |
| | SOHM Jean-Claude | Electrochimie |
| | ZADWORNY François | Electronique |
| | DURAND Francis | Chimie Physique |
| | CARLIER Georges | Biologie végétale |
| | PFISTER Jean-Claude | Physique |
| | CHIBON Pierre | Biologie animale |
| | IDELMAN Simon | Physiologie animale |
| | BLOCH Daniel | Electrotechnique I. P. |
| | MARTIN-BOUYER Michel | Chimie (C. S. U. Chambéry) |
| | SIBILLE Robert | Construction mécanique (I. U. T.) |
| | BRUGEL Lucien | Energétique I. U. T. |
| | BOUVARD Maurice | HYdrologie |
| | RICHARD Lucien | Botanique |
| | PELMONT Jean | Physiologie animale |
| | BOUSSARD Jean-Claude | Mathématiques Appliquées (I. P. G.) |
| | MOREAU René | Hydraulique I. P. G. |
| | ARMAND Yves | Chimie I. U. T. |
| | BOLLIET Louis | Informatique I. U. T. |
| | KUHN Gérard | Energétique I. U. T. |
| | PEFFEN René | Chimie I. U. T. |
| | GERMAIN Jean-Pierre | Mécanique |
| | JOLY Jean-René | Mathématiques Pures |
| Melle. | PIERY Yvette | Biologie animale |
| | BERNARD Alain | Mathématiques Pures |
| | MOHSEN Tahsin | Biologie (C. S. U. Chambéry) |
| | CONTE René | Mesures Physiques I. U. T. |
| | LE JUNTER Noël | Génie Electrique Electronique I. U. T. |
| | LE ROY Philippe | Génie Mécanique I. U. T. |
| | ROMIER Guy | Techniques Statistiques quantitatives I. U. T. |
| | VIALON Pierre | Géologie |
| | BENZAKEN Claude | Mathématiques Appliquées |
| | MAYNARD Roger | Physique |

| | | |
|---|---|---|
| MM. | DUSSAUD René | Mathématiques (C. S. U. Chambéry) |
| | BELORIZKY Elie | Physique (C. S. U. Chambéry) |
| Mme. | LAJZEROWICZ Jeannine | Physique (C. S. U. Chambéry) |
| M. | JULLIEN Pierre | Mathématiques Pures |
| Mme. | RINAUDO Marguerite | Chimie |
| MM. | BLIMAN Samuel | E. I. E. |
| | BEGUIN Claude | Chimie Organique |
| | NEGRE Robert | I. U. T. |

MAITRES DE CONFERENCES ASSOCIES

| | | |
|---|---|---|
| MM. | YAMADA Osamu | Physique du Solide |
| | NAGAO Makoto | Mathématiques Appliquées |
| | MAREZIO Massimo | Physique du Solide |
| | CHEECKE John | Thermodynamique |
| | BOUDOURIS Georges | Radioélectricité |
| | ROZMARIN Georges | Chimie Papetière |

*Je remercie Monsieur Le Professeur* KUNTZMANN *de m'avoir fait l'honneur de présider le Jury de cette thèse.*

*Je tiens à exprimer ma reconnaissance à Monsieur Le Professeur* GASTINEL, *dont l'intérêt pour l'outil que constitue un ordinateur a toujours été pour moi une source d'encouragement et m'a donné l'occasion de lui être utile.*

*Cette reconnaissance va aussi à Monsieur* BOLLIET *Maître de Conférences qui m'enseigna jadis la programmation et fut tout au long de ces années celui qui m'incita et m'aida à aller de l'avant.*

*Monsieur* CEA, *professeur à la Faculté des Sciences de Rennes, a bien voulu me faire l'honneur de juger ce travail. Je lui exprime mes profonds remerciements.*

*Il m'est agréable de remercier publiquement Monsieur* André LAPLACE *assistant à la Faculté des Sciences, qui m'a aidé avec passion, opiniâtreté et esprit d'initiative.*

*Enfin, juste retour des choses, je ne saurais oublier ma femme* Liliane, *mes collègues et amis du Service de Programmation, dont la compétence, l'esprit de service, la critique pertinente c'est à dire bienveillante furent maintes fois mise à contribution : Messieurs* BELLOT, BELLINO, DU MASLE, DUPUY, HANS, LECARME, LE HEIGET, LUCAS, SAYA.

*Le service de dactylographie en la personne de Madame* CURTET, *et le service de tirage ont assuré avec leur soin coutumier la publication de cette thèse.*

*Au Père COUX*

"L'arithmomètre de M. Thomas peut servir à former des puissances et à extraire des racines. C'est donc la machine à calculer la plus parfaite qui ait été construite jusqu'ici. Cependant, malgré l'admiration que cet instrument excite, il est douteux qu'on en tire des avantages *pratiques* en proportion avec les efforts intellectuels qu'il a coûtés."

Article Machine à calculer. Page 711
Dictionnaire des Mathématiques Appliquées
Par H. SONNET
Librairie Hachette PARIS 1884

# TABLE DES MATIERES

## INTRODUCTION

Ce document est loin d'apporter une contribution fondamentale à l'informatique théorique ! Il présente plutôt quelques travaux effectués dans l'exploration des possibilités des ordinateurs pour le calcul formel. En ce sens, il est le contraire d'un aboutissement, tellement le domaine est vaste et les problèmes à résoudre nombreux.

Si du point de vue algorithmique, il n'y a pas de méthode générale, il semble que pour la programmation, une technique fasse l'unanimité : le traitement des listes. Par goût peut-être plus que par nécessité, j'ai choisi le langage qui formalisait le mieux cette technique, encouragé en cela par les réussites de ceux qui en la matière sont nos ainés : les universitaires d'outre-atlantique.

La disproportion entre le chapitre II et le chapitre III n'est qu'apparente.

Dans le chapitre II, j'ai voulu montrer un exemple parmi bien d'autres. Une foule de problèmes peut être abordée de cette façon, aussi bien de "vieilles questions" remises à l'honneur parce que l'on peut aller "plus loin", que des questions plus modernes. On pourra consulter à ce sujet le très bel article de Monsieur Jean Jacques Duby sur les vecteurs de Witt. Sans doute, pendant quelques années encore, la programmation de ces problèmes fera-t-elle appel à des gens de métier dont l'expérience leur permet de voir comment traduire pour une machine, et par une méthode efficace, les algorithmes des mathématiciens.

Le chapitre III semble plus technique. Le point de vue n'est pas le même puisqu'il s'agit de fabriquer un outil, utilisable de façon conversationnelle avec les restrictions que cela comporte.

Je demande beaucoup d'indulgence pour le décalage entre les espoirs impliqués par le nom que j'ai donné à ce système conversationnel et la réalisation !

Je suis conscient du fait que certaines améliorations sont à apporter tant du point de vue de la conception que de son implantation sur une machine.

Mais en fin de compte, j'ai l'espoir que les exigences des utilisateurs qui voudront bien fournir des problèmes réels, permettront de dépasser le stade des expériences dont le seul but est de montrer que la solution du problème est possible....

LISP ? *mais c'est très simple !*

# CHAPITRE I

## LE LANGAGE LISP

Le langage LISP a été développé par John Mac Carthy et son équipe, au M.I.T. entre les années 1958 et 1960. Très curieusement, il a pris la réputation d'un langage difficile, alors que la syntaxe des êtres qu'il permet de traiter est très simple et que les règles d'interprétation des programmes sont données par une fonction qui allie à la fois la concision et la rigueur.

Notre but n'est pas ici de décrire ce langage. Il existe d'excellents livres sur la question. Nous allons simplement en exposer les principes généraux, en considérant successivement les données, qui appartiennent à une classe d'expressions symboliques, et les fonctions de ces expressions.

### Les expressions symboliques

La classe des expressions symboliques, est définie comme suit :

<expression symbolique> ::= <atome> | (<expression symbolique>.<expression symbolique>)

<atome> ::= <lettre> | <atome><lettre> | <atome><chiffre>

EXEMPLES : M1, ABC, (A.Z), (A.((B.C).K2))

Par définition, les expressions symboliques qui ont la forme :

(<S>.(<S>.(<S>.(..... .(<S>.NIL).....))))

[<S> étant une expression symbolique quelconque et NIL un atome distingué]

sont appelées listes et notées

(<S><S><S>.....<S>) *(1)*

Par convention ( ) est une autre dénotation de NIL

EXEMPLES

| | | |
|---|---|---|
| (A) | représente | (A.NIL) |
| (A B C) | " | (A.(B.(C.NIL))) |
| (A(B D) Z) | " | (A.((B.(D.NIL)).(Z.NIL))) |

Fonctions d'expressions symboliques : le métalangage

Nous pouvons définir maintenant, une classe de fonctions d'expressions symboliques. Les expressions seront repérées par des variables écrites en petites lettres et la notation sera préfixée, comme dans
f[x] , g[x;h[y]]

a) Les fonctions primitives

Elles sont au nombre de 5 :

. Une fonction de construction (totale), notée cons

cons [x;y] = (x.y)

(La partie droite n'ayant de sens que si l'on remplace x et y par leur valeur effective).

. Deux fonctions de sélection (partielles, parce que non définies sur les atomes), notées car et cdr

Soit x = (<s1>.<s2>)
car [x] = <s1>
cdr [x] = <s2>

*(1)* NOTE : *Le caractère d'espacement joue le rôle de séparateur quand c'est nécessaire.*

Appliquées à une liste, ces deux fonctions donnent respectivement le premier élément et la liste amputée de son premier élément.

Soit x = (A B C)
car [x] = A
cdr [x] = (B C)

NOTE : *Les noms ésotériques car et cdr sont consacrés par l'usage et peuvent être compris comme tête et queue.*

. Deux prédicats : notés atom et eq

atom est une fonction totale ; eq est définie uniquement sur les atomes.

atom [x] = T si x est un atome
NIL autrement

eq [x;y] = T si x est le même atome que y
NIL autrement

T et NIL représentent donc entre autres choses les valeurs booléennes vrai et faux

b) Notion de forme

Une forme est une expression susceptible d'avoir une valeur. Une forme est simple ou conditionnelle.

forme simple : Une forme simple est une expression symbolique, une variable, ou une expression du genre f[x1;x2,...] où f est une fonction, les $x_i$ étant eux mêmes des formes (simples ou conditionnelles).

Pour évaluer une telle forme, on considère que les variables et leur valeur sont couplées dans une table et l'on applique le cas échéant les fonctions à leurs arguments évalués. Naturellement, si une forme est une expression symbolique, c'est à dire une constante, sa valeur est cette constante elle-même.

EXEMPLE avec la table :

| | |
|---|---|
| x | (A.B) |
| y | (C.D) |
| z | (B.E) |

car [(A.B)] = A

cons[car [x] ; cdr [y]] = (A.D)

eq [cdr [x] ; car [z]] = T

eq [atom[x] ; NIL] = T

Forme conditionnelle

Alors que les formes simples s'expriment avec un symbolisme usuel, une forme conditionnelle revêt l'aspect suivant :

$$[p_1 \rightarrow e_1 \; ; \; p_2 \rightarrow e_2 \; ; \ldots ; \; p_n \rightarrow e_n]$$

Pour en connaître la valeur, on évalue $p_1$. Si $p_1$ a pour valeur T, l'expression entière a pour valeur celle de $e_1$.

Si $p_1$ a pour valeur NIL, l'on recommence le processus avec le couple "$p_2 \rightarrow e_2$", et ainsi de suite.

Si aucun $p_i$ n'a pour valeur T, l'expression conditionnelle est indéterminée.

EXEMPLE

[eq[car[(A.B)];A] → cdr[(A.B)];T → NIL] = B

[NIL → T;T → NIL] = NIL

Dans la pratique, le dernier $p_i$ est souvent l'atome T.

c) Définition de nouvelles fonctions

Une forme n'est pas une fonction. Pour définir une fonction à partir d'une forme on utilise la notation dite λ de Church [ 9 ].

Soit une forme quelconque, contenant les variables $x_1, x_2, x_3, \ldots$, on construira une fonction par l'assemblage :

$$\lambda[[x_1;x_2;x_3;\ldots] ; \langle\text{forme}\rangle]$$

L'évaluation d'une forme telle que :

$$\lambda[[x_1;x_2;x_3;\ldots] ; \langle\text{forme}\rangle] \ [arg1;arg2;arg3;\ldots]$$

se fera en couplant dans une table les $x_i$ aux valeurs associées aux $arg_i$ et en évaluant <forme> relativement à cet environnement.

La forme conditionnelle, permet en particulier la définition récursive des fonctions : une fonction est dite définie récursivement si la forme définissante fait intervenir la fonction à définir.

La notation correcte d'une telle fonction est réalisée avec l'opérateur label (qui généralise en quelque sorte l'opérateur λ), utilisé de la façon suivante :

$$\text{label } [fct ; \lambda[[x_1;x_2;\ldots] ; \langle\text{forme contenant } x_1, x_2, \ldots fct\rangle]]$$

Pour appliquer une telle fonction à une liste d'arguments [arg1;arg2;...] il faudra

1) attacher à l'identificateur fct, la fonction

$$\lambda[[x_1;x_2;\ldots] ; \langle\text{ forme contenant } x_1, x_2, \ldots fct\rangle]$$

2) Evaluer la forme

$$\lambda[[x_1;x_2;\ldots] \; ; \; < \text{forme contenant } x_1, x_2, \ldots \text{fct}>][\text{arg1};\text{arg2};\ldots]$$

suivant les règles précédentes, avec la convention que chaque occurence de l'identificateur fct comme nom de fonction, doit être remplacée par la fonction qui lui est attachée.

EXEMPLE :

$$\text{label}[\text{dernier} \; ; \; \lambda[[x] \; ; \; [\text{eq}[\text{cdr}[x] \; ; \; \text{NIL}] \rightarrow \text{car}[x]; \; T \rightarrow \text{dernier}\; [\text{cdr}[x]]]]]$$

est la fonction qui appliquée à une liste, permet de sélectionner son dernier élément.

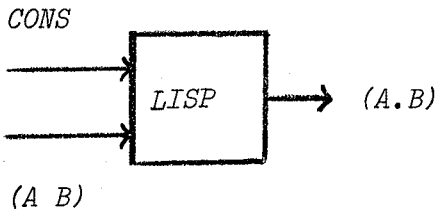
L'application des fonctions ainsi définies, est réalisée par un interprète universel [10], [11].

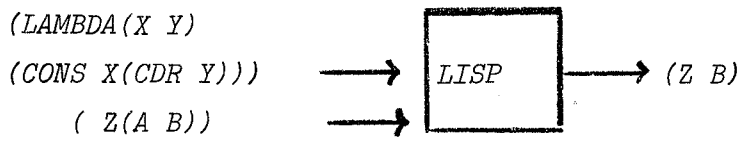
Moyennant une représentation des fonctions dans le langage même des données, l'interprète est une fonction qui vérifie l'équation:

$$\text{Interprete}\; [f^*; \text{arg1};\text{arg2};\ldots.] = f\; [\text{arg1};\text{arg2};\ldots.]$$

$f^*$ étant la traduction de f dans le langage des expressions symboliques.

EXEMPLE




Cependant, une telle utilisation est impraticable puisque théoriquement il faut écrire les fonctions chaque fois qu'on veut les appliquer.

D'où un grand nombre d'extensions qui rendent le système efficace sans que les concepts fondamentaux auxquels il se rattache, soient abandonnés.

Les extensions les plus importantes sont :

a) Les listes de propriétés des atomes

On peut attacher, par des fonctions internes, des propriétés utilisables de façon spécifique : valeur constante, λ-définition de fonction (cette deuxième possibilité dispense de l'usage de label) Un usage judicieux de ces listes de propriétés permet une programmation souple et efficace.

b) La forme programme

La forme définissante d'une fonction peut être conçue, non pas comme une seule expression, mais comme une liste de formes qui seront in-

terprétées comme des instructions. On peut par exemple conserver des valeurs intermédiaires par des instructions d'affectation et se transférer à des instructions étiquettées par des instructions de saut.

c) Les nombres

Les nombres sont introduits comme atomes particuliers et le système est pourvu des fonctions arithmétiques indispensables.

d) Les arguments fonctionnels

e) L'évaluation programmée des arguments

Il est possible de confier à une fonction, la liste de ses arguments non évalués. L'évaluation des arguments peut être faite dans le programme. Il s'agit en quelque sorte, d'offrir à l'utilisateur la possibilité de définir des formes (dites spéciales) analogues à la forme conditionnelle, ou des fonctions à nombre indéfini d'arguments. Pour évaluer, les arguments, le programmeur dispose des fonctions qui constituent l'interpréte. On retrouve alors dans un langage à haut niveau, la possibilité qu'a un programmeur en langage machine de construire dynamiquement une séquence de programme et de s'y transférer.

f) L'assembleur et le compilateur

Un système LISP étant implanté sur une machine, l'assembleur permettra d'écrire des sous programmes efficaces et le compilateur de traduire, en code symbolique pour l'assembleur, la définition d'une fonction.

L'existence de ces deux fonctions accroit considérablement la rapidité d'exécution de l'application des fonctions.

En conclusion, toutes ces extensions auxquelles s'ajoute un grand nombre de fonctions utiles, insérées à priori dans le système, font de LISP, un langage extrèmement puissant pour l'écriture des algorithmes traitant des objets que l'on peut décrire sous forme d'expressions symboliques.

*"La machine arithmétique fait des effets qui approchent plus de la pensée que tout ce que font les animaux : mais elle ne fait rien qui puisse faire dire qu'elle a de la volonté comme les animaux..."*

*Blaise PASCAL*

Les règles de traduction sont les suivantes :

| | | |
|---|---|---|
| <variable> | → | atome correspondant |
| exemple : x | → | X |
| <expression symbolique> | → | (QUOTE <expression symbolique>) |
| exemple : X | → | (QUOTE X) |
| fct [x;y;...] | → | (FCT X Y ...) |
| $[p_1 \rightarrow e_1; p_2 \rightarrow e_2; \ldots]$ | → | $(\text{COND}(p_1^* e_1^*)(p_2^* e_2^*)\ldots)$ |
| $\lambda[[x;y;\ldots]; e]$ | → | $(\text{LAMBDA}(X\ Y\ldots)\ e^*)$ |
| label [fct;λ...]] | → | (LABEL FCT (LAMBDA...)) |

L'interpréte s'exprime dans le langage des fonctions et c'est lui qui théoriquement donne les règles d'évaluation des formes et doit être programmé sur une machine.

Pour écrire un système LISP, il faudra donc :

1 - Convenir d'une représentation des expressions symboliques qui traduit leur structure parenthésée.

2 - Ecrire les sous-programmes correspondant aux primitives

3 - Ecrire la fonction interpréte en suivant les règles d'évaluation des formes.

Un tel système se présente alors comme un automate qui accepte en entrée sous forme d'expressions symboliques une fonction et la liste de ses arguments, et qui délivre en sortie, le résultat de l'application de la fonction à ses arguments.

# CHAPITRE II

## LISP utilisé comme langage algorithmique

### 1.1. INTRODUCTION

Dès son apparition, vers 1958, LISP a connu la faveur d'une foule de programmeurs intéressés par la résolution automatique de problèmes non numériques. Parmi les travaux les plus célèbres il faut citer le programme d'intégration formelle (SAINT) de Slagle [18] et celui plus récent de MOSES [19], le programme de simplification des formules de WOOLRIDGE [21]. En démonstration automatique, on ne saurait passer sous silence l'algorithme de WANG [14], pour le calcul des propositions. Parallèlement maints programmeurs se sont attaqués au calcul des prédicats du premier ordre.

Enfin, citons pour mémoire : des programmes pour le jeu d'échecs, la simulation de test d'intelligence [20], les systèmes de questions réponses [23], l'analyse syntaxique [22] etc...

Il est un domaine où ce langage a été particulièrement employé : celui du traitement formel des expressions mathématiques usuelles, représentées sous forme préfixée. Les cours de LISP, présentent de nombreux exemples qui sont devenus maintenant des exercices classiques pour les étudiants [17] (dérivation, simplification etc...)

Cependant, certains êtres mathématiques sont traités de façon particulière. C'est le cas des polynômes et des fractions rationnelles dans le système Mathlab [26] de Carl Engleman. Nous allons expliquer la méthode de représentation de ces êtres en mémoire et montrer un exemple d'algorithme utilisant cette représentation.

## 1.2. LES POLYNOMES DANS MATHLAB

### 1.2.1. Formalisme utilisé

Soit Z l'anneau de entiers rationnels et $Z[x]$ l'anneau des polynômes à une indéterminée (x) sur Z.

On peut former un anneau qui est une réalisation de $Z[x]$ (DUBREUIL [53])

Soit l'ensemble E des suites finies représentées dans le langage des listes par :

$$(a_n, a_{n-1}, \ldots, a_1, a_o) \qquad a_i \in Z$$

Soit P un élément de cet ensemble avec $a_n \neq 0$. On définira la classe d'équivallence de P, comme l'ensemble des suites obtenues à partir de P en ajoutant en tête de la liste autant de O (zéro de Z ) que l'on désire :

$$(0,0,a_n,a_{n-1},\ldots,a_1,a_o) \equiv (0,a_n,\ldots,a_1,a_o) \equiv (a_n,a_{n-1},\ldots,a_1,a_o)$$

Soit Q un autre élément de E

$$Q = (b_m, b_{m-1}, \ldots, b_i, b_o)$$

Supposons m > n

Posons :

$$P+Q = (b_m, b_{m-1}, \ldots, b_n+a_n, \ldots, b_1+a_1, b_o+a_o)$$

$$P.Q = (u_r, u_{r-1}, \ldots, u_1, u_o)$$

avec $u_i = a_o b_i + a_1 b_{i-1} + \ldots + a_i b_o$

muni de ces 2 lois l'ensemble E est un anneau, le neutre pour la loi additive étant (O) et celui pour la loi multiplicative étant (1)

Posons $X = (1\ 0)$

Alors $X^2 = (1\ 0\ 0)$ , $X^3 = (1\ 0\ 0\ 0)$ etc...

En convenant que $X^0 = (1)$, on peut écrire :

$$P = (a_n, a_{n-1}, \ldots, a_1, a_0) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0 X^0$$

c'est à dire $P = \Sigma\, a_i X^i$

L'application qui à x fait correspondre X et au polynôme de $Z[x]$, $\Sigma a_i x^i$ fait correspondre $\Sigma a_i X^i$, est un isomorphisme.

Or, d'un point de vue algorithmique, c'est à dire finalement sur une machine, les éléments de E sont très faciles à traiter.

L'intérêt du raisonnement précédent, réside dans le fait qu'il n'est nullement dépendant de Z, et qu'il va nous permettre d'exhiber sous forme de liste un anneau qui est une réalisation de

$$Z\,[x_1][x_2]\ldots\,[x_n].$$

La réalisation de $Z\,[x_1][x_2]\,\ldots\,[x_n]$ est l'ensemble des listes de la forme

$$(p_k, p_{k-1}, \ldots, p_1, p_0)$$

où chaque $p_i$ est un élément de l'anneau qui est une réalisation de

$$Z\,[x_1][x_2]\,\ldots\,[x_{n-1}]$$

et ainsi jusqu'à $Z\,[x_1]$.

Dans cet anneau l'élément unité est

$$((\ldots(((1)))\ldots))$$

n parenthèses

et le zéro :

$$((\ldots(((0)))\ldots))$$

n parenthèses

Le langage des données de LISP (listes) est donc parfaitement adapté à la représentation des éléments d'un anneau isomorphe à l'anneau des polynômes en n variables sur Z.

La représentation obtenue est dite récursive sur le nombre des variables. L'ordre des variables détermine la forme interne du polynôme, de façon unique.

Considérons le polynôme

$$x_1 x_2 x_3^2 + 4x_2 x_3 + 3x_1$$

Ecrivons l'ordre d'importance des variables sous la forme d'une liste $(x_1\ x_2\ x_3)$, les variables étant placées dans la liste par ordre d'importance croissante.

Le polynôme isomorphe est alors :

(((1 0)(0))((4)(0))((3 0)))

Si la liste d'importance est $(x_3\ x_2\ x_1)$, on obtient :

(((1 0 0)(3))((4 0)(0)))

La représentation récursive fait apparaitre parfois les zéros des anneaux $Z[x_1]$ , $Z[x_1][x_2]$,etc... c'est à dire des listes de la forme

((...(0)...))

Il est pratique pour aérer la notation, mais surtout pour faciliter les tests de programmation de remplacer ces sous listes par l'atome distingué NIL, qui symbolisera ainsi le polynôme nul en un nombre quelconque ($\neq 0$) de variables ou si l'on veut qui dénote les zéros des anneaux $Z[x_1]$, $Z[x_1][x_2]$ etc... le zéro de Z étant toujours noté 0.

Il existe dans Mathlab, une foule de fonctions opérant sur cette représentation des polynômes. On ne peut en donner ici une description. Disons simplement que la forme des êtres à traiter conduit tout naturellement à l'emploi systématique de sous programme récursifs. Le nom de ces sous-programmes évoque leur fonction :

POLPLUS, POLDIFFERENCE, POLMINUS, POLTIMES, POLGCD. etc...

Représentation des fractions rationnelles

Par convention, une fraction rationnelle sera représentée par un couple de 2 polynômes transformés avec la même liste de variables et tels que la fraction soit irréductible.

On obtient donc en machine la représentation de l'élément irréductible de la classe d'équivallence de la fraction considérée.
Comme cas particulier, un nombre rationnel sera représenté par un couple de deux entiers. [ex : 3/4 ↔(3.4)]

Mathlab contient de même, un ensemble de fonctions qui réalisent les opérations classiques :

PLUSF, DIFFERENCEF, MINUSF, TIMESF, INVERTF, POLQUOTIENT...

Toutes ces fonctions appellent fréquemment le sous-programme qui trouve le PGCD de deux polynômes de façon que les résultats intermédiaires et définitifs soient toujours mis sous forme irréductible.

Naturellement, il s'agit là d'une opération coûteuse mais nécessaire si l'on ne veut pas que les résultats intermédiaires soient envahissants.

