UNIVERSITÉ PARIS XI
U.E.R. MATHÉMATIQUE
91405 ORSAY FRANCE

N^os 156 - 75.45

Georges KIREMITDJIAN

Vérification interactive de démonstrations mathématiques

Note ECSTASM N° 13
Publication Mathématique d'Orsay

TABLE DES MATIERES

Vérification interactive de démonstrations mathématiques.

Cette étude est une contribution au projet AUTOMATH dirigé par Mr le professeur De Bruijn de l'université d'Eindhoven.

L'étude mathématique de la notion de démonstration mathématique permet de situer les systèmes AUTOMATH par rapport à la théorie de la démonstration classique.
La formalisation des démonstrations est faite dans un langage mathématique qui est un système de lambda-calcul lambda-typé. L'étude de la typification au sens de la correspondance de Howard est entreprise. Ses théories sont appliquées pour la spécification du langage LAMBDA. Un vérificateur de livres mathématique écrits en langage LAMBDA a été mis en place en utilisant le système SISGOF.

I / La démonstration assistée par ordinateur.

1 : Objectifs

Quelle que soit l'école à laquelle il appartient (classique, intuitionniste,...), quel que soit le domaine de ses travaux (analyse, logique, géometrie,...) le mathématicien effectue une grande quantité de vérifications minutieuses, répétitives, qui bien qu'indispensables pour conserver une certaine rigueur, constituent la partie la moins intéressante de son travail.

Le premier objectif est donc de faire effectuer ces vérifications par une machine. Pour cela il faut définir les démonstrations de façon à ce qu'il n'y ait plus d'ambiguités possibles. Cet effort d'explicitation des démonstrations, qu'il soit accompli par le mathématicien ou par la machine, apporte une meilleure compréhension des mathématiques elles-mêmes. En particulier, l'étude des démonstrations en tant que telles, leur mise sous une forme standard, les simplifie et fait apparaître les hypothèses réelles dont elles dépendent, alors que les hypothèses initiales pouvaient être plus fortes.

2 : Nécessité de définir un langage mathématique.

Les travaux de l'école formaliste d'Hilbert ont bien mis en évidence la nécessité d'une formalisation des théories mathématiques, c'est à dire la définition explicite du langage

utilisé par de telles théories. On a donc été amené à distinguer nettement le langage de la théorie et le langage utilisé pour parler de la théorie (métalangage).

D'un autre côté, l'informatique fournit le moyen de traiter des langages formalisés. Alors, bien que la définition d'un langage mathématique formel puisse se faire indépendamment de tout support physique de traitement, une définition de niveau assez bas pour être traitée automatiquement par une machine, présente un certain intérêt.

En conclusion, la définition d'un langage mathématique, bien que présentant un intérêt mathématique en tant que tel, se justifie d'autant plus que le handicap de la lourdeur d'un tel système sera supporté par la machine.

3 : Quelles doivent être les caractéristiques d'un tel langage ?

Il devra être indépendant de tout système logique. Mieux, il servira à étudier différents systèmes logiques et à les comparer, par exemple, les système classique, intuitionniste et modal...

Il faut aussi que l'on puisse y formaliser toutes les théories mathématiques, pour qu'il mérite son nom de langage mathématique.

De plus, il est souhaitable qu'il soit lui même un objet mathématique, c'est à dire qu'il soit bien défini de façon à pouvoir être étudié en tant que tel, ce qui permettra de connaître très exactement ses propriétés.

D'un point de vue pragmatique, il doit être linguistiquement simple pour être analysé par les méthodes informatiques usuelles.

Un dernier point peut être soulevé, son interaction avec l'utilisateur. Ce point est en réalité secondaire, car on peut envisager des interfaces adaptées à chaque profil d'utilisateur. Par exemple, un langage de niveau plus élevé pour les besoins de l'algébriste, un autre pour le topologue... Au niveau de ces langages spécialisés on introduira des facilités pour l'apprentissage et l'utilisation du système.

4 : Pourquoi la vérification et non pas la construction de démonstrations ?

Le problème de la démonstration automatique de théorèmes mathématiques peut prendre deux formes.

La première consiste à définir un système formel et à faire générer par la machine "tous" les théorèmes. A l'utilisateur de voir quels sont ceux qui sont "intéressants" .

Une deuxième approche est de fixer un théorème à démontrer. Théoriquement, ce problème dans sa généralité n'est pas décidable. Si l'on se place dans un système formel décidable aussi simple que le calcul des propositions, des difficultés combinatoires apparaissent très vite. Le problème est donc alors de définir les méthodes heuristiques qui permettront de le résoudre effectivement.