NOTE : *Il est important de remarquer que la représentation récursive n'est pas restreinte aux polynômes et fractions rationnelles en des variables simples : toute expression rationnelle en 'certaines' sous-expressions formelles, peut se représenter sous forme récursive à condition de considérer ces sous-expressions comme des variables simples.*

PAR EXEMPLE :

$x^2+y^2$ et $\sin^2(x) + (x^x)^2$. *ont la même représentation interne.*

*Dans le second cas la variable* x *n'apparaitra pas dans la liste des variables : celle-ci contiendra par contre des éléments correspondants à* $\sin(x)$ et $x^x$.
*Cette propriété sera utilisée plus loin.*

### 1.2.2. Entrée-sortie

Pour l'utilisateur, la notation récursive n'est évidemment pas très lisible. Aussi a-t-on prévu la transformation automatique de la notion usuelle dans la représentation récursive et vice-versa.

Tout d'abord la chaîne d'entrée qui représente le polynôme sous sa forme habituelle est transformée en notation préfixée, sous forme de liste :

a x + b → (+(* A X)B)

Une fonction, appelée REP, prend une telle liste comme argument, et donne comme résultat, la forme récursive. Comme REP est définie à l'aide des fonctions de bases (POLPLUS, PLUSF,....,POLGCD), la forme récursive obtenue est une forme dans laquelle toutes les simplifications rationnelles ont été effectuées. La fonction DISREP fait le travail inverse, c'est à dire produit la forme préfixée. Il est ensuite aisé de reconstituer la chaîne de caractères qui donnera un polynôme sous forme lisible. Il est clair que ni REP, ni DISREP ne peuvent fonctionner si on ne leur donne pas l'ordre d'importance des variables :

- cet ordre peut être imposé à priori : la liste des variables au sens large, est affectée à une variable globale appelée VARLIST.

- On peut également confier à une fonction (REPVAR), le soin de trouver les éléments en lesquels la formule est rationnelle.

On verra plus loin comment on peut combiner ces deux actions, pour que l'appel final à DISREP donne un résultat ordonné d'une façon pré-établie.

Ce qui suit est un exemple d'utilisation de ces transformations.

### 1.2.3. Méthode de Waring pour calculer une fonction symétrique rationnelle et entière des racines d'une équation.

#### 1.2.3.1. Rappel

Nous reprenons là, les termes mêmes de SERRET (Algèbre supérieure T1,p 385)

"Soit l'équation :

$$x^m + p_1 x^{m-1} + p_2 x^{m-2} + \ldots\ldots + p_{m-1} x + p_m = 0$$

dont les m racines sont

$$x_1, x_2, \ldots, x_m$$

et supposons qu'il s'agisse de trouver la valeur d'une fonction symétrique et entière V de ces racines.

Pour plus de clarté, il convient d'imaginer que l'on a ordonné la fonction V de la manière que nous allons indiquer. Soit $\alpha$, l'exposant de la plus haute puissance à laquelle se trouve élevée chaque racine, et en particulier la racine $x_1$ ; par $\beta$ l'exposant de la plus haute puissance à laquelle se trouve élevée la racine $x_2$ dans la partie de V qui contient $x_1^{\alpha}$ . En continuant ainsi, on sélectionnera un terme de la forme :

$$A\, x_1^{\alpha}\, x_2^{\beta}\, x_3^{\gamma} \ldots\ldots$$

avec $\alpha \geq \beta \geq \gamma \geq \ldots$

Plaçons ce terme en tête et appliquons la même règle à la détermination du rang des autres termes. Nous pourrons écrire alors :

$$V = A x_1^{\alpha} x_2^{\beta} x_3^{\gamma} \ldots x_m^{\mu} + \ldots.$$

Cela posé, considérons les fonctions symétriques fondamentales :

$$p_1 = \Sigma x_i$$

$$p_2 = \Sigma x_i x_j$$

$$\ldots\ldots\ldots\ldots$$

$$p_m = x_1 x_2 x_3 \ldots . x_m$$

et formons le produit $A p_1^{\alpha-\beta} p_2^{\beta-\gamma} \ldots p_{m-1}^{\lambda-\mu} p_m = P$

P est une fonction symétrique des racines, que nous pouvons ordonner de la même façon que V. Son premier terme sera évidemment

$$A x_1^{\alpha} x_2^{\beta} \ldots . x_m^{\mu}$$

En retranchant P à V, on obtient une nouvelle fonction symétrique $V_1$.

$$V_1 = V - P$$

Si l'on opère sur $V_1$ comme on a opéré sur V, on obtiendra une nouvelle fonction symétrique $V_2$.

$$V_2 = V_1 - P_1$$

$P_1$ étant un polynôme formé de la même façon que P. En poursuivant les opérations, on obtiendra une suite de fonctions symétriques

$$V_1, V_2, \ldots . V_{\mu}$$

telles que

$$V_1 = V-P$$
$$V_2 = V_1-P_1$$
.....
$$V_{\mu-1} = V_{\mu-2} - P_{\mu-2}$$
$$V_\mu = V_{\mu-1}-P_{\mu-1}$$

Le procédé est nécessairement limité, car si l'on imagine une fonction U, formée des premiers termes des fonctions $V_i$ et ordonnée de la même façon que ces fonctions, alors le premier terme de l'une quelconque des fonctions $V_i$, occupera dans U, un rang supérieur, au rang du premier terme de la fonction $V_{i-1}$. Or le nombre de termes susceptibles d'occuper dans U, un rang supérieur à celui d'un terme donné, est nécessairement limité. On finira donc par arriver à une constante (éventuellement nulle) et l'opération sera terminée. Supposons que ce soit $V_\mu$, ajoutons membre à membre les égalités ci-dessus, il vient :

$$V = P + P_1 + P_2 + ..... + P_{\mu-1} + V_\mu$$

formule qui exprime V, en fonction des $P_i$".

Ce processus est un processus de <u>démonstration</u> et de <u>construction effective</u>. Dans la pratique V n'est souvent composée que d'un polynôme homogène; ce que nous supposerons dans ce qui suit. Le degré de V détermine le degré de l'équation au delà duquel V, garde la même forme, par exemple le calcul de la somme des carrés des racines ne fera intervenir que $p_1$ et $p_2$ aussi suffit-il de se limiter au $2^{ème}$ degré. D'un autre côté, si le degré de l'équation est imposé, et que le degré de V lui est supérieur on se limitera aux $p_i$, tels que $i \leq n$.

### 1.2.3.2. Programmation de l'algorithme

Nous nous limitons au cas d'un polynôme symétrique homogène, à coefficient 1, et nous allons supposer dans un premier temps que les arguments que nous donnons à la fonction sont les suivants :

1) Le degré de l'équation
2) La fonction symétrique écrite en toute lettre.

La fonction LISP est appelée WARING, et un exemple d'appel pourra être :

```
WARING(3 (x1**3 + x2**3 + x3**3))
```

Nous verrons plus loin comment se libérer de l'écriture du second argument. La programmation suit exactement l'algorithme théorique. Tout d'abord on affecte à priori à la variable VARLIST, la liste (x1 x2...) les $x_i$ pouvant être écrits dans n'importe quel ordre, puisque la fonction est symétrique. Le second argument est alors mis sous forme récursive. La méthode de Waring nécessite, la liste des polynômes fondamentaux, jusqu'à l'ordre n. Cette génération ne se fait pas en formant les polynômes en $x_i$ et en les confiant à REP, mais directement sous forme récursive, en se fondant sur la remarque suivante :

Pour un ordre donné, on peut former par récurrence, les fonctions symétriques fondamentales, à partir des fonctions symétriques de l'ordre n-1. Il suffit en effet de remarquer, que par rapport à une racine quelconque, ces polynômes sont linéaires, et que l'on peut écrire :

$$p_1^{(n)} = x_n \times 1 + p_1^{(n-1)}$$

$$\vdots$$

$$p_i^{(n)} = x_n\, p_{i-1}^{(n-1)} + p_i^{(n-1)}$$

$$\vdots$$

$$p_n^{(n)} = x_n p_{n-1}^{(n-1)}$$

Comme tout est symétrique on peut permuter $x_1$ et $x_n$

NOTE : *Cette façon d'écrire n'est pas sans rapport avec la relation bien connue des combinaisons* $C_n^p = C_{n-1}^{p-1} + C_n^{p-1}$

EXEMPLE :

$p_1^{(1)} = x1$

---

$p_2^{(2)} = x1 + x2$

$p_2^{(2)} = x1\ x2$

---

$p_1^{(3)} = x1 + (x2 + x3)$

$p_2^{(3)} = x1\ (x2 + x3) + x2\ x3$

$p_3^{(3)} = x1\ (x2\ x3)$

---

Sur la forme récursive, cela se traduit ainsi :

- pour une variable : (x1)

$p_1^1 = (1\ 0)$

- pour 2 variables : (x2 x1)

$p_1^{(2)}$ a 2 termes : - La constante en x1 c'est à dire $p_1^{(1)}$

- Le coefficient de x1, qui doit s'écrire comme la constante 1 en une variable

on obtient donc ( (1) (1 0))

$p_2^{(2)}$ n'a pas de constante, son dernier terme est donc NIL et le coefficient de x1 est $p_1^{(1)}$.

Il s'écrit donc : ((1 0) NIL)

- pour 3 variables on aura

$p_1^{(3)}$ = ((1)) ((1) (1 0)))

$p_2^{(3)}$ = (((1) (1 0)) ((1 0) NIL))

$p_3^{(3)}$ = (((1 0) NIL) NIL)

A titre d'exemple, voici la forme récursive des 5 polynômes symétriques fondamentaux de l'équation du 5ème degré obtenus à partir de

$p_1^{(1)}$ = (1 0)

$p_1^{(5)}$ = ((((((1)))(((((1)))(((1))((1)(1 0)))))

$p_2^{(5)}$ = (((((1)))(((1))((1)(1 0))))(((((1)))((1)(1 0)))(((1)(1 0))((1 0)NIL))))

$p_3^{(5)}$ = ((((((1))((1)(1 0)))(((1)(1 0))((1 0)NIL)))((((1)(1 0))((1 0)NIL))
(((1 0)NIL)NIL)))

$p_4^{(5)}$ = (((((1)(1 0))((1 0)NIL))(((1 0)NIL)NIL))((((1 0)NIL)NIL)NIL))

$p_5^{(5)}$ = (((((1 0)NIL)NIL)NIL)NIL)

Pour un ordre donné, les formes récursives obtenues sont attachées respectivement aux atomes $p_1, p_2, p_3, \ldots$

Une fois ceci fait, il faut savoir déterminer sur les formes récursives des polynômes $V_i$, les puissances $\alpha$, $\beta$, $\gamma$...

En vertu de la méthode de représentation :

$\alpha$ = (nombre d'éléments de $V_i$)-1

$\beta$ = (nombre d'éléments de la première sous liste de $V_i$)-1 etc...

On s'arrête dès qu'une sous liste a pour longueur 1.

En effet, cela signifie que l'exposant correspondant et les suivants sont nuls. Ayant déterminé le n-uple $(\alpha,\beta,\gamma,\ldots,\lambda,\mu)$, on forme le n-uplé $(\alpha-\beta, \beta-\gamma,\ldots,\lambda-\mu,\mu)$. Cette liste sert à obtenir d'une part la chaîne

$$A\ p_1^{\alpha-\beta}\ p_2^{\beta-\gamma}\ldots \qquad (I)$$

dans laquelle la constante A est l'atome le plus à gauche de $V_i$, d'autre part le polynôme $P(x_1,x_2,\ldots,x_m)$ correspondant, sous forme récursive à l'aide des polynômes attachés aux symboles $p_i$. L'impression immédiate de la chaîne (I) donne une partie du résultat.

Quant au polynôme obtenu en machine, il est soustrait à $V_i$ pour donner le polynôme $V_{i-1}$. Le processus est répété tant que la différence n'est pas NIL. (polynôme nul en n variables).

### 1.2.3.3. Variante dans la présentation des arguments

Le fait d'écrire, textuellement la fonction symétrique comme $2^{ème}$ argument de Waring peut être acceptable pour la somme des puissances semblables, mais ne l'est plus pour les fonctions plus générales : ainsi l'exemple de SERRET $\Sigma\ x_i^3\ x_j^2\ x_k$, pour l'ordre 6 n'a pas moins de 120 termes. On pourrait songer à générer la formule, mais il parait plus simple de procéder comme suit :

Les arguments de WARING seront

1) Le degré de l'équation n

2) la liste ($\alpha$ $\beta$ $\gamma$....)

Si $\alpha+\beta+\gamma+\ldots \leq n$, alors on posera $n = \alpha+\beta+\gamma+\ldots$, en vertu d'une remarque faite plus haut. On formera ensuite le produit

$$P = p_1^{\alpha-\beta}\, p_2^{\beta-\gamma} \ldots$$

Ce polynôme est de la forme $V+V_1$ et V est nécessairement le polynome formé avec les monômes de plus haut degré de P. Il 'suffit' donc de supprimer, sur la forme récursive de P, les sous-listes qui correspondent à ces monômes de plus haut degré, c'est à dire en quelque sorte de soustraire V à P sans que V n'apparaisse effectivement quelque part. Cela fait, la suite de l'algorithme est semblable à ce que nous avons dit précédemment.

### 1.2.3.4. Résultats

L'algorithme a été essayé sur des exemples connus, que l'on peut d'ailleurs trouver dans SALMON [54]. On pourrait vérifier (ce qui paraît bien inutile!) les tables de CAYLEY, que l'on trouve dans cet ouvrage, mais on est vite arrêté par des considérations de temps.

Sommes de puissances semblables

```
Waring(2(2))P1** - 2*P2
Waring(3(3))P1**3 - 3*P1*P2 + 3*P3
Waring(5(5))P1**5 - 5*P1**3*P2 + 5*P1*P2**2 + 5*P1**2*P3 - 5*P2*P3 - 5*P1*P4 + 5*
Waring(2(5))P1**6 - 6*P1**4*P2 + 9*P1**2*P2**2 - 2*P2**3
Waring(5(6))P1**6 - 6*P1**4*P2 + 9*P1**2*P2**2 + 6*P1**3*P3 - 2*P2**3
                  - 12*P1*P2*P3 - 6*P1**2*P4 + 3*P3**2 + 6*P2*P4 + 6*P1*P5
```

```
Waring(3(7))P1**7 - 7*P1**5*P2 + 7*P1**4*P3 - 7*P1*P2**3 - 21*P1**2*P2*P3
                  + 7*P2**2*P3 + 7*P1*P3**2
```

Exemples divers (le dernier est l'exemple type de Serret).

```
Waring(5(3 2))P1*P2**2 - 2*P1**2*P3 - P2*P3 + 5*P1*P4- 5*P5
Waring(6(2 2 1 1))P2*P4 - 4*P1*P5 + 9*P6
Waring(6(3 2 1))P1*P2*P3 - 3*P1**2*P4 - 3*P3**2 + 4*P2*P4 - 7*P1*P5 - 12*P6
```

On pourra trouver dans la référence [24], un autre exemple d'application de cette représentation. Le travail décrit dans le chapitre suivant, en fait également usage.

*"Je suis ton esclave ! Parle, que veux-tu ?"*

*Histoire d'Aladin et de la lampe merveilleuse*

*"Livre des Mille et une nuits"*

# CHAPITRE III

## ALADIN : Un système conversationnel utilisant LISP comme machine primaire.

### 1.1. INTRODUCTION

La structure des données de LISP et la facilité de leur traitement ont incité de nombreux programmeurs à utiliser le système LISP, comme superviseur de systèmes particuliers, la plupart du temps, orientés vers la manipulation algébrique.

Dans ce cadre, il faut citer un certain nombre de travaux qui ont leur importance aux U.S.A. :

- Le système MATHLAB cité plus haut
- Le système REDUCE de A. Hearn [28] développé à Stanford pour répondre à divers besoin de physiciens.
- Le système de W.A.MARTIN developpé au M.I.T. [25]
- Le système de GRIESMER (IBM Yorktown Heights), très complet, [30]

Ces systèmes sont tous écrits en LISP, mais s'utilisent comme une machine de bureau évoluée : ceux à qui ils sont destinés (ingénieurs, physiciens, mathématiciens) n'ont pas à connaître le langage hôte, non plus qu'à écrire un programme au sens habituel du mot.

L'utilisateur dispose en fait d'un ensemble de commandes permettant de confier des données symboliques au calculateur afin de les transformer par des opérations, qui par exemple, pour les formules usuelles, sont la substitution, la dérivation, l'évaluation numérique etc... Le contrôle est redonné à l'utilisateur chaque fois que la commande a été exécutée. Syntaxiquement, ces commandes s'écrivent comme des fonctions ayant des expressions comme arguments; ce qui donne aux entrées un aspect très analogue à des instructions de type algol.

Le système ALADIN que nous allons décrire dans ce chapitre demeure dans ce cadre. Il s'agit d'un système expérimental écrit pour l'étude des commandes les plus importantes applicables aux problèmes de manipulations algébriques usuelles.

Les résultats partiels ou définitifs peuvent être, à la demande imprimés sous forme linéaire sur le terminal utilisé pour l'entrée, ou disposés sur un écran cathodique sous leur forme bidimentionnelle.

L'usage d'une console graphique et des dispositifs qui s'y rattachent, permet de donner comme argument à une commande tout ou partie d'une formule apparaissant sur l'écran. Cela libère, l'utilisateur de l'obligation de linéariser ce qu'il voit sur l'écran, dans le cas où il voudrait appliquer une commande sur un résultat intermédiaire qui a été visualisé.

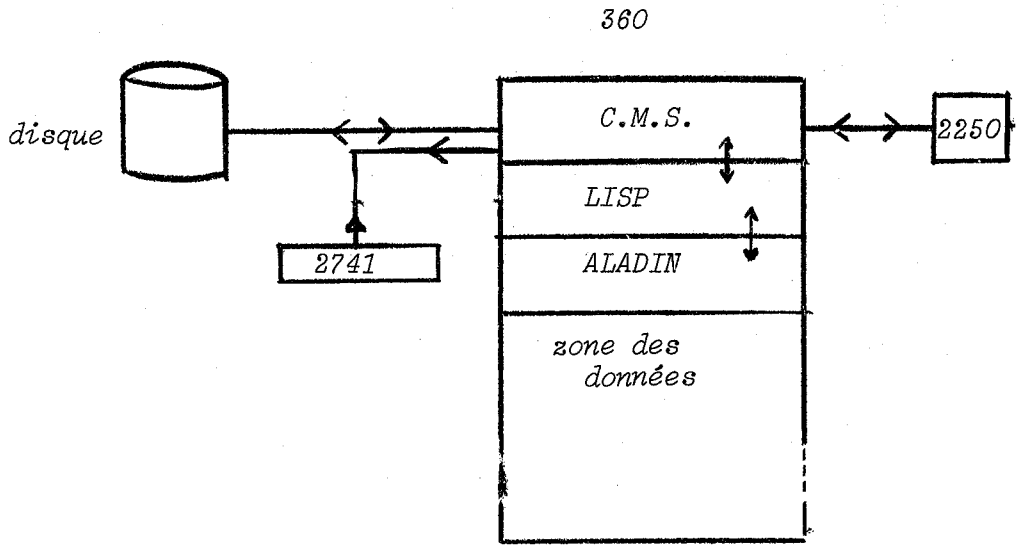
Quant au nom "Aladin", disons qu'il veut rappeler, sans aller trop loin dans l'analogie, que pour une certaine classe de travaux, il peut être judicieux d'utiliser le calculateur comme un "bon génie obéissant"...

## 1.2. MATERIEL UTILISE ET GENERALITES SUR LA PROGRAMMATION DU SYSTEME

Le matériel dont nous avons disposé se composait :

- d'un ordinateur IBM 360 modèle 67.
- d'une console de visualisation IBM 2250 modèle 1
- d'un terminal, type machine à écrire (IBM 2741 ou télétype).

Au point de vue de la programmation, Aladin est écrit en LISP 1.5. et fonctionne sous le système CP 67/CMS. Plus précisément, nous pouvons dire qu'il fonctionne sur un ordinateur IBM 360, ayant la configuration suivante.




En fait, l'élément le plus important est le système LISP qui joue le rôle d'une machine primaire :

- Les données pour Aladin (formules), sont transformées en données pour LISP (listes et atomes)
- Les fonctions de transformation sont écrites en LISP (compilées ou interpretées selon leur importance).

Comme dans les travaux cités plus haut, trois niveaux de compréhension doivent être distingués.

a) L'utilisateur n'a besoin de connaitre que la façon d'appliquer les commandes.

b) Pour ajouter de nouvelles commandes, il faut savoir écrire une fonction en LISP et se conformer aux règles simples de définition des commandes.

c) Pour modifier, améliorer, (voire corriger !) le système, le lecteur interessé doit étudier l'ensemble des fonctions qui le composent.

Le chapitre suivant fournit des renseignements relatifs au point a) et constitue en quelque sorte "un manuel d'utilisateur", sans que ce terme soit pris à la lettre.

Les autres chapitres donnent quelques explications sur la méthode que nous avons suivie, c'est à dire apportent une réponse partielle à b) et c), la réponse aux "comment" sinon aux "pourquoi" se trouvant dans la liste complète du système !

## 2 . LE MANUEL DE L'UTILISATEUR

### 2.1. LES DONNEES

Les données, c'est à dire les objets sur lesquels on travaille, sont les expressions arithmétiques usuelles générées par une grammaire de type Algol. Ce type de grammaire est décrit dans de nombreux ouvrages et nous le supposons familier au lecteur.

Ces expressions contiennent des nombres, des variables, des opérateurs et certains symboles tels que i ($i^2=-1$)

#### 2.1.1. Les nombres

Ils sont entiers ou réels. La valeur absolue d'un nombre entier doit être inférieure à $2^{31}$, celle d'un nombre réel non nul doit être comprise entre $10^{75}$ et $10^{-75}$. Le + unitaire n'est pas autorisé.

Il n'y a qu'un seul zéro : (noté 0)

0 = -0 = $\pm$ 0.0 = $\pm$ 0.0E $\pm$ nn.

EXEMPLES

entiers : 34, -288

réels : 3.42, -0.1257E-2,

<F0> $A+B-\frac{C}{D}+E^{F}$

<F1> $A^{2}+B^{C^{N}}+X^{X}$

<F2> $\cfrac{1}{1+\cfrac{X}{2+\cfrac{X}{3+\cfrac{X}{4+X}}}}$

<F3> $\sum_{K=1}^{N}\left[A_{I,K}*B_{K,J}\right]$

<F4> $\int_{-1}^{1}FCT(X,A)dX$

<F5> $\frac{d^{2}}{dX^{2}}U(X,Y)+\frac{d^{2}}{dY^{2}}U(X,Y)$

<F6> $\frac{-B+\sqrt{B^{2}-4*A*C}}{2*A}$

*Disposition des formules sur l'écran*

<FORMULE> $A+B^{2}-\cfrac{C}{1-\frac{D}{X}}+\cfrac{\frac{1}{1}}{\cfrac{1}{\cfrac{1}{X}}}*FCT\left[X,\frac{Y}{2},K^{2}+1\right]+(X+1)^{\frac{Z}{2}}$

$FCT\left[X,\frac{Y}{2},K^{2}+1\right]$

*Sous-expression sélectionnée par le crayon optique.*

Cas des quotients de nombres entiers : Bien que les nombres rationnels ne soient pas définis explicitement dans Aladin, des expressions comme 3/4, 12/20, -8/9, 10/(-11) seront considérées par certaines commandes, comme des nombres rationnels, et traitées comme tels. Il est donc possible de conduire certains calculs, entièrement dans le mode rationnel.

En fait, dans cette version du système, la limitation des rationnels est celle des entiers ; ni le numérateur, ni le dénominateur ne peuvent dépasser $2^{31}$ en valeur absolue.

Dans une version améliorée, il peut être prévu d'inclure le traitement des nombres entiers et rationnels, sans limitation de précision (au moins d'un point de vue pratique). Cette extension existe dans les systèmes de Hearn et Griesmer. On pourra consulter également les références [41], [46].

### 2.1.2. Les variables

Les variables n'ont pas de type. Le nom qu'on leur donne suit la règle de formation des identificateurs :

EXEMPLE : A12 , VOLT

La longueur des identificateurs est limitée à 80 caractères.

Dans un but de généralité, nous pouvons introduire des variables indicées comme $a_i$, $t_{i,j+1}$, notées : A.(I)., T.(I,J+1)., mais ce type de variable ne peut pas recevoir de valeur par une instruction d'affectation.

Variables réservées : Les noms des commandes et ceux des fonctions standards ne peuvent être utilisés comme noms de variables simples. Le nombre complexe $i(i^2=-1)$ est noté 'I'.

La variable spéciale ? : Si le symbole ? apparait comme opérande dans une expression, sa valeur est, par convention, une expression affichée

sur l'écran cathodique. Plus précisément, lors de la décodification de l'expression qui contient ce symbole, le système attendra une détection au crayon optique. Une fois que l'utilisateur aura désigné une expression entière ou une sous-expression, ce qui a été "vu" par le crayon sera substitué logiquement au symbole ? en cours de traitement. Naturellement, le système se mettra autant de fois en attente qu'il y a de symboles ? dans l'expression d'entrée.

Les règles d'affectation de valeur à ? sont les suivantes :

1°) Pour sélectionner une sous-expression d'une formule 'pointer' son opérateur principal

2°) On ne peut pas sélectionner une variable simple : le fait de la pointer au crayon, sélectionne la plus petite sous-expression significative qui la contient.

EXEMPLE

Supposons que la formule suivante soit affichée sur l'écran :

$$\frac{A + B^2 + C + D}{1 - \sqrt{1 + \frac{X}{2}}}$$

Une détection au crayon optique sur :

B donne $B^2$

Un signe + du numérateur donne le numérateur

Le signe - donne le dénominateur

Le symbole $\sqrt{\ }$ donne $\sqrt{1 + \frac{X}{2}}$

etc....