Une deuxième critique à la démonstration automatique, d'ordre philosophique, est que le mathématicien veut comprendre la démonstration du théorème. Or, la démonstration la plus simple pour la machine, ne l'est pas forcément pour le mathématicien du fait d'une longueur et d'une complexité combinatoire trop grandes.

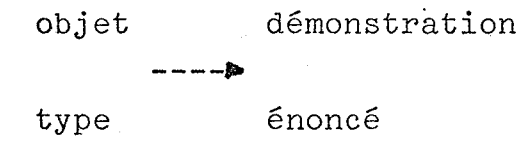
La compréhension de la démonstration est aussi importante que le théorème lui même. Ces problèmes seront peut être résolus un jour. Cependant la vérification automatique des démonstrations peut déjà apporter une aide pratique au mathématicien en allégeant sa tâche et en garantissant, par un contrôle strict, la rigueur de ses démonstrations. De plus, ce travail peut amener à une meilleur compréhension des mécanismes de démonstration en mathématique et par là même apporter une aide appréciable au problème de la démonstration automatique.

5 : Présentation de la méthode suivie

La théorie de la démonstration créée par Hilbert est étudiée au chapitre II, on y montre pourquoi les systèmes déductifs de Gentzen sont mieux adaptés à l'étude des preuves. Pour Hilbert la théorie de la démonstration consistait surtout à énoncer des résultats sur ce qui était démontrable dans un système formel. Gentzen a aussi étudié la façon dont s'effectuent les démonstrations.

Une première formalisation, P.A.L montre comment

on peut utiliser la correspondance de Howard :



Ensuite, une analyse plus fine de la notion de fonction, à l'aide du lambda-calcul, nous conduit aux systèmes du type AUTOMATH.

Le chapitre III est consacré à un rappel du lambda-calcul. La relation entre le système de Gentzen G et le lambda-calcul typé D est étudiée. Le système D est une formalisation des démonstrations dans G.

Une formalisation des démonstrations dans le lambda-calcul lui même conduit à un lambda-calcul lambda-typé.

C'est le système de Nederpelt étudié au chapitre IV.

Dans ce chapitre, l'étude de la typification est une contribution originale. Elle permet de faire le lien avec les différents systèmes AUTOMATH qui sont étudiés au chapitre V.

Ce chapitre V contient aussi la description des systèmes effectivements réalisés : AUT-SL et LAMBDA.

Cette réalisation utilise le système SISGOF et la partie gestion du livre et des lignes de LIMA-PAL.

II / LA THEORIE DE LA DEMONSTRATION.

1 : Hilbert et les systèmes formels.

C'est grâce aux travaux d'Hilbert que l'utilisation de systèmes formels s'est généralisée en mathématique. Cette approche permet de distinguer le langage du méta-langage. Dans le langage sont écrits les théorèmes et dans le métalangage est décrite la façon d'obtenir ces théorèmes. C'est pourquoi l'étude métalinguistique peut s'appeler "théorie de la démonstration".

Exemple de système formel d'Hilbert.

Soit H = (A,F,Z,R) un système formel ainsi défini :

. A = (V,L,U) l'alphabet composé de

V ensemble des variables p,q,r,s,t....

L ensemble des connecteurs logiques $\rightarrow$

U ensemble des signes auxiliaires ()

. $F \subset A^*$ ensemble des formules ou ebf (expressions bien formées)

1) $V \subset F$

2) P et $Q \in F$ alors $P \rightarrow Q \in F$

. $Z \subset F$ ensemble des axiomes

S1 $P \rightarrow (Q \rightarrow P)$

S2 $(P \rightarrow (Q \rightarrow M)) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow M))$

, R ensemble des règles de déductions

R1 règle de substitution

à P,Q,M on peut substituer n'importe quelle ebf.

R2 modus ponens

si P et Q∈F, P et P→Q alors Q.

Démonstrations dans H.

La démonstration d'un théorème dans H est une suite de formules dont la dernière est le théorème.
Toute formule d'une démonstration est, soit un axiome, soit déduite des formules précédentes par application des règles de déduction.

ex: démonstration de p→p

1	p→((q→p)→p)	R1(p~P,p→q~Q)S1
2	p→(q→p)	R1(p~P,q~Q)S1
3	(p→((q→p)→p)→((p→(q→p))→(p→p))	R1(p~P,q→p~Q,p~M)S2
4	(p→(q→p))→(p→p)	R2(1,3)
5	p→p	R2(2,4)

Remarques :

1) toutes les ebf utilisées dans la démonstration sont des théorèmes de H.
2) le choix des axiomes et des règles à utiliser n'est pas évident.

L'étude métamathématique d'Hilbert visait surtout à énoncer des propriétés sur les théorèmes démontrables dans un système formel.

Par exemple :

- la consistance (non-contradiction)
- la décidabilité
- la complétude
- ...

2 : Système de déduction naturelle de Gentzen.