Pour aider l'utilisateur, une fois la détection faite, seule reste sur l'écran la sous-expression logiquement sélectionnée ; tout le reste est momentanément effacé. Si l'utilisateur s'est trompé, ou a mal placé son crayon, il peut annuler l'affectation

fictive du symbole ?, en appuyant sur une touche prévue à cet effet, et essayer une nouvelle détection. Il peut répéter cela jusqu'à ce qu'il obtienne ce qu'il désirait. Son accord définitif sera donné en appuyant sur une autre touche. Ces deux touches, sont deux clés prises dans l'ensemble des clés de fonctions de la console 2250 [51]

### 2.1.3. Les opérateurs

Il s'agit ici des opérateurs de type arithmétique. Lorsqu'ils apparaissent dans une formule, ils ne sont pas appliqués à leurs arguments : '3+7' donné en entrée, n'est pas automatiquement transformé en '10'.

Nous pouvons les classer ainsi :

a) Opérateurs arithmétiques usuels : +,-,*,/,** , =

b) fonctions standards : SIN, COS, ARCTG, LN (logarithme népérien)
EXP, SQRT, ABS

c) Opérateurs spéciaux : ITG, SUM, DRVD, DRVP

utilisés par exemple dans les expressions suivantes :

| | | |
|---|---|---|
| ITG (X,A,B,F(X)) | signifiant | $\int_a^b f(x)\,dx$ |
| SUM (I,N1,N2,F(I)) | " | $\sum_{i=n1}^{n2} f(i)$ |
| DRVD (F(X),X,N) | " | $\frac{d^n}{dx^n} f(x)$ |
| DRVP(F(X,Y,Z),X,N1,Y,N2,Z,N3) | " | $\frac{\partial^{n1+n2+n3}}{\partial x^{n1}\partial y^{n2}\partial z^{n3}} f(x,y,z)$ |

Nous insistons encore sur le fait, que l'écriture de telles expressions n'implique aucunement une transformation automatique.

d) opérateurs préfixés non spécifiés.
   Par exemple FCT,G,H dans FCT(X), G(U,H(V)).

## 2.2. LES COMMANDES

Ce sont les véritables opérateurs d'Aladin. Elles sont divisées en 3 groupes.

a) la commande d'affectation
b) les commandes de transformations, qui sont des fonctions ayant des expressions comme arguments.
c) les commandes de contrôle, pour l'impression des résultats, l'affichage sur l'écran, l'effacement etc...

### 2.2.1. La commande d'affectation

Sa forme syntaxique est la suivante

<identificateur> <= = <expression> ;

L'expression en partie droite du signe <= est donnée comme valeur à la variable dont le nom est <identificateur> . Le cas échéant, les les commandes de transformation apparaissant dans l'expression, sont appliquées avant l'affectation.

Dans toute expression en partie droite, une variable non affectée se représente elle-même, alors qu'une variable affectée précédemment est remplacée par sa valeur. Cependant, les affectations successives n'ont pas d'effet rétroactif (En anglais : unravelling). Ainsi, après la série d'affectations,

```
FO <== A+B/C ;
A  <== 1 ;
B  <== K ;
C  <== 2 ;
F1 <== FO+1 ;
F2 <== A+B/C ;

l'on a    F1 = A+B/C+1
   et     F2 = 1+K/2.
```

Si une formule est affichée sur l'écran, <identificateur> apparait comme nom de la formule. Une détection au crayon optique sur ce nom affectera comme valeur au ? l'expression qui lui est associée.

### 2.2.2. Les commandes de transformation

La forme syntaxique est du type fonction.

CT (<expression>,...,<expression>)

L'opérateur effectif CT est appliqué à sa liste d'arguments, pour donner une autre expression

EXEMPLE :

DERIVER (SIN(X),X,1) donne COS(X)

L'imbrication des commandes est autorisée. L'évaluation se fait de gauche à droite et les commandes les plus internes sont appliquées les premières. Cette règle est d'ailleurs la règle standard d'évaluation des fonctions LISP.

EXEMPLE :

Supposons que CT1,CT2,CT3 soient des commandes (éventuellement les mêmes), nous pouvons écrire :

```
V1 <== CT1(CT2(VO),CT3(V2))+K ;
```

Les commandes les plus souvent appelées sont celles qui sont relatives aux opérations de substitution, sélection, séparation, simplification, dérivation et évaluation numérique. Naturellement cette liste n'est pas limitative et on peut toujours rajouter suivant les besoins de l'utilisateur des nouvelles commandes fabriquées ou non à partir des commandes existantes.

Dans les exemples qui suivent, nous utiliserons la commande de contrôle IMPRIMER, qui permet d'imprimer une expression sur le terminal. C'est une fonction identité, c'est à dire que sa valeur est identique à son argument.

Le système Aladin donne le contrôle à l'utilisateur après avoir imprimé le symbole.

-->

et l'impression se fait en lettres majuscules.

2.2.2.1. Substitution

Deux commandes sont fournies :

REMPLACER et RPLOC

REMPLACER $(e_1, e_2, e_3)$.

Toutes les occurences de l'expression $e_1$ dans l'expression $e_3$ sont remplacées par l'expression $e_2$. En fait $e_3$ n'est pas détruite, mais on obtient une nouvelle expression dans laquelle la substitution a été effectuée.

RPLOC$(e_2)$ (remplacement local)

Cette commande permet de remplacer une sous-expression non atomique dans une formule affichée. La sous-expression à remplacer est désignée par le crayon optique et seule l'occurence désignée sera remplacée par $e_2$.

Le résultat est une formule qui ne diffère de la formule sur laquelle a été faite la détection optique, que par la sous-expression $e_2$. Dans le cas où $e_2$ est un ?, on se souviendra que la première détection servira à affecter une valeur au symbole ?, la seconde seulement servira à désigner la sous-expression à remplacer.

NOTE : RPLOC *est identique à* REMPLACER *si la sous expression qui doit être modifiée n'apparait qu'une seule fois dans la formule affichée.*

### 2.2.2.2. Sélection de sous-expressions

Ces commandes sont utiles si l'on ne dispose pas de la console de visualisation. Dans le cas contraire, l'usage du ? permet de s'en dispenser chaque fois que cela est possible.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| NUM | (e) | si e est de la forme | $e_1/e_2$, | donne | $e_1$ | sinon | e |
| DENOM | (e) | " " | " | " | " | $e_2$ | " | 1 |
| DROITE | (e) | " " | " | $e_1=e_2$ | " | $e_2$ | " | e |
| GAUCHE | (e) | " " | " | " | " | $e_1$ | " | e |

NOTE : *Le signe = n'est pas le signe d'affectation, mais il a son sens algébrique conventionnel.*

### 2.2.2.3. Distribution du dénominateur

Si une expression e est de la forme (a+b+c+...+k)/d la fonction SEPTERMS(e) donne comme résultat :

$$a/d + b/d + c/d + \ldots + k/d$$

et e autrement.

EXEMPLES

```
  -->
e0<=1/(1+x/(1+1/x));
  -->
imprimer(e0);

  1/(1 + X/(1 + 1/X))
  -->
e1<=remplacer(1,z**2,e0);
  -->
imprimer(e1);

  Z**2/(Z**2 + X/(Z**2 + Z**2/X))
  -->
imprimer(remplacer(z**2,sin(x),e1));

  SIN(X)/(SIN(X) + X/(SIN(X) + SIN(X)/X))
  -->

e2<=imprimer(remplacer(x,y,remplacer(sin(x),z,remplacer(z**2,sin(x),e1))));

  Z/(Z + Y/(Z + Z/Y))
  -->



imprimer(num(e2));

  Z
  -->
e2<=1+denom(e1)-num(denom(1/1/x));
  -->
imprimer(e2);

  1 + Z**2 + X/(Z**2 + Z**2/X) - X
  -->

e2<=imprimer(1+denom(e1)-num(denom(1/(1/x))));

  1 + Z**2 + X/(Z**2 + Z**2/X) - 1
  -->


f0<=(a+b*x-s+b**2)/(1-x);
  -->
imprimer(f0);

  (A + B*X - S + B**2)/(1 - X)
  -->

imprimer(septerms(f0));

  A/(1 - X) + (B*X)/(1 - X) + (-S)/(1 - X) + B**2/(1 - X)
  -->

f1<=a*x=b;
  --\
```

```
imprimer(gauche(f1)-droite(f1)=0);

  A*X - B = 0
  -->

f0<=f(x,y,z);
  -->
f1<=remplacer(x,f0,f0);
  -->
imprimer(f1);

  F (F (X,Y,Z),Y,Z)
  -->
imprimer(remplacer(z,f1,f1));

  F (F (X,Y,F (F (X,Y,Z),Y,Z)),Y,F (F (X,Y,Z),Y,Z))
  -->


x1<=-b+sqrt(b**2-4*a*c);
  -->
imprimer(x1);

  -B + SQRT(B**2 - 4*A*C)
  -->
x1<=imprimer(x1/2);

  (-B + SQRT(B**2 - 4*A*C))/2
  -->

x2<=(-b-sqrt(b**2-4*a*c))/2;
  -->
s<=imprimer(x1+x2);

  (-B + SQRT(B**2 - 4*A*C))/2 + (-B - SQRT(B**2 - 4*A*C))/2
  -->
```

2.2.2.4. La simplification.

Tout d'abord, les commandes de simplifications permettent de réaliser les transformations triviales suivantes :

A+0 = 0+A ⇒ A

A+0 = 0+A ⇒ 0

0/A ⇒ 0

A+1 = 1×A = A/1 ⇒ A

A/A ⇒ 1

Par ailleurs si m et n sont des entiers :

$A^m * A^n \Rightarrow A^{m+n}$

$(A+B)^m \Rightarrow A^m * B^n$

n+A+n ⇒ A + [m+n]

m*A + n*A ⇒ [m+n] * A (le crochet dénotant l'opération effectuée).

Mais le lecteur comprendra mieux l'esprit dans lequel est faite la simplification, s'il retient les principes suivants :

Toute expression peut être considérée comme rationnelle en un certain nombre d'éléments qui peuvent être :

- des variables simples ou indicées
- des expressions formelles telles que :

F(X,Y), SIN(X),
ITG(X,A,B,G(X)),
X * * N (N étant une variable non affectée)
etc...

Une fois cette reconnaissance faite, l'expression générale est ordonnée en ces éléments en fonction des règles de traitement des expressions polynomiales et rationnelles.

En particulier une fraction est toujours mise sous forme irréductible, comme on le verra sur les exemples. Une telle méthode ne permet pas les simplifications provenant de propriétés fonctionnelles.
Par exemple : $\sin^2(x) + \cos^2(x)$ n'est pas réduit à 1. Cependant les expressions complexes rationnelles en i ($i^2=-1$) peuvent être simplifiées (voir exemples).

Contrairement à certains systèmes de manipulation algébrique [46], [26], il n'y a pas dans Aladin de simplification automatique, lors de l'entrée d'une formule. C'est à l'utilisateur d'appliquer explicitement les commandes qui lui sont fournies. Ces commandes sont SMP, SMPO, SMPTERMS.

#### 2.2.2.4.1. Simplification standard : SMP($e_1$)

Le résultat de l'application de la commande SMP sur $e_1$ est une expression simplifiée au sens ci-dessus : c'est à dire qu'il se présente comme un polynôme ou une fraction rationnelle ordonnés en plusieurs variables à la façon de Mathlab. Pour un polynôme, la forme résultat est :

$$\sum_{i=o}^{n} f(v_2,v_3,\ldots,v_n)\ v_1^i$$

les coefficients des $v_1^i$ étant des polynômes en n-1 variables, de la même forme, $v_2$ étant la variable principale, etc...

Les polynômes en $v_n$ sont des polynômes sur les entiers.

L'ordre d'importance standard est le suivant :

1) La variable 'I'
2) Les identificateurs (atomes) ordonnés suivant un ordre propre à LISP. Pour les variables à une lettre on retrouve l'ordre lexicographique.

3) Les expressions formelles ordonnées par une fonction spéciale.

2.2.2.4.2. Simplification avec ordre imposé.

SMPO ($e_1$, LISTE ($v_1$,$v_2$,...,$v_n$))

La façon dont le résultat est ordonné dans le cas précédent, pouvant ne pas convenir à l'utilisateur, ce dernier peut imposer son ordre sur tout ou partie de l'ensemble des éléments en lesquels $e_1$ est rationnelle grâce à la commande SMPO. La liste d'arguments de la fonction LISTE donne l'ordre des "variables" dans le sens décroissant. Les "variables" qui n'apparaissent pas dans cette liste seront ordonnées entre elles par la méthode standard utilisée par SMP. D'un autre coté, toute "variable" $v_i$ argument de LISTE et n'appartenant pas à $e_1$, sera ignorée.

2.2.2.4.3. Simplification d'une somme terme à terme.

SMPTERMS($e_1$)

Parfois, une formule 'simplifiée' peut apparaître comme une formule compliquée !.Cela provient souvent du fait que les termes d'une somme sont tous réduits au même dénominateur. Pour éviter cet inconvénient, l'utilisateur peut simplifier une somme terme à terme par SMPTERMS.

Naturellement, il est facile d'écrire une commande SMPOTERMS analogue à SMPO.

2.2.2.4.4. Evaluation numérique à l'aide des commandes de simplification.

Les diverses commandes de simplification peuvent être utilisées pour obtenir une évaluation numérique des formules. Si l'on substitue aux variables des nombres entiers ou des quotients de nombres, c'est à dire de ce fait des nombres rationnels, la partie numérique du

APPLICATION DE SMP

```
   ef<=a*1+b/1+c/c-1+d*0+0+6+56+a*2-78;
     -->
   imprimer(smp(ef));

     3*A + B - 16
     -->

   imprimer(smp(3214/678-55/345));

     11906/2599
     -->
   ef<=a*1+b/3+1/2&@-5/6;
     -->
   ef<=imprimer(smp(ef));

     (3*A + B - 1)/3
     -->
   s1<=imprimer(smp((a+b)*(c-d)));

     (C - D)*A + (C - D)*B
     -->
   s2<=b*2*a+b**2+a*a;


   -->

 s3<=imprimer(smp(s1/s2));


   (C - D)/(A + B)
   -->

ee<=((x**x)**2-sin(x)**2)/(sin(x)+x**x);
   -->
ee<=imprimer(smp(ee));

   -SIN(X) + X**X
   -->
for<=a+x-g(x)+b*x*a+a**2*x*g(x);
   -->
imprimer(smp(for));

   G (X)*X*A**2 + (X*B + 1)*A + X - G (X)
   -->
for2<=imprimer(smp(for*ee));

   (-G (X)*SIN(X) + G (X)*X**X)*X*A**2 + ((-SIN(X) + X**X)*X*B - SIN(X) + X**X)*A +
    (-SIN(X) + X**X)*X + G (X)*SIN(X) - G (X)*X**X
   -->
imprimer(smp(for2/for));

   -SIN(X) + X**X
   -->
```

## CAS DES EXPRESSIONS COMPLEXES

```
e0<=(a+'i'*b-'i'*c)*(b*a*'i'-c);
  -->
imprimer(smp(e0));

  (-B**2 + C*B - C)*A + (B*A**2 - C*B + C**2)*'I'
  -->
i<='i';
  -->
e1<=1+i+i**2+i**3+i**4+i**5;
  -->
imprimer(smp(e1));

  1 + 'I'
  -->

imprimer(smp((121-a**2*'i')/(11+'i'*a)));

  (121 - A**2*'I')/(11 + A*'I')
  -->

imprimer(smp((121-a**2*'i'**2)/(11+'i'*a)));

  11 - A*'I'
  -->

e2<=imprimer(smp((a+'i'*b)**4));

  A**4 - 6*B**2*A**2 + B**4 + (4*B*A**3 - 4*B**3*A)*'I'
  -->
```

## APPLICATION DE SMPO

```
s2<=(a+b)**2;
  -->
s0<=s2*(x+a-b);
  -->
imprimer(smpo(s0,liste(x,a,b)));

  (A**2 + 2*B*A + B**2)*X + A**3 + B*A**2 - B**2*A - B**3
  -->
imprimer(smpo(s0,liste(b,x,a)));

  -B**3 + (X - A)*B**2 + (2*A*X + A**2)*B + A**2*X + A**3
  -->

e0<=a+b/e+b**2-d+c/a-f+x**n1-f0(x,y);
  -->


e1<=imprimer(smp(e0));

  (E*A**2 + (E*B**2 + B - E*D + (-F + X**N1 - F0 (X,Y))*E)*A + E*C)/(E*A)
  -->
```

```
imprimer(smpo(e0,liste(f,d,a,x**n1,f0(x,y))));


  (-E*A*F - E*A*D + E*A**2 + (E*X**N1 - E*F0 (X,Y) + E*B**2 + B)*A + E*C)/
  (E*A)
  -->
imprimer(smpo(e0,liste(f0(x,y),e,b)));

  (-A*E*F0 (X,Y) + (A*B**2 + A**2 + (-D - F + X**N1)*A + C)*E + A*B)/(A*E)

  -->
imprimer(smpo(e1,liste(xx,a,zz,f,qq,b1,d,b)));



  (E*A**2 + (-E*F - E*D + E*B**2 + B + (X**N1 - F0 (X,Y))*E)*A + E*C)/(E*A
  )
  -->

f0<=a/a*x+(x-y)**2(@/(x**2-y**2)+456/24*a;
  -->
imprimer(smp(f0));

  ((19*X + 19*Y)*A + X**2 + (Y + 1)*X - Y)/(X + Y)
  -->
imprimer(smpterms(f0));

  X + (X - Y)/(X + Y) + 19*A
  -->
```

EVALUATION AVEC LES COMMANDES DE SIMPLIFICATION

```
imprimer(smp(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1)))))));

  34/21
  -->
imprimer(smp(34.0+17*2-1));

  6.700000E+01
  -->

e0<=a+b/c-d+e;
  -->
e1<=smp(remplacer(a,12/3,remplacer(b,3,remplacer(c,3/2,
       remplacer(d,1,remplacer(e,89/3,e0))))));
  -->
imprimer(e1);

  104/3
  -->

e0<=(a+b*x)/(y*z+34);
  -->
e1<=imprimer(num(e0)/(smp(remplacer(z,3/5,remplacer(y,12/345,denom(e0))))));

  (A + B*X)/(19562/575)
  -->
imprimer(smp(e1));
  (575*A + 575*X*B)/19562
  -->
```

résultat sera un nombre rationnel (éventuellement entier). En substituant au moins un nombre réel, cette partie numérique sera réelle. En fait, en mode rationnel, on est vite arrêté par la précision de la machine. La plupart des systèmes du même type qu'Aladin, possèdent la propriété d'effectuer les calculs sur les entiers sans limitation de précision. Cette extension se révèle être indispensable pour conduire à bien des calculs non triviaux.

#### 2.2.2.5. Evaluation numérique

EVALUER($e_1$) ;

L'évaluation numérique de certaines formules est possible par la commande EVALUER. Cette possibilité est restreinte aux expressions arithmétiques ne contenant que +,-,*,/,* * et les fonctions standards. L'apparition de tout autre opérateur provoque l'impression d'un message d'erreur et donne 0 comme résultat. Autrement le résultat est toujours réel. Si au moment de l'évaluation de $e_1$, certaines variables ont reçu une affectation numérique, cette valeur sera prise en considération pour l'évaluation. Il s'agit là d'une exception à la règle de non-rétroactivité, valable uniquement pour la durée de l'évaluation. Pour les autres variables, la fonction demandera les valeurs numériques que l'utilisateur veut leur substituer. Cette correspondance faite entre une variable et une valeur numérique n'est pas non plus à confondre avec une affectation.

Naturellement, les performances d'un tel sous-programme, liées à une méthode interprétative, sont faibles. La performance n'est d'ailleurs pas le but recherché : il s'agit avant tout de disposer d'une commande qui permette à l'utilisation de se faire une idée sur le plan numérique.

EVALUATION NUMERIQUE

```
f1<=a+b/c-c**n+x+(d-a/2);
  -->
imprimer(evaluer(f1));
  D=?
12;
  X=?
4.0;
  N=?
2;
  C=?
45.3e-01;
  B=?
30;
  A=?
78.9834;

  4.159332E+01
  -->
a<=23;
  -->
c<=6.89;
  -->
d<=a*f(x);
  -->
b<=smp(a*d);
  -->
b<=smp(a*c);
  -->
imprimer(b);

  1.584700E+02
  -->
imprimer(evaluer(f1));
  D=?
67;
  X=?

4;
  N=?
3;

  -2.215827E+02
  -->
imprimer(f1);

  A + B/C - C**N + X + D - A/2
  -->
```

### 2.2.2.6. La différentiation

Les deux commandes utilisables sont :

DERIVER et APDERIV

#### 2.2.2.6.1. DERIVER

la forme DERIVER $(e_1,x_1,n_1,x_2,n_2,\ldots,x_p,n_p)$ permet d'obtenir l'expression

$$\frac{\partial^{n_1+n_2+\ldots+n_p}}{\partial x_1^{n_1}\partial x_2^{n_2}\ldots\partial x_p^{n_p}}\ e_1$$

Les $x_i$ doivent être des variables simples. La dérivation de $e_i$ par rapport à ces variables se fait à tous les niveaux. Si l'expression $e_1$ contient des sous-expressions telles que F(x,y,H(G(z)) ou encore si quelques $n_i$ ne sont pas numériques, l'application de la commande produit des expressions contenant les symboles DRVD (cas d'une seule variable) ou DRVP (cas de plusieurs variables). Si le dernier $n_i$ est 1, il peut être omis, c'est à dire que DERIVER(e,x,1) est équivallent à DERIVER(e,x).

Si le résultat est une somme, la simplification est laissée au soin de l'utilisateur, cependant les termes de la somme sont simplifiés.

#### 2.2.2.6.2. APDERIV

La commande APDERIV employée dans APDERIV($e_1$) permet de transformer une expression contenant les opérateurs formels DRVD ou DRVP, en une expression dans laquelle les dérivations impliquées par ces opérateurs sont effectuées, si c'est possible.

Si $e_1$ ne contient ni DRVD, ni DRVP ou encore si ces opérateurs ne peuvent pas s'appliquer, l'expression demeure inchangée.

Comme pour DERIVER, lorsque le résultat est une somme, seuls les termes sont simplifiés.

NOTE importante : *On remarquera que :*

DERIVER(DRVD(F(X),X,N)) donne DRVD(F(X),X,N+1)
APDERIV(DRVD(F(X),X,N)) " DRVD(F(X),X,N)

*Par ailleurs, si l'on a obtenu, après quelque transformation une expression de la forme :*
(1) DRVD(F($e_i$),$e_i$,1), *dans laquelle* $e_i$ *n'est pas une variable simple, et si l'on remplace* F($e_i$) *par une expression effectivement dérivable, on devra avant d'appliquer* APDERIV, *substituer une variable simple à* $e_i$ *dans* (1). *Ceci provient du fait que la dérivation ne peut se faire que par rapport à une variable simple.*

### 2.2.2.7. Division des polynômes ordonnés suivant les puissances croissantes d'une variable.

La commande est DEVELOP et s'emploie dans DEVELOP (e,var,n).

Le résultat est un développement partiel de la fraction rationnelle e, suivant les puissances croissantes de la variable var, jusqu'à l'ordre n. Le reste est ignoré.

Si e a un pôle en 0 d'ordre k, le résultat comportera $1/(var)^k$ en facteur et le développement,s'il existe,sera poursuivi jusqu'à l'ordre n+k.

L'intérêt d'une telle commande n'est pas à démontrer, vu l'importance des développements limités.

# EXEMPLES DE DERIVATION

```
e0<=a*sin(b*t+c);
  -->
e1<=imprimer(deriver(e0,t,2));

  -SIN(B*T + C)*B**2*A
  -->
e2<=imprimer(deriver(a+b*x+c*x**n+d*ln(x)+f*exp(arctg(x/y)),x));


  B + X**(N - 1)*N*C + D/X + (EXP(ARCTG(X/Y))*Y*F)/(X**2 + Y**2)
  -->
imprimer(deriver(x**x**x,x));

  ((X**X)**(X - 1)*X**(X - 1)*LN(X) + (X**X)**(X - 1)*X**(X - 1))*X**2 + (X**X)**(X
   - 1)*X**X*LN(X**X)
  -->

imprimer(deriver(x**(x**x),x));

  (X**(X**X - 1)*X**(X - 1)*LN(X)**2 + X**(X**X - 1)*X**(X - 1)*LN(X))*X**2 + X**(X**X
   - 1)*X**X
  -->

e3<=deriver(f(x),x,n);
  -->
imprimer(e3);

  DRVD(F (X),X,N)
  -->
imprimer(deriver(e3,x,2));

  DRVD(F (X),X,N + 2)
  -->

e4<=deriver(f(y(x)),x,1);
  -->
imprimer(e4);

  DRVP(F (Y (X)),Y (X),1)*DRVD(Y (X),X,1)
  -->
imprimer(deriver(e4,x,1));

  DRVP(F (Y (X)),Y (X),2)*DRVD(Y (X),X,1)**2 + DRVP(F (Y (X)),Y (X),1)*DRVD(Y (X),X,2)
  -->
e5<=itg(x,a(t),b(t),f(x,t));
  -->
e6<=imprimer(deriver(e5,t));


  ITG(X,A (T),B (T),DRVP(F (X,T),T,1)) + F (B (T),T)*DRVD(B (T),T,1) - F (A (T),T)*DRVD
   (T),T,1)
  -->
```

```
e0<=deriver(f(x),x,2);
  -->
imprimer(e0);

  DRVD(F (X),X,2)
  -->
e1<=apderiv(remplacer(f(x),sin(omega*x),e0));
  -->
imprimer(e1);

  -SIN(OMEGA*X)*OMEGA**2
  -->
e0<=imprimer(deriver(f(x,y(x)),x,n));

  DRVD(F (X,Y (X)),X,N)
  -->
imprimer(apderiv(e0));

  DRVD(F (X,Y (X)),X,N)
  -->
e1<=imprimer(apderiv(remplacer(v@n,1,e0)));

  DRVP(F (X,Y (X)),X,1) + DRVP(F (X,Y (X)),Y (X),1)*DRVD(Y (X),X,1)
  -->

e2<=itg(x,a,b(t),f(x,t));
  -->
e3<=imprimer(deriver(e2,t));

  ITG(X,A,B (T),DRVP(F (X,T),T,1)) + F (B (T),T)*DRVD(B (T),T,1)
  -->
e4<=imprimer(apderiv(remplacer(b(t),sin(1/t),e3)));

  ITG(X,A,SIN(1/T),DRVP(F (X,T),T,1)) + F (SIN(1/T),T)*(-COS(1/T)/T**2)
  -->

imprimer(smp(e4));

  (ITG(X,A,SIN(1/T),DRVP(F (X,T),T,1))*T**2 - F (SIN(1/T),T)*COS(1/T))/T**2
  -->

e2<=imprimer(remplacer(x,t,remplacer(f(x,t),f(t),e2)));

  ITG(T,A,B (T),F (T))
  -->
imprimer(deriver(e2,t));

  F (B (T))*DRVD(B (T),T,1)
  -->
imprimer(deriver(e2,a));

  -F (A)
  -->
```

DEVELOPPEMENTS LIMITES

```
e0<=(x-x**3/6+x**5/120)/(1-x**2/2+x**4/24);
  -->
e1<=imprimer(develop(e0,x,5));

 X + X**3/3 + (2*X**5)/15

  -->
e2<=imprimer(develop(imprimer(smp(e0)),x,5));

 (X**5 - 20*X**3 + 120*X)/(5*X**4 - 60*X**2 + 120)

 X + X**3/3 + (2*X**5)/15

  -->
f0<=imprimer(develop((a*x+b)/(c*x+d),xm ,4));

 B/D + ((D*A - C*B)*X)/D**2 + ((-D*C*A + C**2*B)*X**2)/D**3 + ((D*C**2*A -

  C**3*B)*X**3)/D**4 + ((-D*C**3*A + C**4*B)*X**4)/D**5

  -->
f1<=imprimer(smp(eps**3*(a-eps)**2));

 EPS**3*A**2 - 2*EPS**4*A + EPS**5

  -->
imprimer(develop(1/f1,eps,3));

 (1/EPS**3)*(1/A**2 + (2*EPS)/A**3 + (3*EPS**2)/A**4 + (4*EPS**3)/A**5 +

  (5*EPS**4)/A**6 + (6*EPS**5)/A**7 + (7*EPS**6)/A**8)

  -->
imprimer(develop((a0+a1*x+a2*x**2+a3*x**3)/(b3*x**3+b2*x**2+b1*x+b0),x,3));

  A0/B0 + ((B0*A1 - B1*A0)*X)/B0**2 + ((-B1*B0*A1 + B0**2*A2 + (-B2*B0 + B1**2)*

  A0)*X**2)/B0**3 + (((-B2*B0**2 + B1**2*B0)*A1 + B0**3*A3 - B1*B0**2*A2 -

  B0**2*A0*B3 + (2*B1*B2*B0 - B1**3)*A0)*X**3)/B0**4

  -->
```

### 2.2.3. Commandes de contrôle

Ce sont des commandes de sortie. Seule, IMPRIMER a une valeur. Les autres sont utilisées pour leur effet.

IMPRIMER (<expression>)

<expression> est imprimée sur la machine à écrire sous forme linéaire, comme nous l'avons déjà montré dans de nombreux exemples. La valeur de IMPRIMER (X) est X, de cette façon on peut placer cette commande n'importe où dans une expression pour contrôler des résultats intermédiaires.

```
EXEMPLE : imprimer(remplacer(imprimer(x),imprimer(z),1+1/(x+imprimer(sin(x)))));
          X
          Z
          SIN(X)
          1+1/(Z+SIN(Z))
```

Les autres commandes sont :

IMAGE, EFFACER, RAYER, DEP

A l'exception de DEP, ces commandes ont pour argument une variable qui est obligatoirement une variable affectée. Nous signalerons ceci en écrivant <nom de formule>. Ces commandes doivent être utilisées au niveau supérieur d'Aladin : c'est à dire que, n'ayant aucune valeur, elles ne peuvent apparaître dans une expression.

IMAGE (<nom de formule>) ;

L'expression correspondant à la valeur de la formule nommée est affichée en deux dimensions sur l'écran, son nom (la variable) apparaissant à gauche. Les appels successifs à IMAGE permettent d'afficher les formules les unes au dessus des autres. Théoriquement, il n'y a aucune formule trop longue pour le système, elle sera tronquée, l'écran

jouant le rôle d'une fenêtre se plaçant sur un plan infini. De même, quand l'écran est plein, un appel ultérieur à IMAGE semble ineffectif, mais en fait, l'expression est virtuellement affichée sur le plan. En cas de troncation dans une direction quelconque, une commande permet de déplacer la fenêtre dans le plan, de façon que l'utilisateur puisse voir ce qui était caché.

Si la valeur d'une variable qui a été affichée est changée par une affectation, l'image (visible, partiellement visible ou invisible) est automatiquement mise à jour : ceci est important pour les détections ultérieures au crayon optique, car tout ce qui est affiché doit refléter exactement l'état interne de l'ensemble des formules visualisées au sens de la commande IMAGE.

Une détection au crayon optique sur une partie de formule partiellement visible, parce que tronquée, à néanmoins le même effet que si cette partie était totalement visible : le moindre point lumineux apparaissant aux limites de l'écran représente une sous expression.

NOTE : *Si l'argument de* IMAGE *est incorrect, la commande est ignorée.*

Commandes d'effacement:

EFFACER (<nom d'image>) ;

L'image de la formule, au sens de la commande précédente, disparait de l'écran qui subit un réarragement. Mais la variable garde sa valeur. Si on affiche à nouveau la formule, l'image correspondante sera ajoutée aux autres comme nouvelle image.

RAYER (<nom de formule>) ;

Cette commande a le même effet visuel que EFFACER mais la variable est 'atomisée', c'est à dire qu'elle redevient une variable non affec-

tée. On peut l'utiliser pour libérer la place prise en mémoire par des résultats intermédiaires devenus inutiles.

Commande de déplacement :

DEP (tx,ty) ;

Cette commande permet un déplacement relatif de l'écran et des images. En fait, les deux arguments de la fonction doivent être des nombres entiers, et sont interprétés comme les composantes d'une translation de l'image par rapport à l'écran. Pour des raisons technologiques, l'unité de mesure est 1/4095 du côté de l'écran. Pour avoir un point de repère, on pourra se souvenir que pour se déplacer, suivant l'axe des x, d'un caractère, on peut écrire DEP(84,0) :

Après de nombreux appels à DEP, l'utilisateur peut désirer remettre les images dans leur position standard initiale.

Par convention, ceci est fait en substituant à tx et ty l'identificateur NIL.

DEP(NIL,NIL) ;

ainsi l'utilisateur est quitte de se souvenir de l'ensemble des translations qu'il a effectuées.

Si tx et ty sont autre chose que des nombres entiers ou NIl la commande est ignorée.

## 2.3. EXEMPLE SIMPLE DE SESSION

On désire faire le changement de variable

$$u = x+at$$
$$v = x-at$$

dans l'équation des cordes vibrantes :

$$\frac{\partial^2 F}{\partial x^2} - \frac{1}{a^2}\frac{\partial^2 F}{\partial t^2} = 0$$

en employant une méthode différentielle. [55].

Les différents pas du calcul, sont numérotés et nécessitent quelques explications.

1 . On écrit pour mémoire l'équation de départ.

2 . Dans notre cas, les différentielles premières des variables nouvelles (u,v) par rapport aux anciennes (x,t) sont simples. Elles sont directement affectées aux variables du et dv.

3 . On affecte à $e_1$ la différentielle seconde de f par rapport aux variables nouvelles. Dans ce cas simple, les différentielles $d^2u$ et $d^2v$ sont nulles.

4 . Dans $e_1$, du et dv ont été remplacés par leurs valeurs en dx et dt. On simplifie,dx et dt étant pris comme variables les plus importantes.
En vertu de la théorie des différentielles, les coefficients de $dx^2$ et $dt^2$ sont respectivement les dérivées secondes $\frac{\partial^2 F}{\partial x^2}$ et $\frac{\partial^2 F}{\partial t^2}$, mais exprimées en fonction des variables nouvelles.

5 . Le coefficient de $dx^2$ est sélectionné au crayon optique et affecté à $e_2$.

6.7.Comme la formule $e_1$ est trop longue et a été tronquée, on opère un déplacement, puis on sélectionne au crayon optique le coefficient de $dt^2$, qui est affecté à $e_3$.

8 . Ayant remis tout en place, on réécrit l'équation des cordes vibrantes, avec $e_2$ et $e_3$. La simplification donne le résultat :
$4 \frac{\partial^2}{\partial u \partial v} f$ dont l'égalité à 0 est implicite.

```
aladin(nil)

  -->

e0<=drvp(f,x,2)-1/a**2*drvp(f,t,2);                    [1]
  -->
image(e0);
  -->
du<=dx+a*dt;                                           [2]
  -->
dv<=dx-a*dt;
  -->
e1<=drvp(f,u,2)*du**2+drvp(f,v,2)*dv**2+2*drvp(f,u,1,v,1)*du*dv;        [3]
  -->
e1<=smpo(e1,liste(dx,dt));                                              [4]
  -->
image(e1);
  -->

imprimer(e1);
(2*DRVP(F,U,1,V,1)+ DRVP(F,U,2) + DRVP(F,V,2))*DX**2 + (2*DRVP(F,U,2) -

 2*DRVP(F,V,2))*A*DT*DX + (-2*DRVP(F,U,1,V,1) + DRVP(F,V,2))*A**2*DT**2


e2<=imprimer(?);                                                        [5]

  2*DRVP(F,U,1,V,1) + DRVP(F,U,2) + DRVP(F,V,2)
  -->
dep(-3000,0);                                                           [6]
  -->
e3<=imprimer(a**2*?);                                                   [7]

  A**2*(-2*DRVP(F,U,1,V,1) + DRVP(F,U,2) + DRVP(F,V,2))
  -->
dep(nil,nil);


  -->
e00<=imprimer(smp(e2-1/a**2*e3));                                       [8]

  4*DRVP(F,U,1,V,1)
  -->
image(e00);
  -->
stop;
  RETOUR_A_LISP
```

$$\langle E0\rangle \quad \frac{\partial^2}{\partial X^2}F - \frac{1}{A^2} * \frac{\partial^2}{\partial T^2}F$$

$$\langle E1\rangle \quad \left[2 * \frac{\partial^2}{\partial U \partial V}F + \frac{\partial^2}{\partial U^2}F + \frac{\partial^2}{\partial V^2}F\right] * DX^2 + \left[2 * \frac{\partial^2}{\partial U^2}F - 2 * \frac{\partial^2}{\partial V^2}F\right] * A * DT$$

*Affichage de E0 et de E1*
*E1 est tronquée sur la droite*

$$2 * \frac{\partial^2}{\partial U \partial V}F + \frac{\partial^2}{\partial U^2}F + \frac{\partial^2}{\partial V^2}F$$

*1ère détection par le crayon optique*
*coefficient de DX ** 2*

$$-2 * \frac{\partial^2}{\partial V^2}F\Big] * A * DT * DX + \left[-2 * \frac{\partial^2}{\partial U \partial V}F + \frac{\partial^2}{\partial U^2}F + \frac{\partial^2}{\partial V^2}F\right] * A^2 * DT^2$$

*Déplacement de la fenêtre*

$$-2 * \frac{\partial^2}{\partial U \partial V}F + \frac{\partial^2}{\partial U^2}F + \frac{\partial^2}{\partial V^2}F$$

*2ème détection*
*.Coefficient de A ** 2 * DT ** 2.*

$$\langle E0\rangle \quad \frac{\partial^2}{\partial X^2}F - \frac{1}{A^2} * \frac{\partial^2}{\partial T^2}F$$

$$\langle E1\rangle \quad \left[2 * \frac{\partial^2}{\partial U \partial V}F + \frac{\partial^2}{\partial U^2}F + \frac{\partial^2}{\partial V^2}F\right] * DX^2 + \left[2 * \frac{\partial^2}{\partial U^2}F - 2 * \frac{\partial^2}{\partial V^2}F\right] * A * DT$$

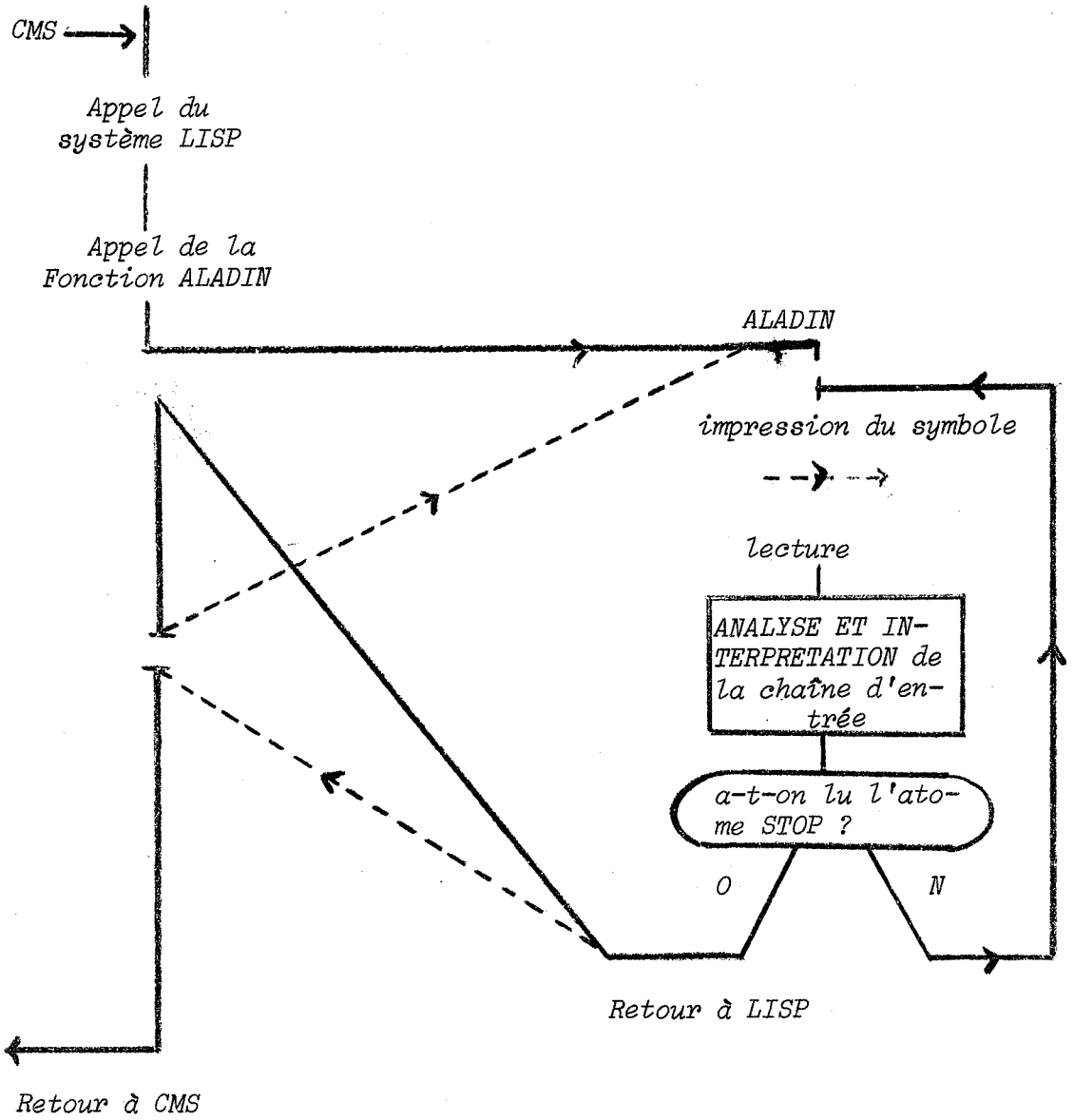
$$\langle E00\rangle \quad 4 * \frac{\partial^2}{\partial U \partial V}F$$

*Remise en place de la fenêtre et affichage de E00.*

## 3 . QUELQUES ASPECTS DE LA PROGRAMMATION DU SYSTEME ALADIN

### 3.1. ORGANISATION GENERALE

Très schématiquement, l'organisation générale est donnée par le diagramme suivant :

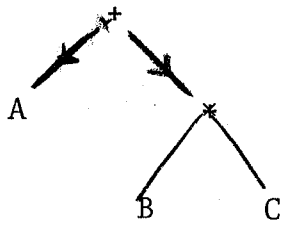


## 3.2. REPRESENTATION DES EXPRESSIONS

Les identificateurs de variables sont transformés en atomes du systèmes LISP et suivent donc la règle d'unicité : un atome non numérique est représenté en un endroit unique de la mémoire, de même les nombres sont transformés en atomes numériques pour lesquels la règle d'unicité n'est en général pas appliquée.

Quant aux expressions arithmétiques, il n'est nul besoin d'insister sur le fait que la façon la plus normale de les traiter est de les transformer en une forme préfixée figurant un graphe.

Ainsi A + B * C est ici transformée, plus précisément en l'arborescence :




qui se traduit naturellement en langage données de LISP par (+ A(* B C))

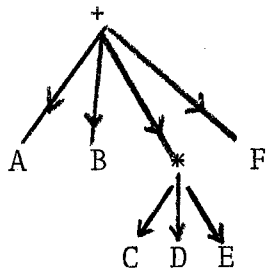
chaque noeud de l'arbre étant un opérateur que l'on place comme premier élément de chaque sous liste.

On trouvera ci-après la table de correspondance que nous avons adoptée. Dans chaque colonne les $e_i$ sont des expressions représentées dans le langage compatible avec cette colonne.

On notera que les opérateurs + et * sont n-aires. En effet, vu les propriétés de ces opérateurs, il est souhaitable de ramener leurs opérandes au même niveau de l'arborescence.

Soit par exemple l'expression A + B + C * D * E + F.

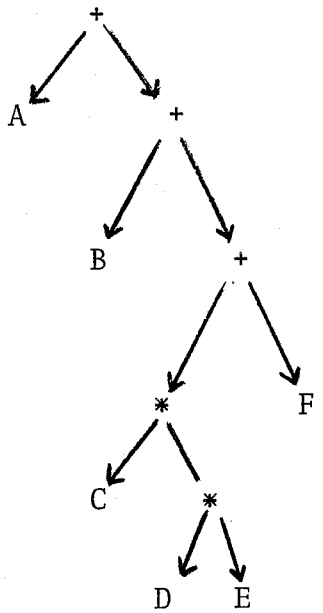
La liste (+ A B (* C D E) F) représentant l'arborescence




est plus facile à traiter (commutation, recherche d'éléments identiques, etc...)

que (+ A (+ B (+ (* C (* D E)) F)))

qui traduit :




Dans la seconde représentation les opérateurs, + et * sont binaires, mais cette représentation est cependant valide dans le système que nous avons adopté.

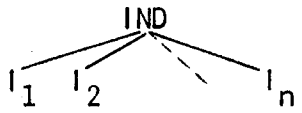
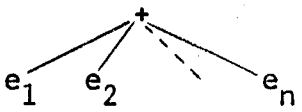
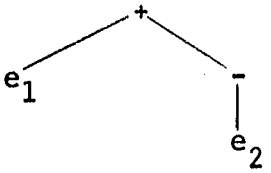
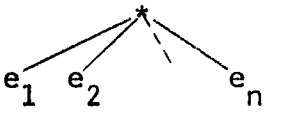
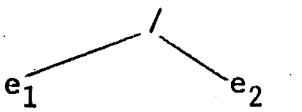
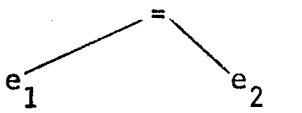
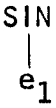
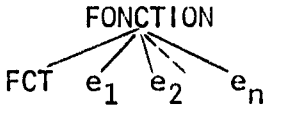
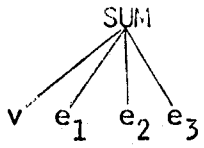
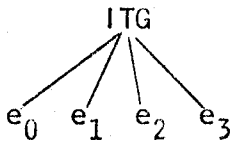
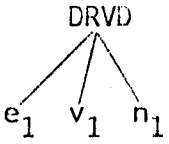
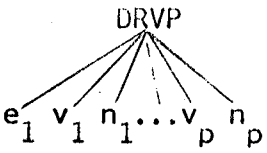
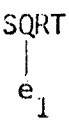
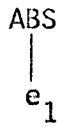
| Forme externe linéaire | Forme externe bidimensionnelle | Structure arborescente | Structure en liste |
|---|---|---|---|
| nombre identificateur | nombre identificateur | nombre identificateur | nombre identificateur |
| $V.(I_1,I_2,\ldots I_n).$ | $V_{I_1,I_2,\ldots\ldots I_n}$ |   | (IND V $I_1$ $I_2$ $\ldots I_n$) |
| $e_1+e_2+\ldots\ldots e_n$ | $e_1+e_2+\ldots\ldots e_n$ |   | (+ $e_1$ $e_2$ $\ldots\ldots e_n$) |
| $-e_1$ | $-e_1$ |   | (- $e_1$) |
| $e_1-e_2$ | $e_1-e_2$ |   | (+ $e_1$ (- $e_2$)) |
| $e_1*e_2*\ldots\ldots e_n$ | $e_1*e_2*\ldots\ldots e_n$ |   | (* $e_1$ $e_2$ $\ldots\ldots e_n$) |
| $e_1/e_2$ | $\frac{e_1}{e_2}$ |   | (/ $e_1$ $e_2$) |
| $e_1**e_2$ | $e_1^{e_2}$ |   | (** $e_1$ $e_2$) |
| $e_1=e_2$ | $e_1=e_2$ |   | (= $e_1$ $e_2$) |
| $SIN(e_1)$ | $SIN(e_1)$ |   | (SIN $e_1$) |
| la même représentation s'applique à COS,ARCTG,LN,EXP | | | |
| $FCT(e_1,e_2,\ldots\ldots e_n)$ | $FCT(e_1,e_2,\ldots e_n)$ |   | (FONCTION FCT $e_1$ $e_2$.$e_n$) |

TABLE GENERALE DE REPRESENTATION

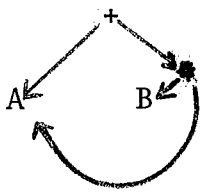
| Forme externe linéaire | Forme externe bidimensionnelle | Structure arborescente | Structure en liste |
|---|---|---|---|
| $SUM(v,e_1,e_2,e_3)$ | $\sum_{v=e_1}^{e_2} e_3$ |   | (SUM v $e_1$ $e_2$ $e_3$) |
| $ITG(e_0,e_1,e_2,e_3)$ | $\int_{e_1}^{e_2} e_3 \, de_0$ |   | (ITG $e_0$ $e_1$ $e_2$ $e_3$) |
| $DRVD(e_1,v_1,n_1)$ | $\frac{d^{n_1}}{dv_1^{n_1}} e_1$ |   | (DRVD $e_1$ $v_1$ $n_1$) |
| $DRVP(e_1,v_1,n_1..v_p,n_p)$ | $\frac{\partial^{n_1+n_2+\ldots}}{\partial v_1^{n_1} \ldots \partial v_p^{n_p}} e_1$ |   | (DRVP $e_1$ $v_1$ $n_1$..$v_p$ $n_p$ |
| $SQRT(e_1)$ | $\sqrt{e_1}$ |   | (SQRT $e_1$) |
| $ABS(e_1)$ | $\lvert e_1 \rvert$ |   | (ABS $e_1$) |

(SUITE)

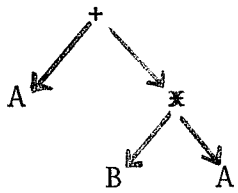
Précisions sur la représentation interne

En LISP, les atomes non numériques, sont représentés en un seul endroit de la mémoire. Donc une liste qui possède des éléments atomiques identiques, est en fait représentée en mémoire par une structure qui est un graphe sans circuit, et non une arborescence.

EXEMPLE : La façon correcte de noter la description en mémoire de la liste (+A(*B A))est

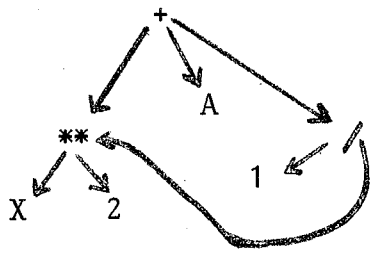



est non




Plus généralement, une liste peut avoir des éléments non atomiques identiques. On peut réaliser une économie de mémoire, en n'ayant qu'une copie de l'élément commun.

EXEMPLE la formule $X^2 + A + \frac{1}{X^2}$ peut avoir le graphe suivant :



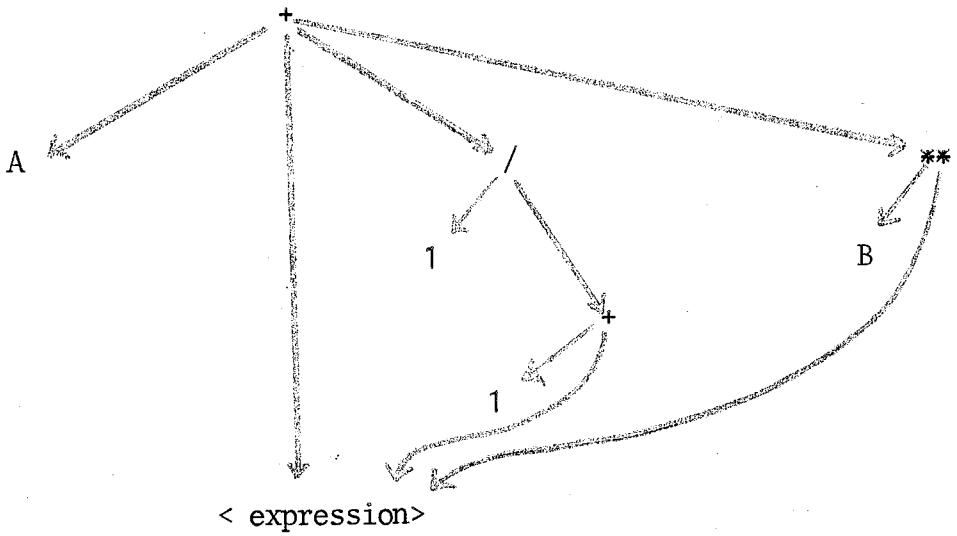

qui traduit la représentation la plus économique.