Nous allons définir le système équivalent à celui présenté au paragraphe précédent.

G = (A,F,R) où A et F ont la même définition que pour H.

Dans ce système il n'y a pas d'axiomes, mais des règles d'introduction et d'élimination des connecteurs logiques.

Règles

$A \vdash A$

règle de répétition

1	A	hypothèse
n	A	répét(1)

Sous l'hypothèse A on peut utiliser A dans la démonstration.

règle de réitération

1	A	hypothèse
2	B	hypothèse
n	A	réit(1)

Sous l'hypothèse B on peut utiliser A, si B est déjà sous l'hypothèse A.

"A",B⊢A→B règle →I

1	A	hypothèse	
n	B	démonstration	
m	A→B	→I(1,n)	

Sous l'hypothèse A, si l'on démontre B, on peut déduire A→B en s'affranchissant de l'hypothèse A.

A,A→B⊢B règle →E (modus-ponens)

1	A→B	hypothèse	
2	A	hypothèse	
3	B	→E(1,2)	

Sous les hypothèses A et A→B, on peut déduire B.

Démonstrations dans G.

La démonstration d'un théorème est une suite de formules dont la dernière est le théorème. Une formule est, soit une hypothèse, soit l'application d'une règle aux formules précédentes. Toute formule n'est pas nécessairement un théorème, mais plutôt une assertion du style suivant :

"sous telles hypothèses, on peut déduire telle formule"

Nous avons vu que la règle →I permet d'éliminer une hypothèse. Un théorème est une formule déduite d'un ensemble vide d'hypothèses.

ex : démonstration de p→p.

1	p	hypothèse
2	p	répét(1)
3	p→p	→I(1,2)

p⇒p ne dépend d'aucune hypothèse, puisque l'hyopthèse 1 est levée lors de l'application en 3 de la règle ⇒I.

Une première remarque s'impose, cette démonstration est beaucoup plus simple et plus naturelle que celle effectuée dans le formalisme d'Hilbert.

Alors que la première démonstration est linéaire, celle-ci est structurée par les barres de contexte.

Donnons un autre exemple.

1	p	hypothèse
2	p⇒q	hypothèse
3	p	réit(1)
4	q	⇒E(2,3)
5	(p⇒q)⇒q	⇒I(2,4)
6	p⇒((p⇒q)⇒q)	⇒I(1,5)

Pour démontrer le même résultat dans H, il faut utiliser la démonstration de p⇒p, et démontrer la règle dérivée suivante :

Si A⇒B et B⇒C alors A⇒C (transitivité).

Alors la démonstration est la suivante :

1	p⇒((q⇒p)⇒p)	} démonstration précédente
	.	
	.	
	.	
5	p⇒p	
6	(p⇒q)⇒(p⇒q)	R1(p∼q⇒p)5
7	((p⇒q)⇒(p⇒q))⇒(((p⇒q)⇒p)⇒((p⇒q)⇒q))	R1(p⇒q∼P,p∼Q,q∼M)S2
8	((p⇒q)⇒p)⇒((p⇒q)⇒q)	R2(6,7)
9	p⇒((p⇒q)⇒q)	transitivité(1,8)

Dans la pratique on utilise des règles dérivées pour pouvoir effectuer des démonstrations dans H. Les règles primitives de G sont directement utilisables et permettent des démonstrations plus directes.

3 : Comparaison des systèmes du type Gentzen et Hilbert.

1 - Un système de Gentzen emploie seulement la dérivation, il ne dispose pas d'axiomes.

2 - Dans un système de Hilbert, les axiomes sont les points de départ des déductions (prémisses), les théorèmes sont les conclusions de l'application des règles aux prémisses. Dans un système de déduction naturelle de Gentzen, les prémisses sont des hypothèses, et les conclusions déduites des règles sont des assertions. Une hypothèse peut être n'importe quelle formule, elle peut être vraie, fausse ou contingente.

3 - Chaque formule d'une déduction valide dans le système d'Hilbert est elle-même un théorème, ce n'est pas le cas chez Gentzen. En fait, dans la plupart des dérivations seule la dernière étape donne un théorème.

Donc dans un système de Gentzen, la distinction est faite entre assertion et théorème. Une assertion est un théorème sous réserve de la satisfaction des hypothèses. Noté $H \vdash TH$.

Un théorème est une assertion avec un ensemble d'hypothèses vide. Noté $\vdash TH$.

4 - Dans le système d'Hilbert on dispose de beaucoup d'axiomes et de deux règles. Dans le système de Gentzen il n'y a pas d'axiomes, mais ceux-ci sont remplacés par de nombreuses règles, deux par connecteur logique, qui peuvent être considérées comme sa définition. L'une permet de l'introduire, l'autre de l'éliminer.