Le cas se produit en particulier dans une expression formée à l'aide d'une variable affectée antérieurement.

Ainsi après : S0 <== <expression> ;

S1 <== A+S0+1/(1+S0)+B**S0 ;

Le graphe associé à S1 sera :



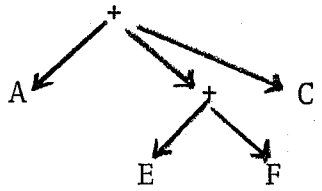

Cette économie de mémoire est conservée, si l'on substitue une formule à<expression>par la commande REMPLACER.

Cependant, l'application de la commande RPLOC demande une précaution : il faut en effet que la sous-expression à remplacer n'appartienne qu'à une seule variable affectée, et que dans le graphe associé à cette variable, elle ne soit accessible, du noeud principal, que par un seul chemin. On procède à une recopie de la formule sur laquelle la détection a été faite, de façon que les seuls éléments communs soient les atomes. La substitution se fera alors sans problème puisque aux atomes près, les sous expressions ont retrouvé leur individualité. La nécessité d'une recopie, qui engendre un graphe équivallent est bien connue des programmeurs qui font du traitement de listes.

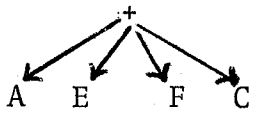
Notons au passage, que si dans une somme on remplace un élément par une autre somme, la structure supérieure est inchangée.

EXEMPLE : S3 <== REMPLACER (B,E+F,A+B+C) ;

donne




et non pas




Bien que sur l'écran S3 soit écrit A+E+F+C, l'élimination du noeud redondant, ne se fera que si l'on applique une commande SMP ou SMPO.

La même remarque s'applique à un produit. Mais aussi longtemps qu'une fonction de simplification n'est pas appelée, le groupement E+F est accessible (par exemple au crayon optique) car il constitue un sous graphe bien précis, alors qu'un groupement comme A+E, ou A+C n'a pas d'existence en tant que sous graphe.



EXEMPLE :

REMPLACER (E+F,K**2,S3) donne A+K**2+C

alors que

REMPLACER (A+B,D,A+B+C) donne A+B+C

(aucun changement)

## 3.3. LE TRADUCTEUR

La partie centrale d'Aladin est le traducteur dont la fonction est de transformer en notation préfixée la chaîne d'entrée frappée au clavier de la console 2741 ou à celui du terminal 2250, en tenant compte, s'il y a lieu, des transformations impliquées par les commandes.

### 3.3.1. Phase d'édition

La chaîne d'entrée est traitée caractère par caractère. Cette édition a pour but :

- de fabriquer les atomes LISP représentant les identificateurs et les nombres.
- de reconnaître les symboles : <=, 'I', .(,). et de fabriquer les atomes correspondants.
- de traiter les caractères illégaux ou mal placés.

En fait l'éditeur est un sous programme du traducteur.

### 3.3.2. Phase de transformation

La syntaxe étant tout à fait simple, nous avons utilisé la méthode bien connue des priorités, qui utilise deux piles : une pile pour les opérandes, une autre pour les opérateurs. Cette méthode est décrite en [47] et [48]. Les piles seront en fait des listes.

Traitement des opérandes. Si un atome n'a pas reçu de valeur, il est placé tel quel, en tête de la pile opérande, sinon on y met la liste qui a été attachée à cet atome par une instruction d'affectation antérieure. Dans le cas de l'atome spécial ?, le traducteur donne le contrôle à un sous-programme qui attend une détection par crayon optique. Après le traitement de l'interruption provoquée par cette détection, ce sous programme déterminera quel sous graphe correspond à ce qui a été désigné par l'utilisateur. La liste qui représente ce sous-graphe, sera insérée dans la pile des opérandes. La méthode de liaison entre la détection optique et la liste qui représente une expression sera expliquée plus loin.

traitement des opérateurs Les opérateurs entrent et sortent de la pile des opérateurs en fonction de leur priorité, définies dans la table suivante :

| OPERATEUR | PRIORITE |
|---|---|
| ; | 0 |
| <= | 1 |
| = | 2 |
| , | 3 |
| + | 4 |
| - | 4 |
| * | 5 |
| / | 5 |
| ** | 6 |
| moins unitaire(-) | 7 |
| FONCTION | 7 |
| IND | |
| ITG | |
| : | |
| fonctions standards | |
| commandes | 7 |

NOTE : *Les symboles <(>, <)>, <.(>,<).> ne sont pas des opérateurs. les sous expressions, qu'ils délimitent sont transformées par un appel récursif du traducteur.*

Les commandes sont empilées, puisque syntaxiquement ce sont des opérateurs. Elles demeurent dans la pile des opérateurs tant que leurs arguments n'ont pas été transformés.

Quand l'opérateur qui sort de la pile où il avait été rangé, n'est pas une commande, on génère alors une sous liste en fonction des règles de production attachées à cet opérateur et on la place dans la pile des opérandes.

Par contre, si c'est une commande, celle-ci est appliquée à ses arguments, qui sont, par le déroulement même de l'algorithme de transformation, placés nécessairement en tête de la pile des opérandes.

EXEMPLE : soit

EO <= A+REMPLACER(X,Y,SIN(Z+X))-....

le début d'une chaîne, envoyée par l'utilisateur.

Lorsque le traducteur traitera le signe <-> , le sommet de la pile des opérandes sera : (SIN(+Z Y)) parce que la commande remplacer a été appliquée au cours même de la transformation.

Cas particulier de la commande d'affectation Il s'agit là d'un exemple de commande dont l'un des arguments (l'identificateur qui apparait en partie gauche du signe <=) est pris en tant que tel sans qu'on se soucie de la valeur qu'on aurait pu lui affecter précédemment. Du point de vue de LISP, la liste résultant de la transformation de la partie droite, est attachée comme propriété à l'atome affecté. On verra plus loin un autre exemple, de l'emploi des listes de propriétés, technique très fréquente en programmation LISP.

## 3.4. REMARQUES SUR LES SOUS-PROGRAMMES DE SIMPLIFICATION

Dans le système Aladin, la simplification est réalisée par une adaptation des sous-programmes de Mathlab qui traitent les fonctions

rationnelles. Il s'agit là d'un 'emprunt' à un autre système. Le principe employé est trivial : pour simplifier on fait un appel à REP suivi d'un appel à DISREP. Rappelons que la fonction REP transforme une fraction rationnelle écrite sous préfixée en sa forme récursive et que la fonction DISREP fait l'opération inverse.

La liste des éléments en lesquels une expression est rationnelle notée VARLIST est déterminée par la fonction auxiliaire REPVAR.

Les éléments de cette liste sont alors ordonnés suivant des règles expliquées plus loin. L'ordre établi par ces règles est susceptible d'être modifié : si l'expression est rationnelle en 'I' auquel cas 'I' est toujours considéré comme la 'variable' la plus importante ou encore si c'est la fonction SMPO qui est utilisée.

Dans ce second cas - sauf cependant pour la variable 'I'-l'utilisateur peut imposer un ordre préférentiel sur tout ou partie des éléments de VARLIST. S'il ne s'agit que d'un sous-ensemble, le complément est ordonné suivant l'ordre canonique.

Nous donnons quelques exemples de la valeur de VARLIST dans quelques cas de simplification. On rappelle que la variable la moins importante est placée en tête :

```
imprimer(smp(a*x+b-h+sin(x)-y**n1));
VARLIST = ((** Y N1) (SIN X) X H B A)
e0<=(a*x+b)*(c-sin(x)+a)*(fct(x)+y1-g3+x**x-y**x);
e5<=imprimer(smp(e0));
VARLIST = ((FONCTION FCT X) (** Y X) (** X X) (SIN X) Y1 G3 X C B A)
e5<=imprimer(smpo(e0,liste(x,fct(x),g3)));
VARLIST = ((** Y X) (** X X) (SIN X) Y1 C B A G3 (FONCTION FCT X) X)
Soit e00 l'expression suivante.
          ('I'*(A*X + B))*(C - SIN(X) + A)*(FCT (X) + Y1 - G3 + 'I'+X**X - Y**X)
e5<=imprimer(smp(e00));
```

```
VARLIST = ((FONCTION FCT X)(** Y X)(** X X)(SIN X) Y1 G3 X C B A 'I')
e5<=imprimer(smpo(e00,liste(x**x,g3,'I',y1)));
VARLIST = ((FONCTION FCT X)(** Y X)(SIN X) X C B A Y1 G3 (** X X) 'I')
```

Ordre canonique sur l'ensemble des expressions symboliques LISP

L'ensemble des expressions symboliques LISP peut être totalement ordonné de la façon suivante :

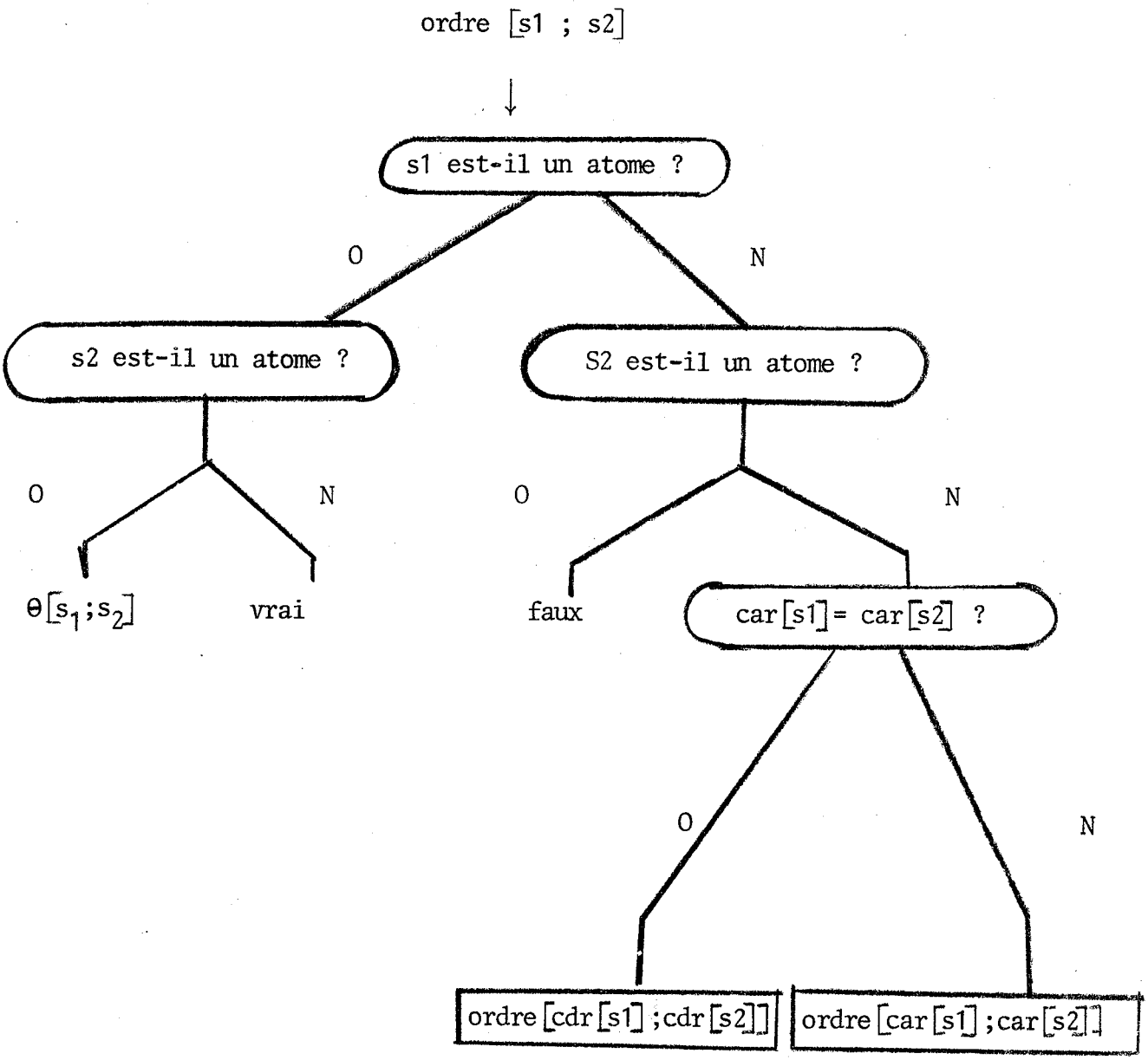
Supposons qu'il existe un ordre sur l'ensemble des atomes, c'est à dire qu'il existe une fonction $\Theta$ à deux arguments atomiques telle que :

$\forall$ x, y, z
$\Theta$ (x,x) = vrai
$\Theta$ (x,y) = vrai entraine $\Theta$ (y,x) = faux pour $x \neq y$
$\Theta$ (x,y) $\wedge$ $\Theta$ (y,z) = vrai entraine $\Theta$ (x,z) = vrai

Etant donné deux expressions symboliques, on peut écrire en LISP, une fonction qui les ordonne. En métalangage LISP, cette fonction peut être définie ainsi :

```
label [ordre ; λ[[s1;s2] ;
      [atom [s1] → [atom [s2] → Θ [s1 ; s2] ;
                   T      →      T] ;
       atom [s2]] → NIL ;
       equal[car[s1] ; car[s2] → ordre [cdr[s1] ; cdr[s2]] ;
         T      →        ordre[car[s1] ; car[s2]]]]]
```

Cette définition récursive traduit l'organigramme suivant :


ordre [s1 ; s2]
s1 est-il un atome ?
O
N
s2 est-il un atome ?
S2 est-il un atome ?
O
N
O
N
θ[s1;s2]
vrai
faux
car[s1]= car[s2] ?
O
N
ordre [cdr[s1] ;cdr[s2]]
ordre [car[s1] ;car[s2]]

Dans la pratique, cette fonction est utilisée pour ordonner un sous ensemble fini d'expressions symboliques. La fonction $\Theta$ [x ; y] est en fait dans le système LISP, la fonction appelée orderp [x ; y] qui, pour les atomes à une lettre conduit à l'ordre lexicographique, et pour les autres, à un ordre lié aux adresses de rangement des atomes dans la mémoire.

Etant donnée une liste $(e_1, e_2, e_3, \ldots, e_n)$ ayant éventuellement des éléments identiques, notre fonction nous permettra, de trouver la permutation qui réordonne cette liste de façon que l'ordre canonique soit respecté.

## 4 . LE SYSTEME DE VISUALISATION

La visualisation des formules sous l'aspect qui nous est familier, c'est-à-dire en deux dimensions, n'est pas un élément essentiel pour les systèmes de manipulation algébrique. Cependant, dans la mesure où la visualisation sur un écran n'est pas seulement une sortie, mais peut servir pour l'entrée, au moins dans le sens où nous l'entendons dans Aladin, la façon de résoudre certains problèmes qui s'y rattachent, mérite d'être étudiée en soi.

Pour passer de la représentation arborescente des formules à leur visualisation, nous avons utilisé un langage de description de dessin dans un plan.

### 4.1. UN LANGAGE GRAPHIQUE ELEMENTAIRE

#### 4.1.1. Introduction

Etant donné un plan, muni d'un système de coordonnées cartésiennes, le langage graphique que nous allons décrire, donne un moyen très simple de tracer des segments de droite, d'écrire du texte, de reproduire des dessins en divers endroits du plan etc...

Pour plus de commodité, nous le désignerons par L.G. Dans ce langage, une image est décrite par un ensemble de "composants". Ces composants correspondent à des instructions qui déplacent un point (spot) dans le plan. La description de l'image est indépendante du support physique de visualisation (tube cathodique, traceur de courbe), qui agit comme une fenêtre sur le plan indéfini.

Cette description sera confiée à un compilateur qui produira les instructions effectives pour le dispositif physique et résoudra, le cas échéant, les problèmes posés par la fenêtre.

### 4.1.2. Forme externe du langage graphique

Le langage L.G. a une structure en liste, qui facilite l'écriture du compilateur. Les notions d'expressions arithmétiques et booléennes sont celles du langage LISP. Nous utiliserons quelques facilités méta-syntaxiques telles que

<S> ... <S>

Un tel assemblage représente une suite quelconque, éventuellement vide de <S>.

<image> ::= (<nom d'image>(<liste de variables><composant>))

<composant> ::= (SKIP <déplacement><déplacement>)|
(SKIPD <déplacement><déplacement>)|
(RVECTA <déplacement><déplacement>)|
(RVECTAD <déplacement><déplacement>)|
( RPOINTA <déplacement><déplacement>)|
(RPOINTAD<déplacement><déplacement>)|
(RTEXTG <chaine>)| (RTEXTP <chaine>|
(COMP <composant> ... <composant>)|
(COMPD<composant> ... <composant>)|
(COND (<expression booléenne><composant>) ... (<expression booléenne><composant>))|
(<nom d'image><argument> ... <argument>)

<nom d'image> ::=<identificateur>

<liste de variable> ::= (<identificateur> ... <identificateur>)

<déplacement> ::= <identificateur>|<nombre entier>|<expression arithmétique entière>

<chaîne> ::= <identificateur>|<nombre>| (QUOTE <expression symbolique LISP>)

<argument> ::= <déplacement>|<chaîne>|<variable booléenne>

### 4.1.3. Interprétation

La valeur de <déplacement> doit être un nombre entier. Supposons que le spot fictif soit au point de coordonnées $x_o, y_o$, nous allons donner l'interprétation de la notion <composant> :

(SKIP x y) : placer le spot au point (x,y).

(SKIPD x y) : placer le spot au point $(x_o+x, y_o+y)$

(RVECTA x y) : tracer un segment visible du point $(x_o,y_o)$ au point qui devient la nouvelle position du spot.

(RVECTAD x y) : même chose, mais le point d'arrivée est $(x_o+x, y_o+y)$

(RPOINTA x+y)
(RPOINTAD x y) : afficher un point en (x,y) et $(x_o+x, y_o+y)$ respectivement.

(RTEXTG <chaîne>): écrire le texte <chaîne> sur la ligne $y=y_o$. Quand cet ordre a été exécuté l'abscisse du spot est : $x_o$ + longueur de la chaîne * largeur d'un caractère.

(RTEXTP <chaîne>) :même ordre, mais avec un modèle de caractère plus petit :
NOTE : *Pour ces 2 ordres, la taille des caractères pourra être imposée par le dispositif de sortie.*

(CØMP <composant> ... <composant>) : Définition récursive d'un composant. A l'entrée de chaque<composant> , le spot est placé au point où le laisse le <composant> précédent. A la fin de l'opération le spot reste placé au point que lui assigne l'exécution de l'instruction la plus à droite.

(CØMPD <composant> ... <composant>) : Définition identique à celles qui précède, mais les déplacements qui interviennent dans les différents <composant> sont relatifs au point $x_0, y_0$.

(CØND(<expression booléenne><composant>)
•••
(<expression booléenne><composant>)). Cette instruction permet un choix conditionnel. Les doublets (<expression booleenne><composant>) sont pris en séquence. Dès que la partie booléenne est vraie, le composant correspondant définit l'action à entreprendre, à l'exclusion de toute autre. Si aucune partie booléenne n'est vraie, l'instruction est vide, c'est à dire que le spot ne bouge pas.

(<nom d'image><argument> ... <argument>)

Ce type d'instruction est analogue à un appel de fonction. Les arguments sont évalués et affectés aux différentes variables associées à la définition de l'image. De même que pour COMPD, le point $(x_0, y_0)$, c'est à dire la position du spot, au moment de l'interprétation de ce composant joue le rôle d'origine pour tous les déplacements intervenant dans les instructions de la définition. Par exemple, pour replacer le spot au point $(x_0, y_0)$, la dernière instruction de l'image peut être (SKIP 0 0)

### 4.1.4. Implantation du langage L.G.

La définition et la compilation d'une image est contrôlée par le système LISP. La visualisation est faite sur une console IBM 2250

Modèle 1. L'enchainement des différents processus est le suivant :

#### 4.1.4.1. Définition des images

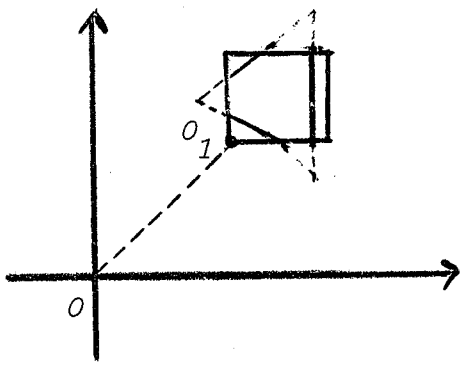
Par la fonction défimage [ℓ], où

ℓ = (<image><image> ... <image>)

on associe à chaque <nom d'image> qui est un atome LISP, la liste ( <liste de variable><composant>). Cette association se fait évidemment par la technique des listes de propriétés.

#### 4.1.4.2. Compilation

La compilation est effectuée par la fonction compimage [e] où e est nécessairement un nom d'image. Cette fonction produit une liste d'ordres symboliques pour la console 2250. Cette liste dépend des coordonnées de la fenêtre dans le plan, : ces coordonnées sont en fait les coordonnées du coin "Sud Ouest" de la fenêtre ($O_1$ sur le croquis).




Le compilateur ne générera que les ordres qui correspondent à ce qui est vu dans la fenêtre. Naturellement, au moment de la compilation, les différents identificateurs devront avoir reçu une valeur effective.

#### 4.1.4.3. Assemblage

Tout en étant très proche du langage binaire de la console 2250, la liste d'ordres est cependant symbolique. Nous verrons plus loin la raison de ce choix. Elle est confiée à une fonction d'assemblage élémentaire qui produit une liste de nombres logiques LISP, qui sont

les profils binaires des ordres de la console de visualisation.

#### 4.1.4.4. Chargement et visualisation

Enfin, les nombres binaires sont placés en séquence dans une zone de la mémoire centrale. De là, le contenu de la zone, qui est un programme exécutable par la console 2250, est envoyé, par un interface d'entrée-sortie, dans sa mémoire interne : l'éxécution de ce programme donne précisément la visualisation recherchée.

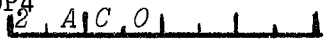
#### 4.1.4.5. Description sommaire de la console IBM 2250

Ce paragraphe très technique est cependant nécessaire pour comprendre le mode de liaison entre une formule et le symbole ? que nous verrons plus loin.

Le terminal IBM 2250 modèle 1, est une machine qui peut exécuter un programme chargé dans sa mémoire interne. Cette exécution provoque le déplacement d'un spot lumineux sur un écran dans toutes les directions ou l'écriture de caractères alphanumériques sur une ligne horizontale. La stabilité de l'image programmée est obtenue par bouclage.

Un programme pour le 2250 comporte des commandes et des données. Grossièrement une commande est une "clé" qui change l'état de la machine déterminant ainsi une interprétation spécifique des informations binaires qui suivent, jusqu'à ce qu'une autre commande soit rencontrée. Un sous-ensemble des commandes est suffisant pour le langage L.G. Nous le décrivons ci-après. Le profil binaire est donné en notation hexadécimale.

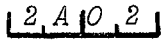
GNOP4 <octets libres>



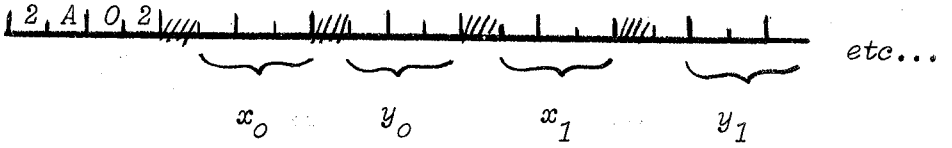

Cette commande est ineffective. Les 2 octets libres peuvent être garnis d'informations utiles.

Symboliquement nous générerons

(GNOP4 n) ou (GNOP4 n m), si nous voulons insérer les quantités n et m.

GEVM 


Cette commande provoque l'interprétation des données qui suivent comme étant les coordonnées successives du spot.
Ces données sont organisées comme suit :




Les $x_i$ et $y_i$ sont codés sur 12 bits, c'est à dire qu'ils varient de 0 à 4095. Dans la place qui reste, un bit est prévu pour indiquer si le déplacement du spot doit être visible (segment) ou invisible (saut).

Bien qu'il n'y ait qu'une commande pour un ensemble de déplacements du spot, le compilateur compimage, générera :

...(GEVM) (GDV $x_0$ $y_0$ <mode>) (GDV $x_1$ $y_1$ <mode>) ...

avec <mode> = U pour un segment
B pour un saut.

Le symbole GDV nous sert à reconnaître les sous listes qui correspondent à des coordonnées.

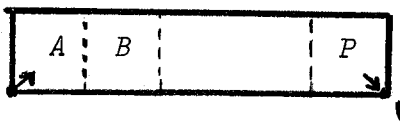
GECP 2,A,4,0

*et*

2,A,4,1

Cette commande met la machine 2250 en mode affichage de caractères

C'est à dire : les octets suivants sont interprétés comme des caractères et affichés horizontalement sur l'écran. Deux tailles de caractères sont disponibles : la taille standard (petits caractères) et la grande taille. Les commandes correspondantes sont 2A40 et 2A41. Les coordonnées du spot, au moment du changement d'état sont considérées comme le centre du caractère. Cependant, la compilation d'un composant (RTEXTG <chaîne>), générera les sauts nécessaires de façon que l'écriture de la chaîne se fasse comme indiqué :




*abscisse du spot au début* — *abscisse du spot à la fin*

Symboliquement nous générerons :

(GECP <mode>)

(<liste de caractères>)

avec <mode> { B pour les petits caractères
L pour les grands ".

Enfin certaines commandes de contrôle sont utilisées :

GSRT pour le contrôle de régénération (horloge)
GTRU pour les transferts inconditionnels
GESD pour autoriser les détections au crayon optique
GDPD pour interdire " " " "

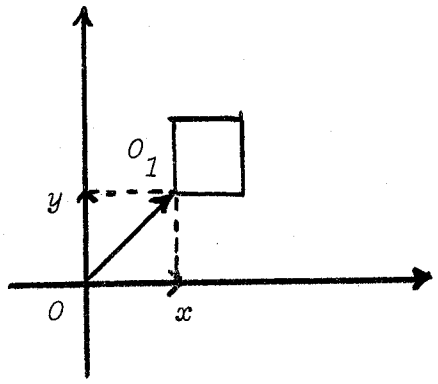
#### 4.1.4.6. Exemple complet

Nous allons définir, compiler, et afficher le sigle bien connu des Centres Scientifiques IBM. Sa définition reflète les propriétés eulériennes de son tracé.

L'image qui sera compilée est celle qui est nommée CARLOS et dont le composant principal est un appel à l'image CARRE, munie d'arguments numériques : le côté du carré extérieur et le nombre de carrés imbriqués. (>0)

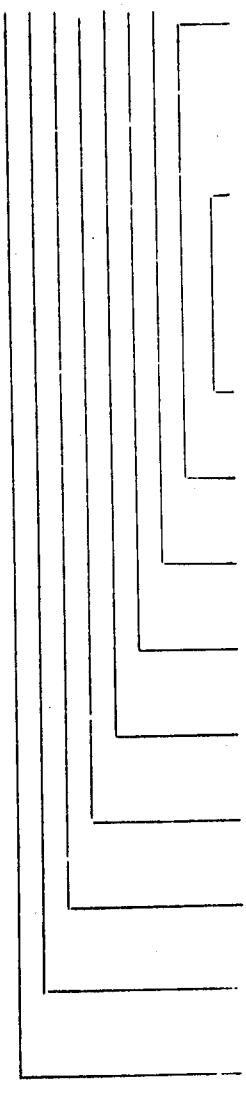
```
defimage ((
     (carre (x n)
        (comp (rvecta x 0)
              (rvecta x x) (rvecta 0 x)
              (rvecta 0 (quotient x 2))
              (cond ((not (equal n 1))(losange (quotient x 2))))
              (rvecta 0 0)
        )
     )
     (losange (x)
        (comp(rvecta x x)
             (rvecta (times 2 x) 0)
             (rvecta x (minus x))
             (rvecta (quotient x 2) (minus (quotient x 2)))
             (carre x (sub1 n))
             (rvecta 0 0)
        )
     )
     (carlos () (comp(skip 0 0)(carre 4000 5)))
          ))
```

La place de la fenêtre dans le plan, est assignée par un appel à la fonction origplay [x;y]



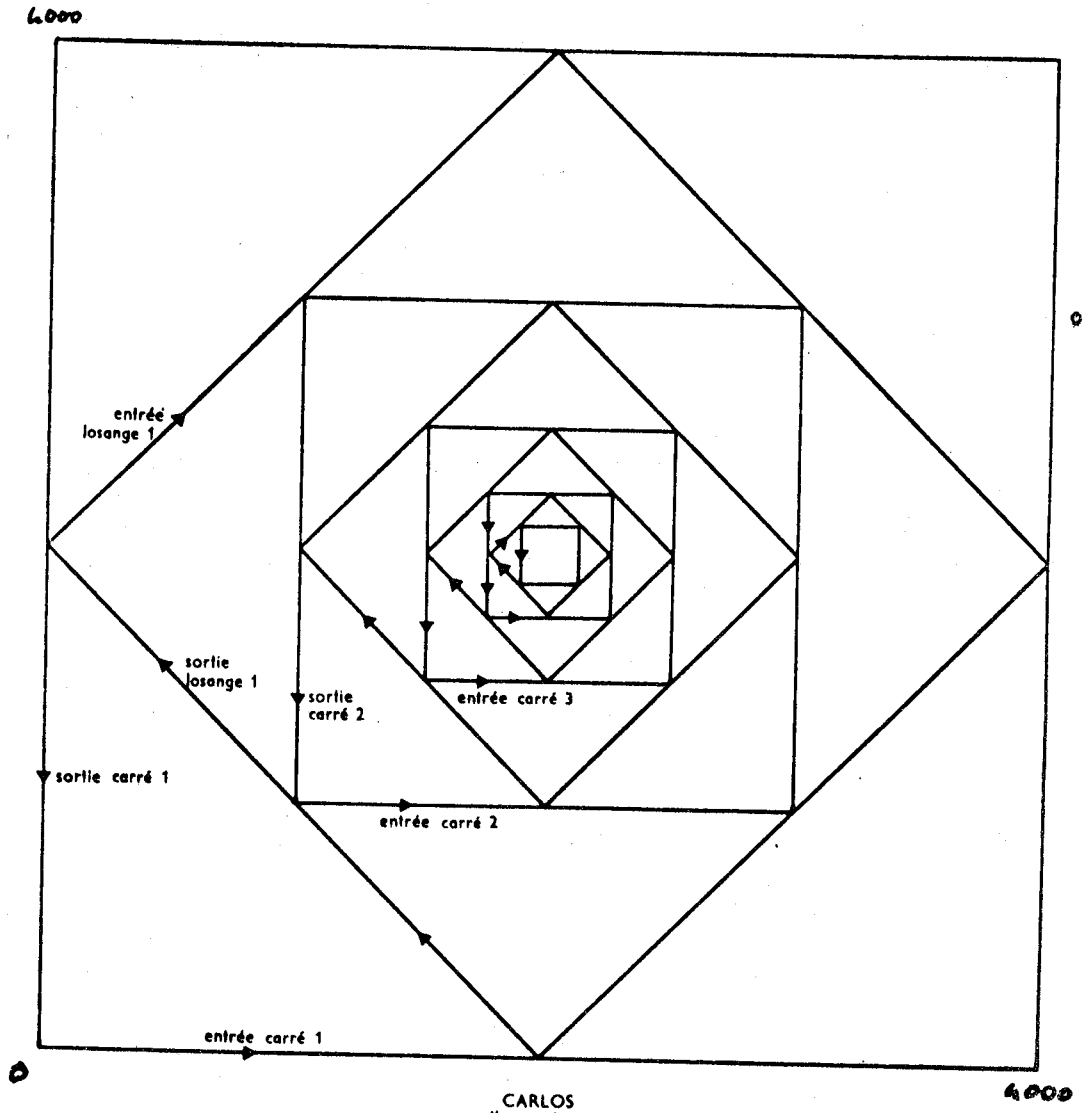

En faisant coincider O et O1, et en compilant l'image carlos, on obtient la séquence d'ordres ci-jointe. qui traduit exactement l'imbrication des carrés et des losanges impliquée par la récursivité croisée de la définition.

```
CARLOS  GSRT
        GNOP4   0                       entrée carré n° 1
        GEVM
        GDV     4000,0,U
        GDV     4000,4000,U
        GDV     0,4000,U
        GDV     0,2000,U
        GNOP4   0                       entrée losange n° 1
        GEVM
        GDV     2000,4000,U
        GDV     4000,2000,U
        GDV     2000,0,U
        GDV     1000,1000,U
        GNOP4   0                       entrée carré n° 2
        GEVM
        GDV     3000,1000,U
        GDV     3000,3000,U
        GDV     1000,3000,U
        GDV     1000,2000,U
        GNOP4   0
        GEVM
        GDV     2000,3000,U
        GDV     3000,2000,U
        GDV     2000,1000,U
        GDV     1500,1500,U
        GNOP4   0
        GEVM
        GDV     2500,1500,U
        GDV     2500,2500,U
        GDV     1500,2500,U
        GDV     1500,2000,U
        GNOP4   0
        GEVM
        GDV     2000,2500,U
        GDV     2500,2000,U
        GDV     2000,1500,U
        GDV     1750,1750,U
        GNOP4   0
        GEVM
        GDV     2250,1750,U
        GDV     2250,2250,U
        GDV     1750,2250,U
        GDV     1750,2000,U
```

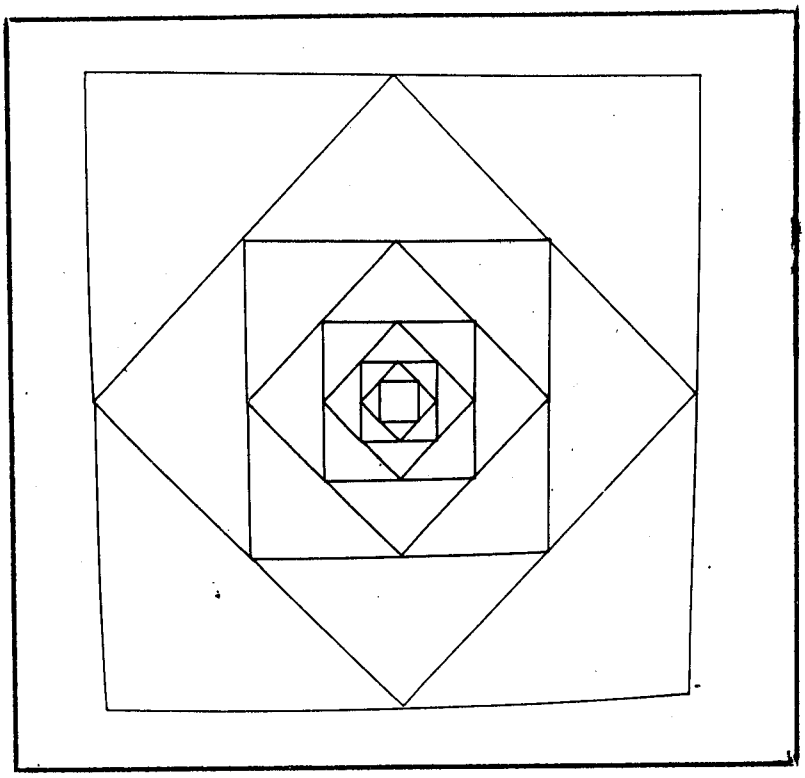


```
GNOP4   0
GEVM
GDV     2000,2250,U
GDV     2250,2000,U
GDV     2000,1750,U
GDV     1875,1875,U
GNOP4   0                       entrée carré n° 5
GEVM
GDV     2125,1875,U
GDV     2125,2125,U
GDV     1875,2125,U
GDV     1875,2000,U
GDV     1875,1875,U
GNOP4   1                       sortie carré n° 5
GEVM
GDV     1750,2000,U
GNOP4   1
GEVM
GDV     1750,1750,U
GNOP4   1
GEVM
GDV     1500,2000,U
GNOP4   1
GEVM
GDV     1500,1500,U
GNOP4   1
GEVM
GDV     1000,2000,U
GNOP4   1
GEVM
GDV     1000,1000,U
GNOP4   1
GEVM
GDV     0,2000,U
GNOP4   1                       sortie losange n° 1
GEVM
GDV     0,0,U
GNOP4   1                       sortie carré n° 1
GTRU    CARLOS
```

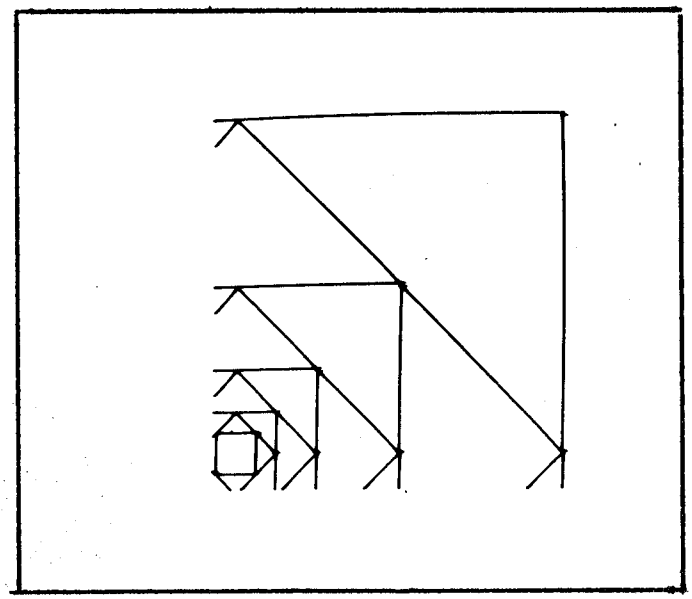
Cet exemple montre en particulier que les "sous-programmes image" sont en réalité des macro-instructions.




CARLOS
« d'un seul trait »



*Carlos*



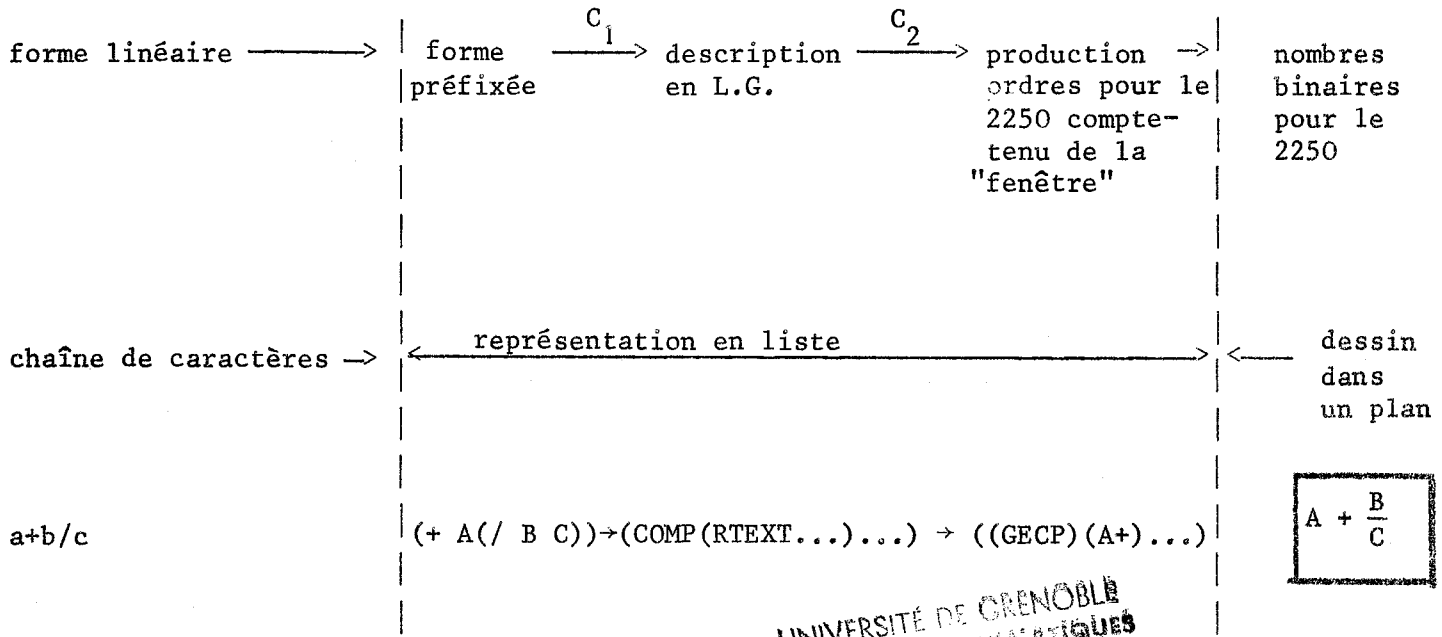
*Déplacement de la fenêtre dans le plan*
*Le dessin est tronqué*

## 4.2. UTILISATION DE L.G. POUR DECRIRE LES TRACES DES FORMULES SUR UN PLAN

### 4.2.1. Introduction

La description d'un tracé de formule en deux dimensions est un cas particulier d'utilisation de notre langage graphique. En fait cette description sera produite par un autre compilateur qui prendra comme donnée la représentation en liste d'une formule préfixée et produira la description en L.G. du dessin usuel de cette formule. Ainsi, L.G. sert de langage intermédiaire entre une structure et sa visualisation. C'est pourquoi sa forme en liste, lourdre pour programmer, est non seulement suffisante, mais encore adéquate pour un traitement interne.

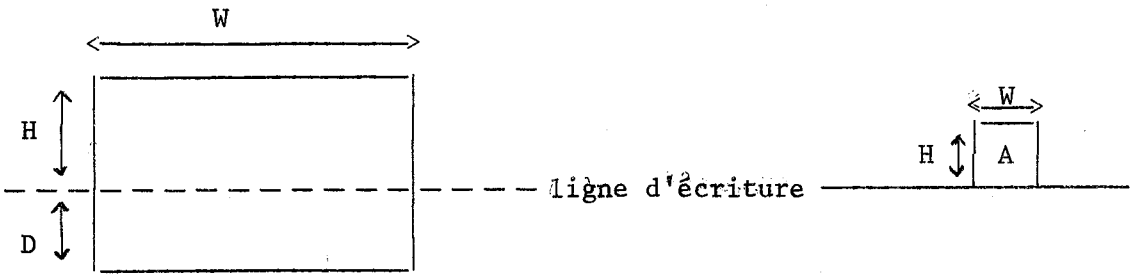
Le passage de la forme linéaire d'une formule à sa forme bidimentionnelle peut être schématisée ainsi :



Le compilateur $C_1$ produit une description indépendante de la fenêtre et c'est le compilateur $C_2$ qui résoudra les problèmes liés au support physique.

### 4.2.1. Principe du compilateur $C_1$

L'idée originale peut être trouvée dans les références [31], [25]. Dans un plan indéfini, où la limitation de place se pose pas, nous pouvons choisir une ligne d'écriture et considérer qu'une formule peut toujours être inscrite dans un rectangle. Une certaine partie du rectangle est au dessus de la ligne d'écriture, une autre éventuellement en dessous. D'où les 3 grandeurs W,D,H.




Cette façon de concevoir les choses est valable jusqu'au caractère, auquel cas : W = largeur du caractère
H = hauteur " "
D = 0

Le dessin d'une formule a une forme déterminée par son opérateur principal. Aussi peut-on dessiner à priori, les types de formules courantes indépendamment des arguments. Une formule n'est alors qu'un assemblage de rectangles. Dans sa thèse [33], Anderson montre que ces assemblages peuvent être engendrés par une grammaire qui généralise les grammaires context-free. En effet l'opération de concaténation n'est qu'un cas particulier d'assemblage d'éléments entre eux. Nous n'avons pas besoin ici d'une telle généralité, nécessitée chez Anderson pour résoudre le

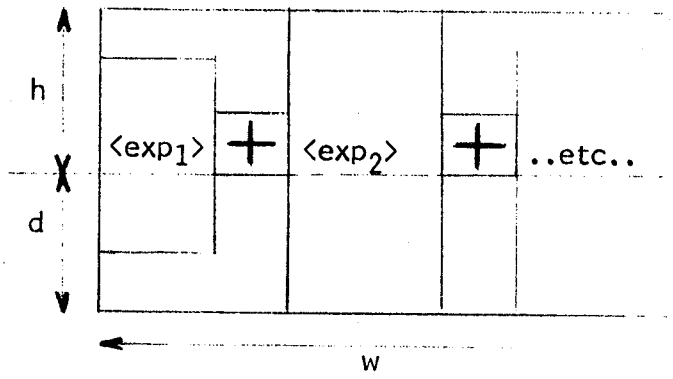
# DISPOSITIONS BIDIMENSIONNELLES ASSOCIEES AUX FORMULES

*identificateur ex : film*

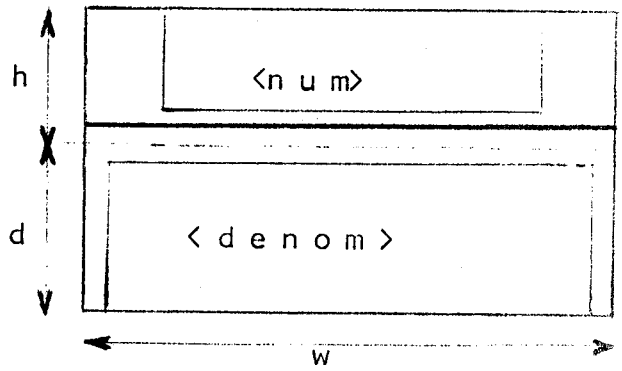
FILM

FILM

*(+<exp 1><exp 2>...)*
*concaténation d'un nombre quelconque de termes. La même disposition est valable pour **

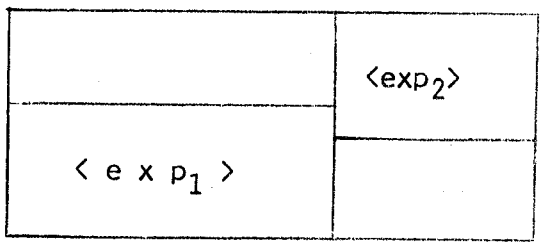



*(/<num><denom>)*



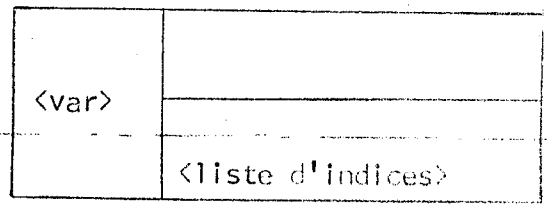

*(**<exp 1><exp 2>)*

*les identificateurs dans <exp2> sont en petits caractères*



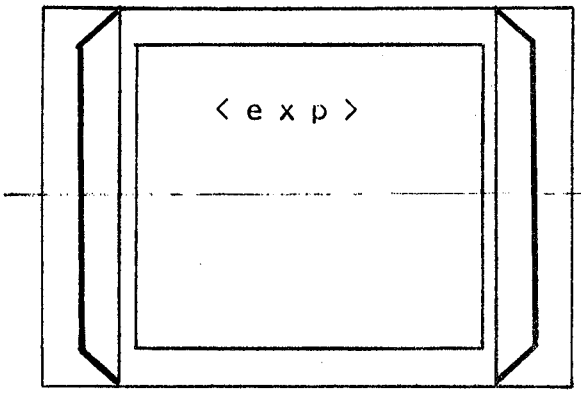

*(IND <var><ind 1><ind 2>)*

*Le rectangle qui contient la liste d'indice est construit comme le rectangle relatif au +, mais la virgule remplace le signe + et les identificateurs sont en petits caractères.*

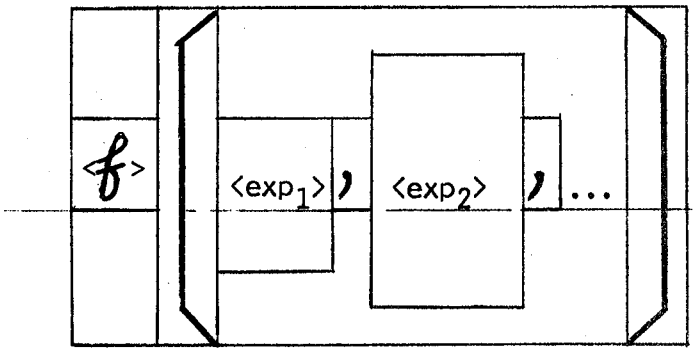


*cas de parenthèses nécessaires :*

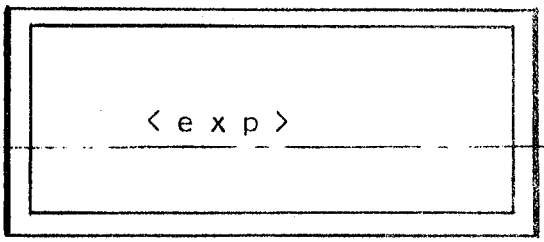
*Elles sont adaptées à la formule qu'elles délimitent*



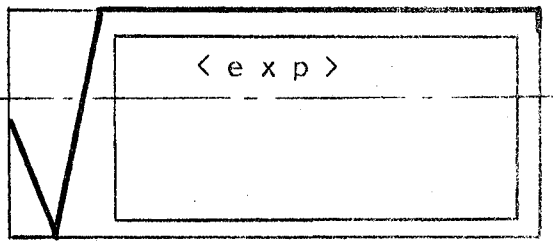

*(FONCTION <f><exp$_1$><exp$_2$>...)*



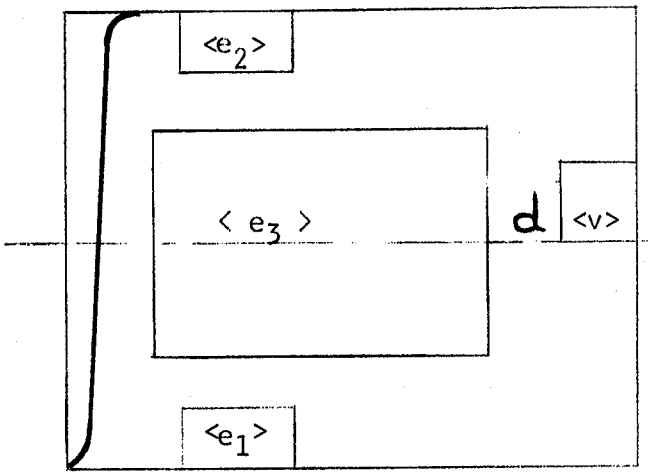

*(ABS<exp>)*



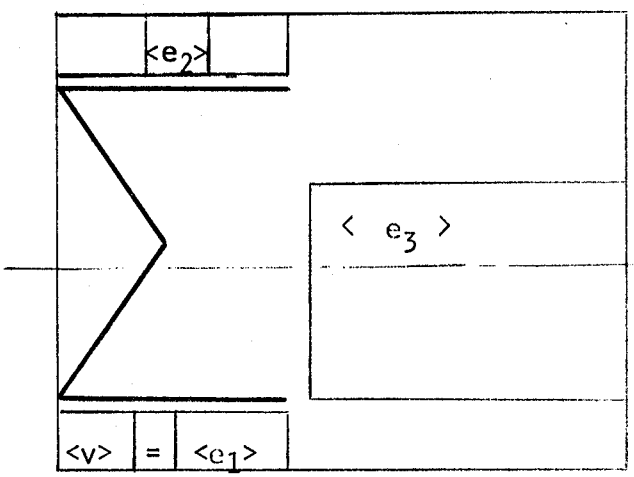

*(SQRT<exp>)*



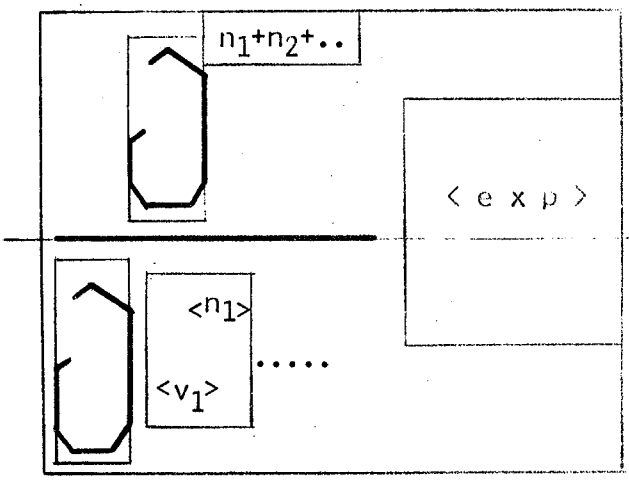
(ITG <v><e 1><e 2><e 3>)




(SUM <v><e 1><e2><e3>)




(DRVP <exp><v 1><n 1><v 2><n 2>...)



problème inverse, à savoir l'analyse d'une formule écrite en deux dimensions sur un plan. La résolution de ce problème lui a permis de franchir un nouveau pas, dans les facilités offertes à l'ingénieur.

### 4.2.2. La compilation

Le processus employé par le compilateur $C_1$ est très analogue à un processus de dérivation. Il s'agit là d'ailleurs d'une méthode très générale : Etant donnée une formule, la production cherchée dépend de l'opérateur principal, mais ne sera obtenue que lorsque les arguments de l'opérateur auront été traités, ce traitement faisant appel en général au processus lui-même.

La fonction centrale est nommée TYPO. Pour un opérateur donné, cette fonction passe le contrôle à un sous-programme spécifique, qui appelle lui-même TYPO, une ou plusieurs fois, de façon à traiter les arguments.

Lorsque les arguments ont été traités, le sous-programme attaché à l'opérateur, génère la description de l'image qui lui est associée et calcule les dimensions générales (W,D,H) du rectangle englobant, ceci, grâce aux informations que les appels à TYPO lui ont fournies.

Nous allons expliciter le cas du quotient :
Soit la liste (/<num><dénom>)

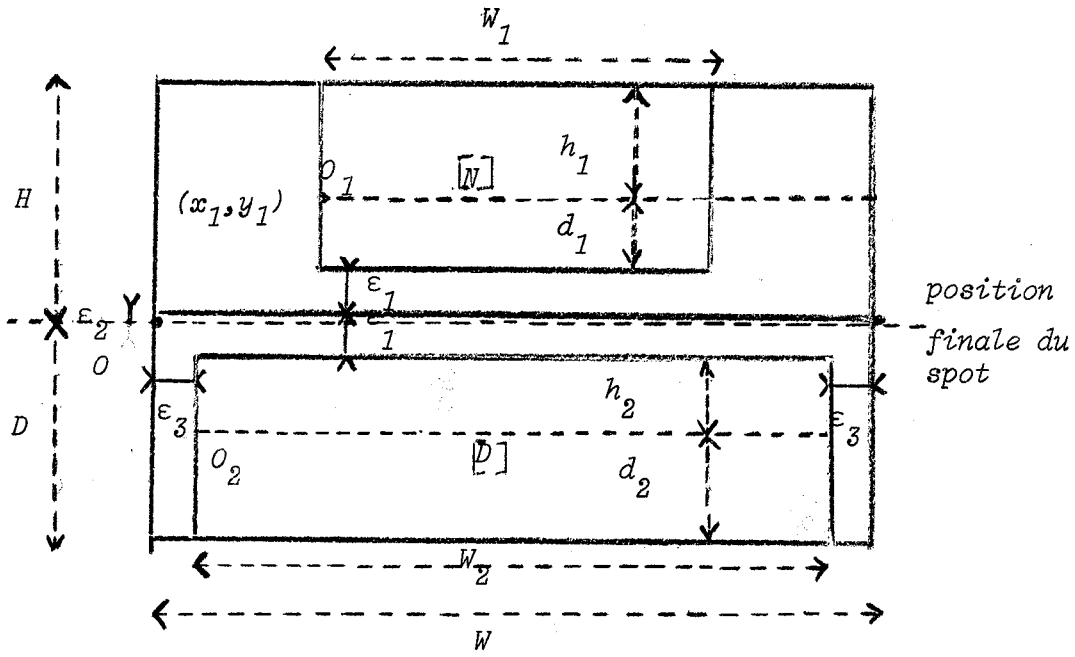
Lorsque cette liste est donnée au compilateur, la fonction auxiliaire TYPODVD prend le contrôle avec la liste (<num><dénom>) comme argument, et obtient les renseignements relatifs à <num>et à <dénom> en confiant ces deux listes l'une après l'autre et à la fonction générale TYPO.

Ces deux appels fournissent, sous forme de liste, les informations :

[N] , $w_1, d_1, h_1$ relatives au numérateur

[D] , $w_2, d_2, h_2$ relatives au dénominateur.

Les termes entre crochets, représentent la description en L.G. de la formule correspondante, relativement à l'origine locale du rectangle qui l'englobe. C'est à dire ici, $O_1'$ et $O_2'$ respectivement




Si l'on tient compte de quelques paramètres esthétiques, $\varepsilon_1, \varepsilon_2, \varepsilon_3$, le sous programme TYPDVD calcule :

(1) $W = \max(w_1, w_2) + 2\varepsilon_3$

$D = h_2 + d_2 + (\varepsilon_1 - \varepsilon_2)$

$H = h_1 + d_1 + (\varepsilon_1 + \varepsilon_2)$

(2) La position des origines locales $O_1$ et $O_2$ par rapport à O.

$$O_1 \begin{cases} x_1 = \frac{1}{2}\,[W-w_1] \\ y_1 = d_1+\varepsilon_1+\varepsilon_2 \end{cases} \qquad O_2 \begin{cases} x_2 = \frac{1}{2}\,[W-w_2] \\ y_2 = -\,[h_2+\varepsilon_1-\varepsilon_2] \end{cases}$$

et donne la description ci-dessous en L.G.

```
(COMPD (SKIP x1 y1)[N]
       (SKIP x2 y2)[D]
       (SKIP 0 ε1) (RVECTA W ε1)
       (SKIP W 0 )
)
```

dans laquelle les paramètres sont effectivement des nombres.

L'usage de COMPD est systématique puisque les déplacements sont mesurés par rapport à l'origine locale O.

Un traitement semblable est associé à chaque opérateur. Naturellement aux identificateurs et aux nombres, correspond un traitement qui ne nécessite aucun appel récursif ultérieur. Le composant produit est tout simplement :

(RTEXTG (QUOTE <liste des caractères>))

ou

(RTEXTP (QUOTE <liste des caractères>))

suivant que l'identificateur ou le nombre, doivent être en grands ou en petits caractères.

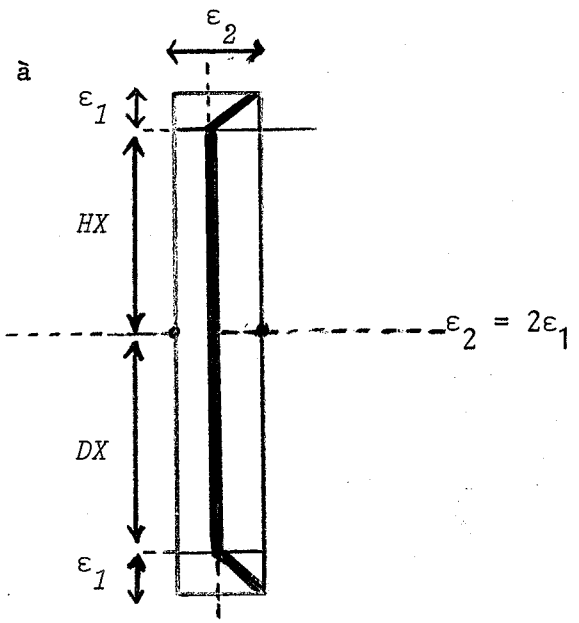
Un identificateur apparaîtra en petits caractères s'il appartient à une expression placée en indice, en exposant, ou comme borne d'une intégrale etc...

Quant aux symboles spéciaux $\Sigma$, $\surd$, $\int$, leur dessin est généré par un appel à une 'fonction image' à laquelle on donne comme paramètres effectifs des nombres dépendants des arguments liés à ces symboles: d'où une adaptation automatique de la taille de ces symboles à leurs arguments. Ce principe est adopté pour les parenthèses.

Il existe par exemple un image appelée PARENG (parenthèse gauche) dont la définition est :

```
(IMAGE(DX HX)
      (COMP(SKIP ε2 (MINUS(PLUS DX ε1)))
           (RVECTAD ε1 ε1)
           (RVECTAD 0 (PLUS HX DX))
           (RVECTAD ε1 ε1)
           (SKIP ε2 0)
      )
 )
```

qui correspond à



Supposons que le quotient étudié plus haut doive être mis entre parenthèses, le compilateur produira un composant qui contiendra :

....(PARENG $D_q$ $H_q$) (COMPD (.....)) (PAREND $D_q$ $H_q$) ....

$D_q$ et $H_q$ étant les dimensions du quotient seul (PAREND est la fonction image pour la parenthèse droite).

On notera, que nous utilisons encore là, l'idée générale de macro-génération.

### 4.2.3. Phase d'optimisation

Il est nécessaire de prévoir une phase d'optimisation pour ce que produit le compilateur TYPO. Par exemple, pour faire apparaître la chaîne 'A+B', un éventuel programmeur en L.G. écrirait

(RTEXTG (QUOTE A+B))

Par contre TYPO génère :

(RTEXTG(QUOTE A))(RTEXTG(QUOTE +))(RTEXTG (QUOTE B))

Un sous programme est donc chargé de balayer la liste générée par TYPO, pour regrouper les caractères qui doivent être écrits sur la même ligne.

Une autre sorte d'optimisation est nécessaire après l'appel du second compilateur, qui produit une liste d'ordres symboliques pour le 2250 : elle est relative à l'élimination des instructions successives de saut du spot : une séquence d'instructions symboliques telles que :

(GDV $x_o$ $y_o$ B) (GDV $x_1$ $y_1$ B) ... (GDV $x_n$ $y_n$ B)

doit être remplacée par la dernière d'entre elles.

Cette élimination est très facile à faire sur un programme symbolique ; c'est la raison pour laquelle nous avons décidé que la sortie de la fonction COMPIMAGE serait une liste d'ordres symboliques à assembler, plutôt qu'une liste de nombres binaires prêts à être chargés.

Néanmoins, cette conclusion n'a rien d'universel et ne doit sa validité qu'à la forme spécifique des commandes et des données du terminal 2250.

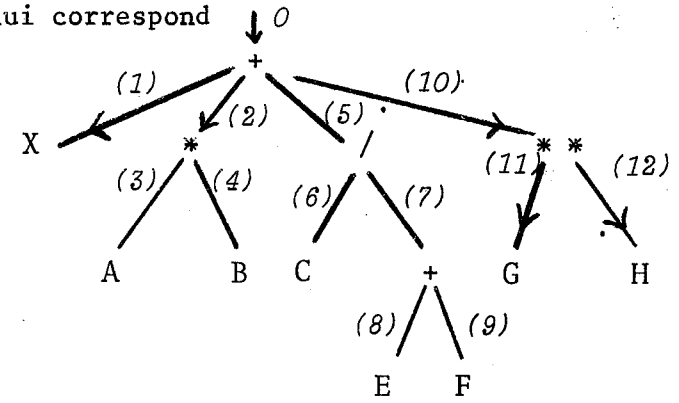
### 4.2.4. L'interaction avec le crayon optique

Puisque ce qui apparaît sur l'écran peut servir dans une certaines mesure à désigner des opérandes (par l'artifice du ?), il faut savoir lier le programme d'affichage qui commande le déplacement du spot et la représentation interne de la formule qui lui correspond. La méthode est tout à fait générale : elle consiste à jalonner le programme binaire d'informations, qui n'ont pas d'effets sur le déplacement du spot, mais qui peuvent être exploitées dans divers cas. On consultera à ce sujet la référence [52 ].

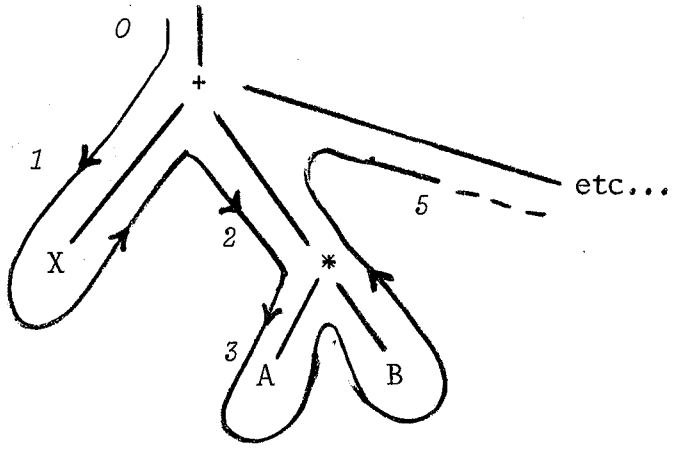
Pour préciser cela, nous allons montrer sur un exemple l'application de cette méthode au cas des formules.

Soit l'expression : $X + A * B + \frac{C}{E+F} + G^{H}$

et l'arbre qui lui correspond



La numérotation des branches correspondant à l'ordre dans lequel on les rencontre si on parcourt le graphe de la façon indiquée ci-dessous.




Cet ordre est d'ailleurs l'ordre habituel d'évaluation des arguments d'une expression.

Pour lier l'information logique à la visualisation, TYPO et ses fonctions auxiliaires doivent introduire dans le programme en L.G. le nombre associé à chaque branche. Le compilateur suivant, c'est à dire COMPIMAGE, transmettra cette information en générant certains ordres non opérationnels (GNOP4) dans le but de placer cette information au sein même du programme binaire.

Plus précisément, le programme binaire correspondant à une sous expression non atomique sera encadré par deux GNOP4 qui joueront le rôle de parenthèses (structure de bloc). Symboliquement le premier ordre d'une telle séquence sera : (GNOP4 0,n), n étant le numéro de la branche correspondante. La fin sera marquée par (GNOP4 1). L'imbrication des séquences ainsi parenthésées traduit la structure de la formule.

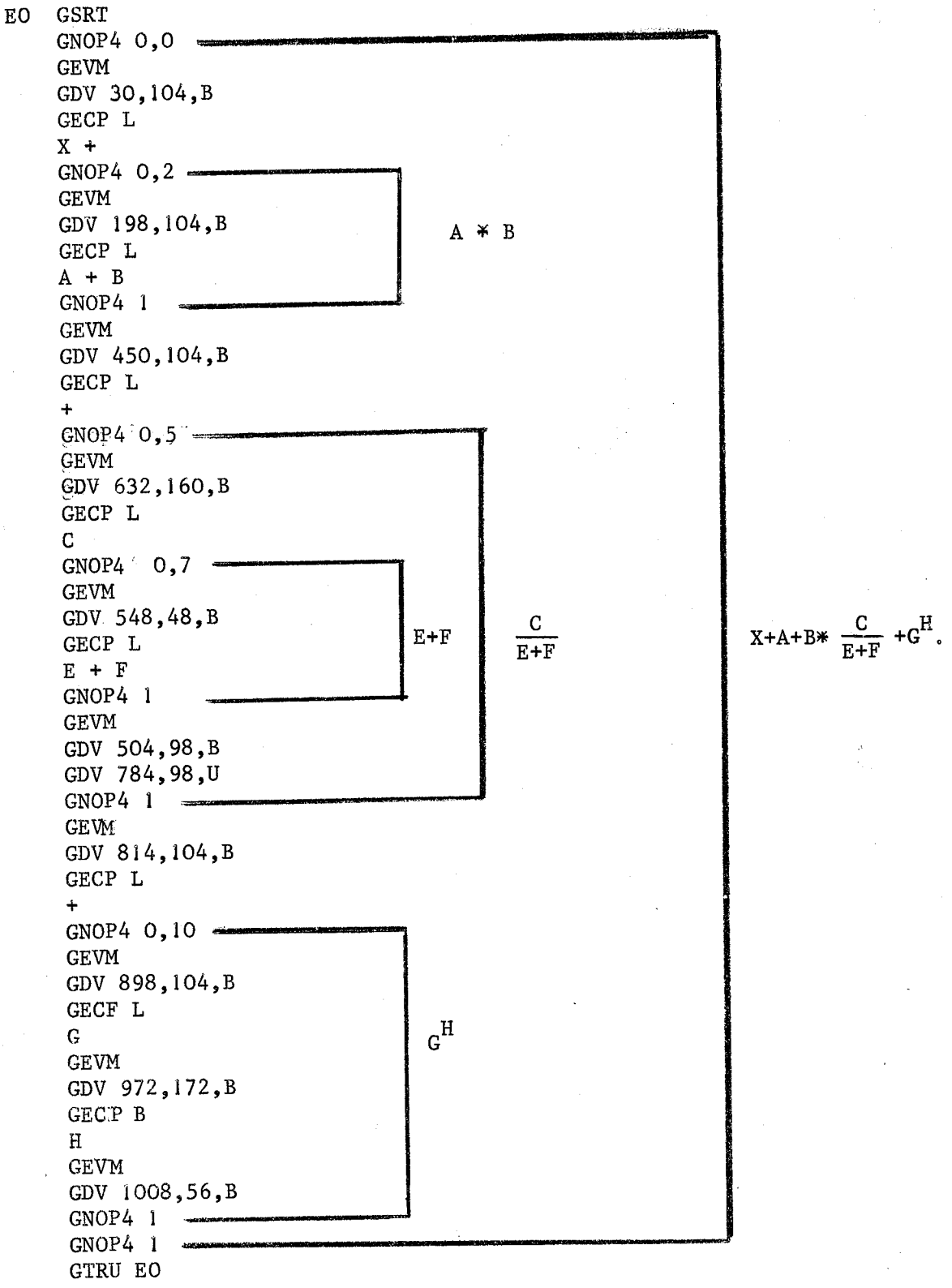
NOTE : *La restriction aux sous-expressions non atomiques est dûe à des raisons d'économie. Il s'agit d'une simple option que nous avons prise, en vue de raccourcir le programme généré.*

Pour notre exemple, TYPO génère la description suivante :

```
(COMPD 0 (RTEXTG (QUOTE (X +)))
   (COMPD 2 (RTEXTG (QUOTE (A * B))))
   (RTEXTG (QUOTE +))
   (COMPD 5 (SKIP 98 56) (RTEXTG (QUOTE C))
            (SKIP 14 - 56)
            (COMPD 7   (RTEXTG (QUOTE (E + F))))
            (SKIP 0 42) (RVECTA 280 42) (SKIP 280 0))
   (RTEXTG (QUOTE +))
   (COMPD 10 (RTEXTG (QUOTE G))
            (SKIPD 0 84)
            (RTEXTP (QUOTE H))
   (SKIP 140 0))) )))
```

Le compilateur COMPIMAGE est modifié pour accepter éventuellement un élément numérique placé derrière le symbole COMPD. Il génère alors le programme symbolique ci-dessous.

Pour des raisons de clarté, la liste de ce programme est donnée à la façon d'une liste d'assemblage.


EO GSRT
GNOP4 0,0
GEVM
GDV 30,104,B
GECP L
X +
GNOP4 0,2
GEVM
GDV 198,104,B
GECP L
A + B
GNOP4 1
A * B
GEVM
GDV 450,104,B
GECP L
+
GNOP4 0,5
GEVM
GDV 632,160,B
GECP L
C
GNOP4 0,7
GEVM
GDV 548,48,B
GECP L
E + F
GNOP4 1
E+F
GEVM
GDV 504,98,B
GDV 784,98,U
GNOP4 1
C/(E+F)
GEVM
GDV 814,104,B
GECP L
+
GNOP4 0,10
GEVM
GDV 898,104,B
GECF L
G
GEVM
GDV 972,172,B
GECP B
H
GEVM
GDV 1008,56,B
GNOP4 1
G^H
GNOP4 1
GTRU EO
X+A+B* C/(E+F) +G^H.

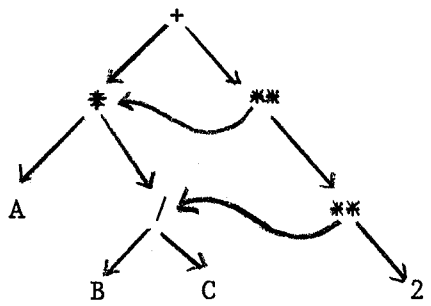
Il est important de noter que dans le cas où une formule contient une sous-expression qui apparait plusieurs fois (ce qui arrive en particulier pour les variables simples), le compilateur TYPO génère une description en L.G. pour chaque occurence de cette sous expression. Ceci est indépendant du fait que cette sous-expression peut n'être représentée qu'en un seul endroit en mémoire. D'ailleurs, ce dernier aspect n'entraine aucune difficulté pour la numérotation des sous-expressions.

Plus précisément, comme cette numérotation se fait d'une façon dynamique, lorsque TYPO génère la description en L.G. de la $k^{ème}$ occurence de la sous-expression, il place derrière le symbole COMPD le numéro associé à la branche correspondante de l'arborescence équivalente du point de vue sémantique.

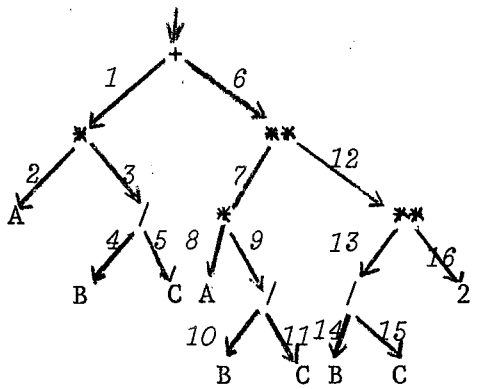
Ceci est général, quelque soit l'imbrication des sous-expressions Par exemple la formule :

$$A \times \frac{B}{C} + \left[A \times \frac{B}{C}\right]^{\left[\frac{B}{C}\right]^2}$$

dont une représentation peut être :




sera interpretée par TYPO comme :



L'exploitation de la liaison entre l'arbre et le programme génèré se fait après une détection autorisée au crayon optique. Une telle détection, permet grâce à un processus de gestion d'interruptions, de récupérer dans le calculateur principal une adresse qui est en relation étroite avec la portion de programme qu'exécutaient les circuits de la machine 2250, lorsque le spot lumineux est passé devant le crayon pointé sur une partie éclairée de l'écran.

Comme nous disposons en mémoire du calculateur principal une copie du programme d'affichage, il est facile d'explorer cette zone et de retrouver les deux commandes GNOP4 qui encadrent la séquence correspondant à ce qui a été "vu" par le crayon optique. Rappelons à ce propos que dans un ensemble d'informations binaires qui est un programme pour le 2250, les commandes sont identifiables par le nombre hexadécimal 2A. Le nombre n, placé dans la zone neutre de la commande GNOP4 est alors récupéré et, confié à un sous-programme qui permet de retrouver la branche appropriée.

NOTE : *Nous avons exposé la méthode, en supposant qu'il n'y avait qu'une seule formule affichée. Nous verrons plus loin comment, dans un programme qui affiche plusieurs formules, on retrouve celle qui est concernée par la détection*

Contrôle de la partie détectée Comme nous connaissons les adresses de début et de fin de la séquence, il est possible de modifier localement le programme d'affichage de façon à obtenir un changement d'aspect de l'écran permettant un contrôle par l'utilisateur. (variation de luminosité, clignotement).

Nous avons choisi d'éteindre tout l'écran, sauf la sous-expression sélectionnée. C'est la méthode qui semblait la plus simple et qui s'est avérée satisfaisante dans tous les cas.

Cas des parenthèses Des parenthèses sont parfois nécessaires dans l'affichage en deux dimensions. TYPO génère l'appel à la macro-instruction appropriée, en fonction de la priorité des opérateurs et de la nécessité de placer des parenthèses.

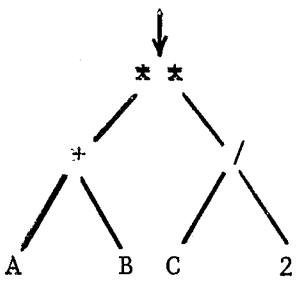
ex : A+B doit être parenthésé si nous affichons:

(* *(+ A B)2) mais pas si nous affichons :

(/ 2(+ A B))

Le nombre associé dans la génération en L.G. est celui qui correspond à la sous expression de niveau immédiatement supérieur à l'expression parenthésée. Une détection sur une parenthèse affichée sera donc interpretée en conséquence.

Exemple :




est affiché $[A + B]^{\frac{C}{2}}$

une détection au crayon optique sur une des parenthèses donnera l'expression toute entière. On peut remarquer que si une telle détection donnait A+B, il serait impossible de sélectionner l'expression complète, alors qu'avec cette convention, il est facile de sélectionner soit l'une des sous expression A+B ou $\frac{C}{2}$, soit l'expression générale.

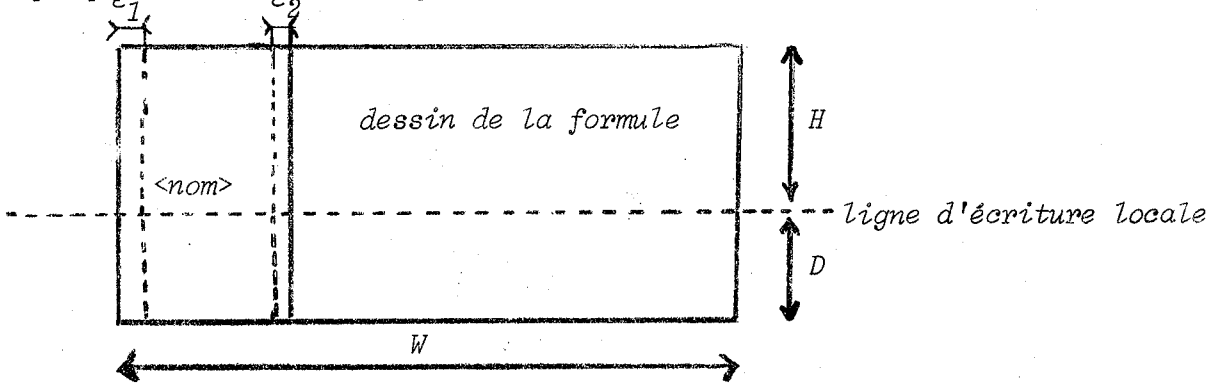
Cas des opérateurs hybrides Nous appelons opérateur hybride un opérateur dont le dessin en deux dimensions utilise certaines combinaisons de dessins d'opérateurs simples. Un exemple type est l'affichage de la notation de Leibnitz pour la différentiation :

En fait $\frac{\partial^{n+1}}{\partial x^n \partial y} F(x,y)$ a une correspondance lointaine avec (DRVP (FONCTION F X Y) X N Y 1). En particulier N+1 n'existe pas dans la liste originale. Par convention, une détection au crayon optique sur un symbole "emprunté" (ex : barre de fraction , ou exposant) donnera l'expression entière.

Détection sur des formules tronquées A la lumière de la méthode suivie pour la visualisation le lecteur comprendra aisément que, la numérotation étant "indépendante de la fenêtre", si une sous expression est tronquée, la séquence correspondant à ce qui est vu sur l'écran est néanmoins inscrite dans le programme binaire entre 2 GNOP4. Ainsi le moindre point lumineux à la limite de l'écran suffit pour retrouver la sous-expression entière correspondant à cette partie tronquée du dessin.

## 4.3. DISPOSITION DES FORMULES SUR L'ECRAN

Etant donné un nom de formule, c'est à dire une variable affectée, un appel à la commande IMAGE provoque la génération en langage graphique L.G. de la description du dessin suivant :



C'est à dire

```
(COMPD <nom> (SKIP ε1 0)
             (RTEXTG (QUOTE <nom>))
             (SKIPD ε2 0)
             <description en L.G. de la formule>
)
```

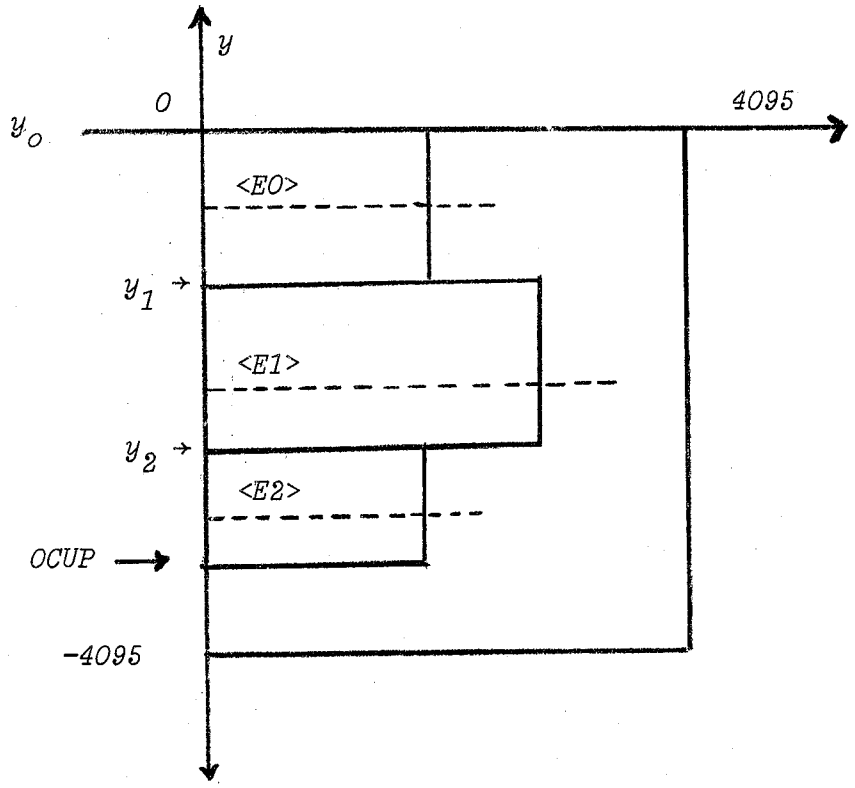
Cette description, ainsi que les dimensions du rectangle englobant, sont attachées au nom de la formule, c'est à dire deviennent des propriétés de l'atome correspondant. Ces propriétés restent valables tant qu'une nouvelle valeur n'est pas affectée à la variable concernée. L'atome <nom> est ajouté à une liste qui contient à tout instant l'ensemble des noms de formules dont on a demandé l'affichage.

Supposons que l'utilisateur ait demandé successivement

```
IMAGE (E0) ;
IMAGE (E1) ;
IMAGE (E2) ;
```

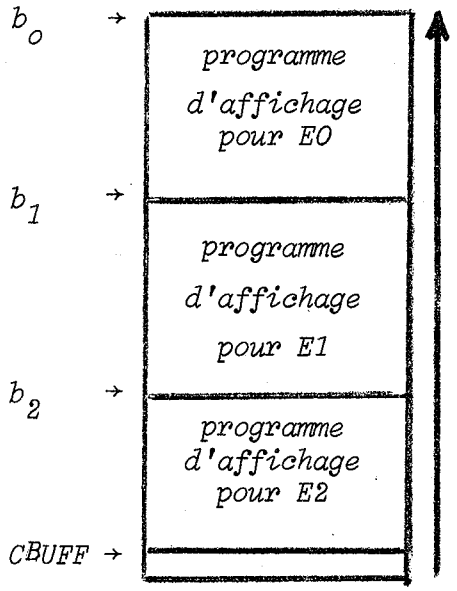
E0,E1 et E2 étant des noms de variables affectées.

Les images seront disposées ainsi :




Par convention, ces images sont placées "les unes sous les autres", dans la région du plan ($x \geq 0$, $y \leq 0$). La position initiale de la fenêtre, c'est à dire de l'écran, est indiquée sur la figure : les coordonnées du coin inférieur gauche sont : (0,-4095);c'est la commande de translation DEP, qui permet de déplacer la fenêtre dans le plan, et donc physiquement d'opérer une translation des images à travers l'écran.

Dans la mémoire interne du terminal 2250, nous avons :

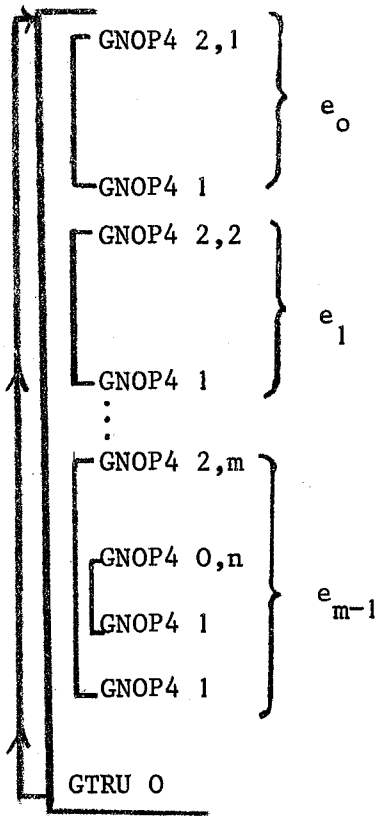



Le dernier ordre du programme est un ordre de transfert inconditionnel (GTRU), vers le début du programme provoquant un bouclage et par conséquent la stabilité de l'image affichée.

Les séquences de programme relatives à chaque formule sont séparées dans la mémoire par des ordres ineffectifs. En fait, chaque séquence, correspondant à une formule complète, est encadrée par deux parenthèses de programme

GNOP4 2,m et GNOP4 1

m étant le rang du nom de la formule dans la liste des noms des formules affichées.




Le sous programme qui permet, après une interruption de crayon lumineux, de retrouver par exemple la $n^{ième}$ sous expression de $E_{m-1}$ est aussi utilisé pour retrouver m. Ainsi, nous avons accès au nom de la formule par m, et à la sous expression désignée par n. On voit tout l'intérêt de la commande ineffective GNOP4.

A chaque $e_i$ sont attachées les informations $y_i$ et $b_i$ qui correspondent respectivement à la plus haute ordonnée de l'image $e_i$ dans le plan, et à l'adresse à laquelle le programme correspondant à $e_i$ a été chargé dans la mémoire du 2250.

Deux variables internes au système gardent la trace de la zone maximum occupée, d'une part dans le plan virtuel (OCUP), d'autre part dans la mémoire du 2250 (CBUF).

Etude des différents cas d'affichage :

a) affichage d'une nouvelle formule $E_i$.

Elle sera placée à la suite des autres, c'est à dire dans le plan virtuel, à partir de l'ordonnée OCUP et le programme correspondant sera

chargé à partir de l'adresse CBUF. Les deux variables d'occupation sont remises à jour.

En langage graphique, la description de l'image commence par :

(SKIP 0[valeur de OCUP -$H_i$]) ...

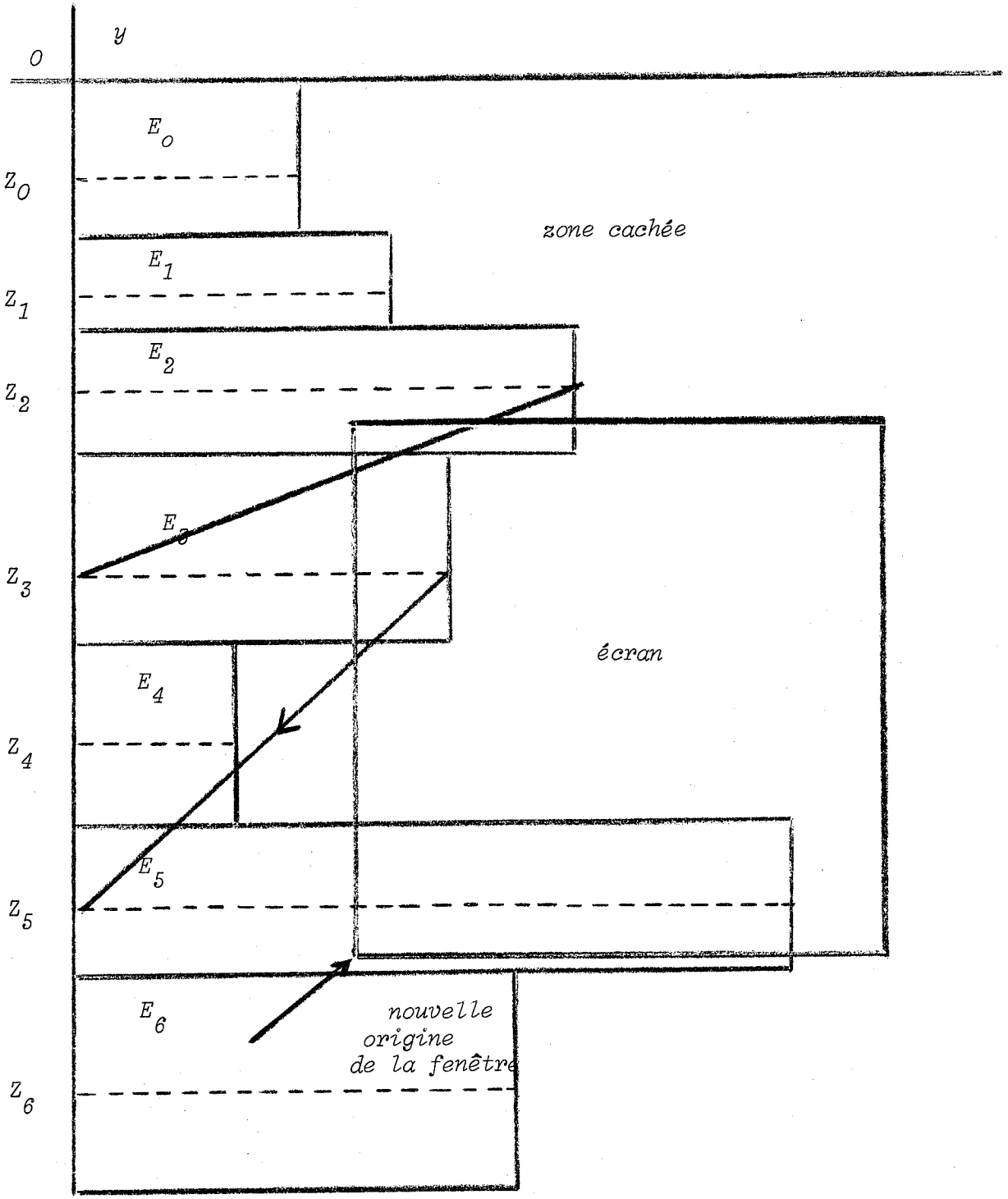
$H_i$ étant la hauteur de la formule, au dessus de la ligne d'écriture locale.

b) Translation de l'ensemble des images.

Il est nécessaire de générer un nouveau programme pour le 2250, puisque le balayage de l'écran par le spot lumineux sera différent. Compte tenu des descriptions en L.G. de chaque formule et des informations géométriques, on génére en L.G. une description globale de l'ensemble des formules qui sont présentement placées dans le plan. Cette description est confiée au compilateur COMPIMAGE qui tiendra compte des nouvelles coordonnées de la fenêtre : d'où la translation désirée. Mais avant de générer cette description, on peut à l'aide des informations géométriques savoir quelles sont les formules qui seront complètement cachées après cette translation. Seule les formules entièrement ou partiellement vues entreront en ligne de compte dans la description de l'image. On réalise ainsi dans certain cas une économie de temps non négligeable, puisque tout le processus de troncation des dessins est réalisé par programme en simulant le déplacement du spot.

NOTE : *Un cas particulier très fréquent de cette économie est réalisée lorsque l'écran est plein et que l'utilisateur fait appel à la commande IMAGE. La nouvelle image qu'il désire afficher est compilée en L.G., son nom est ajouté à la liste courante des images, mais COMPIMAGE n'est pas appelé. Le dessin effectif de la formule ne sera vu que si l'utilisateur opère une translation adéquate de la fenetre.*

Montrons sur un exemple ce qui se passe lors d'une translation




Cas d'une translation

La description confiée à COMPIMAGE, sera dans l'exemple précédent :

(COMP (SKIP 0 $Z_2$)
[E2]
(SKIP 0 $Z_3$)
[E3]
(SKIP 0 $Z_5$
[E5]
)

les termes entre crochets étant les descriptions en L.G. des dessins des formules $E_i$.

c) Re-affectation ou effacement

Dans chacun de ces deux cas, il y a mise à jour de la disposition globale, à partir de la formule que l'on modifie (commandes d'affectation) ou qui disparait (commandes RAYER ou EFFACER).

Considérons la liste des formules affichées

$$(E_1, E_2, E_3, \ldots, E_i, \ldots, E_n)$$

Si la formule $E_k$ est modifiée ou effacée, rien n'est modifié dans le plan virtuel et éventuellement sur l'écran pour les formules $E_i$ dont le rang i dans la liste précédente est tel que :

$$i < k$$

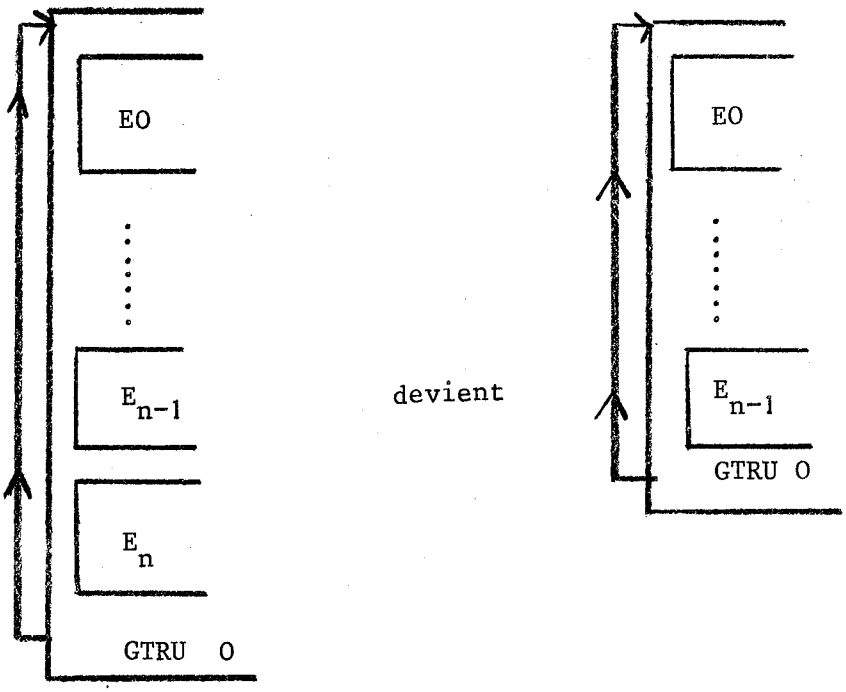
On applique alors l'algorithme décrit en b) à l'ensemble des formules dont le rang j est tel que :

$$j \geq k$$

Le programme correspondant sera chargé dans le 2250, à partir de l'adresse $b_k$.

Suivant les cas, certaines formules peuvent apparaître sur l'écran et d'autres disparaitre : ceci est du au fait que la mise à jour de l'image est faite dans le plan virtuel. Dans notre exemple, si l'on efface $E_o$, l'image subit une translation vers le haut et $E_6$ peut éventuellement apparaitre*; elle apparaitra à coup sûr si l'on efface $E_5$. Si $E_4$ est effacé, le bas de l'écran subira une translation vers le haut, car $E_4$ est intervenue par ses dimensions dans la description de la précédente image.

Remarque sur l'effacement Dans un but d'efficacité il est recommandé si c'est possible, d'effacer les images dans l'ordre inverse de leur affichage. En effet, du point de vue de la programmation, il suffit de modifier l'ordre de saut dans la mémoire du 2250, comme indiqué sur notre schéma.




* Il faut que la partie non vue de $E_5$ soit inférieure à la hauteur totale de $E_o$.

## CONCLUSION

Le nombre croissant de publications et les réunions de plus en plus fréquentes sur le calcul formel en machine, montrent l'intérêt que ce genre de travaux suscite chez tous ceux qui pour gagner du temps, ou s'éviter un travail fastidieux aimeraient automatiser certains calculs algébriques et s'aider d'un calculateur pour chercher rapidement des solutions.

C'est un encouragement pour l'avenir et le meilleure sort que puisse subir ce document est d'être rapidement dépassé par les thèses de ceux qu'il aura peut-être inspirés.

# BIBLIOGRAPHIE


SUR LE TRAITEMENT DE LISTE EN GENERAL

[1] Newell, Allen : *Information Processing Language (IPL V) Prentice Hall, Inc. Englewoods Cliffs N.J.*

[2] Weizenbaum : *Symmetric List Processor CACM vol. 6, n° 9, (Sept. 1966)*

[3] Perlis, A.J. : *Symbol Manipulation by Threaded Lists CACM vol. 3(1960)*

[4] Perlis, Iturriagea : *Formula Algol, Carnegie Institute of Technology Pittsburgh (1966)*

[5] Cohen j. et Nguyen Huu Dung : *traitement des listes en Algol revue AFIRO (CHIFFRES) n° 8 (1965)*

[6] Jenkins D.P. : *List Programming - Introduction to system Programming by P. WEGNER, Academic Press (1964).*

[7] SUR LISP ET DIFFERENTES QUESTIONS THEORIQUES

[8] Mac Carthy John : *Recursive Functions of Symbolic Expressions and their Computation by Machines, CACM vol 3 1960*

[9] Church A : *The calculi of Lambda conversion, Princeton University Press n° 6 (1941)*

[10] Davis : *Computability and Unsolvability Mc Graw Hill. New York (1958)*

[11] Rogers H. : *Theory of Recursive Functions and Effective Computability, Mc Graw Hill. New York (1967).*

[12] Minsky : *Computation on finite and infinite machines Prentice Hall Series in automatic computation, Englewoods Cliffs N.J. (1967)*

[13] Mac Carthy John : *A basis for a Mathematical Theory of computation, in Computer Programming and Formal system. Braffort et Hirshberg. North Holland Company.*

[14] Mac Carthy et al : *LISP 1.5 Programmer's Manual, MIT Press Cambridge, Mass. (1962).*

[15] Berkeley and Bobrow : *The programming Language LISP : its operation and Applications. Information International Inc. Cambridge Mass. Mass. (1964).*

[16] Ribbens : *Programmation non Numerique : Lisp 1.5, Dunod (1969)*

[17] Weissman C. : *Lisp 1.5. Primer. Dickenson Publishing Company, Belmont California.*

SUR DES APPLICATIONS DE LISP

[18] Slagle James R. : *A heuristic Program that solves Integration Problem in Freshman Calculus : Symbolic Automatic Integrator (SAINT), Rep 5 G-0001, M.I.T. Lincoln Lab., Lexington Mass. (May 1961).*

[19] Moses J. : *Symbolic Integration. Thèse PH.D. M.I.T. 1967*

[20] Evans Thomas G. : *A program for the solution of a class of geometric Analogy Intelligence Test Questions. Thèse PH.D. M.I.T. May 1963.*

[21] Woolridge D. Jr : *An Algebraic Simplify Program in LISP. Stanford Artificial Intelligence Nem. N° 11 (1963).*

[22] Griffiths M., Petrick S. : *On the relative Efficiencies of Context-Free grammar Recognizers CACM Vol 8. May 1965.*

[23] Green C., Raphael B. : *Research on Intelligent Questions-Answering System Standford Research Institute, Mento Park, California.*

[24] Siret Y. : *Obtention du Discriminant d'une équation par la méthode de J.A. Serret. Revue Afiro n° 9. 1968.*

SUR DES SYSTEMES CONVERSATIONNELLS ECRITS EN LISP

[25] Martin William A. : *Symbolic Mathematical Laboratory Thèse PH.D. M.I.T. 1967*

[26] Engleman Carl : *Mathlab. A program for on line Machine assistance in symbolic computations. Proc. FJCC. Nov 65.*

[27] Fenichel R.R. : *An on line System for algebraic Manipulation (FAMOUS) PH.D. Harvard. Cambridge, Mass (1966).*

[28] Hearn A. : *REDUCE. An on-line algebraic manipulation system. Stanford Artifical Int. Memo. (1968).*

[29] Korswold K. : *An one line algebraic Simplify Program. Stanford Art. Intell. Memo. N° 37 (Nov. 1965).*

[30] Griesmer J.G., Blair F.W., Jenks R.D. : *An iteractive facility for symbolic Mathematics, IBM Watson Research. report RC 2776 (1970).*

SUR LES METHODES DE VISUALISATION

[31] Krakauer L.J. : *Syntax and Display of Printed Format Mathematical Formulas. Thèse M.S. MIT (1964). (Voir aussi Martin 25 ).*

[32] Millen J.K. : CHARYBDIS : *a LISP Program to display mathematical expression on typewriter-like dévices MITRE corporation MTP. 63 Bedford Mass.*

[33] Anderson : *Hand Printed Symbols Syntax Recognition. Thèse PHD Harvard 1968*

[34] Sipala P. : *Formatting an display of algol expressions. Computer Journal 1969 Page 365.*

[35] Wells, Mark B. MADCAP : *A Scientific Compiler for a displayed Formula Text Book language. CACM Janvier 1961.*

OUVRAGES D'INTERET GENERAL ET ARTICLES DIVERS

[36] Knuth : *The Art of Computer Programming Vol 1 et 2. Addison-Wesley 1968 et 1969.*

[37] Sammet Jean : *Survey of Formula Manipulation. CACM August 1968*

[38] Sammet Jean : *Programming Languages. Prentice-Hall. Inc, Englewood Cliffs. N.Y. 1969*

[39] Van de Riet : *Formula Manipulation in Algol 60 (2 tomes) Mathematisch Centrum. Amsterdam 1968.*

[40] Collins, G.E. : *The SAC-1 List Processing System. University of Wisconsin Computing Center Report, September, 1967.*

[41] Collins, G.E. : *and Pinkert, James R. The Revised SAC-1 Integer Arithmetic System. University of Wisconsin Computing Center, Technical Reference NO. 9, November 1968*

[42] Collins, G.E. : *The SAC-1 Polynomial System, University of Wisconsin Computing Center, Technical Reference n° 2, January, 1968.*

[43] Collins, G.E. : *The SAC-1 Rational Function System. University of Wisconsin Computing Center, Technical Reference n° 8, July, 1968.*

[44] Collins, G.E., et al. *The SAC-1 Modular Arithmetic System. University of Wisconsin Computing Center, Technical Reference n° 10, June, 1969.*

[45] Martinet J. et Siret Y. : *Obtention automatique des équations de Runge et Kutta dans Computers in mathematical Research édité par Churchhouse et Hertz. North Holland Comp. 1968.*

[46] Tobey : *Eliminating Monotonous Mathematics With Formac. CACM. Aug. 1966.*

[47] Bolliet - Gastinel : *Algol - Edition Hermann 1964.*

[48] Randel et Russel : *Algol 60 Implantation. Academic Press 1964*

[49] Bolliet L. : *Les systèmes Conversationnels. Colloque Grenoble Décembre 1968 - DUNOD.*

OUVRAGES TECHNIQUES

[50] PL/1 Language Specifications : *IBM Form. C28 - 6571.*

[51] IBM Display 2250 Mod 1 : *Form A 27-2701*

[52] Lucas : *Techniques de programmation et utilisation en mode conversationnel des terminaux graphiques thèse 3ème cycle GRENOBLE Juin 1968.*

OUVRAGES MATHEMATIQUES

[53] Dubreuil : *Algèbre. Gauthier Villars (1963)*

[54] Salmon : *Algèbre Supérieure. Gauthier Villars (1890)*

[55] Humbert : *Cours d'analyse à l'Ecole Polytechnique. Gauthier Villars (1911)*

[56] Berge : *Théorie des graphes et ses applications. DUNOD (1959)*

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

Vu, et permis d'imprimer

Le Recteur de l'Académie de GRENOBLE