Dans deux systèmes équivalents, l'un de Gentzen et l'autre de Hilbert, les théorèmes sont les mêmes. En particulier, dans le système de Gentzen, on peut démontrer les axiomes du système de Hilbert.

5 - Les démonstrations sont plus simples et plus naturelles dans un système de Gentzen que dans un système de Hilbert. Cela est dû à l'extension de la notion de déduction (prémisses vers conclusion et non pas théorèmes vers théorème) et à la structuration de la démonstration.

4 : Forme normale d'une démonstration.

Dans les systèmes de Gentzen, les démonstrations sont des suites de déductions élémentaires.
Ces déductions élémentaires portent sur un seul connecteur logique, elles sont de deux sortes :

introduction et élimination.

Ce couple de règles, inverses l'une de l'autre, peut être considéré comme la définition du connecteur.

La décomposition d'une démonstration en déductions élémentaires est donc l'analyse la plus fine que l'on puisse faire de celle-ci. Ainsi, il est possible d'étudier le développement d'une preuve et de supprimer les étapes inutiles. Intuitivement, on peut dire qu'une démonstration est sous forme normale si l'on a supprimé toutes les étapes inutiles. Gentzen a été le premier à démontrer dans son système "sequents calculi", l'existance d'une forme normale pour toute démonstration.

Nous reviendrons sur cette question des formes normales de démonstrations lors de l'étude du formalisme utilisé pour le langage mathématique. Il est équivalent au formalisme de Gentzen, mais ces propriétés y sont plus évidentes.

5 : Premier essai de formalisation d'un langage mathématique : P.A.L.

Le système de déduction naturelle a été appliqué à la formalisation d'une partie de la logique. Mais si l'on considère la méthode utilisée, elle peut être appliquée plus généralement. Par exemple en arithmétique, au lieu d'introduire les connecteurs logiques, on introduit la fonction successeur, les opérateurs ...

Dans notre système il faut définir les connecteurs logiques, les axiomes, les opérateurs que l'on veut utiliser. Ce sont les constantes, elles peuvent avoir plusieurs arguments. On peut aussi y définir des

variables qui correspondent aux hypothèses.

Le format des lignes est similaire à celui donné lors des preuves dans G.

indicateur de contexte	indentificateur de la ligne	démonstration	énoncé.

La seule règle étant la substitution, ou plutôt le changement de contexte. Une constante C, à n paramètres, définie dans un contexte (p,q,r...s,t) peut être utilisée dans un autre contexte (a,b,c...d,e,f)

$$D = \ldots C(\underbrace{c,a,..e,f}_{n})\ldots.$$

De plus, l'analogie suivante, appelée "correspondance de Howard" permet si l'on considère la typification des objets, d'obtenir le modus ponens sur les types. Les objets sont alors les démonstrations.

$x \in X$	x a pour type X
$f \in X \rightarrow Y$	f a pour type l'application de X dans Y
$f(x) \in Y$	f(x) a donc pour type Y.

En fait, dans le format de la ligne, la démonstration est un objet et l'énoncé est son type. Il faut, bien entendu, imposer une condition d'applicabilité lors du changement de contexte des constantes. Les paramètres correspondants, lors de leur définition et de leur utilisation, devront avoir le même type.

La vérification qu'une démonstration donne bien le théorème indiqué, consiste à constater que le type de l'objet démonstration est bien le type énoncé.

De cette façon on obtient le système P.A.L de De Bruijn.

6 : P.A.L.

La description de P.A.L est donnée dans LIMA PAL (note ESCTASM n°8).

Les pages qui suivent en sont un extrait.

La version décrite dans cette note, n'utilisait pas SISGOF. La différence entre cette première version et la version actuelle se situe au niveau expression.

La validité et le type d'une expression sont fournis par un vérificateur généré par SISGOF. En annexe 1, se trouve la description de la syntaxe et de la sémantique du langage des expressions.

Structure - Définitions.

Un texte écrit en P.A.L est appelé LIVRE. La structure physique du livre est une séquence de LIGNES. La structure logique est une arborescence.

La structure de la ligne est la suivante :

IN ID DEF CAT

ID : identificateur qui est le nom donné à l'expression introduite par la ligne

DEF : la définition de cette expression

CAT : la catégorie qui indique le type de l'expression

IN : est un indicateur de contexte.

Chaque mot de P.A.L est représenté par son identificateur. Il y a deux sortes de mots : les variables et les constantes. Ces mots sont introduits dans le texte P.A.L suivant les besoins de la théorie dont il est la formalisation. Les variable sont définies par le symbole 'EB' en zone définition. Les constantes sont

- soit des éléments primitifs définis par 'PN' en zone définition
- soit des expressions formées de constantes et de variables déjà introduites.

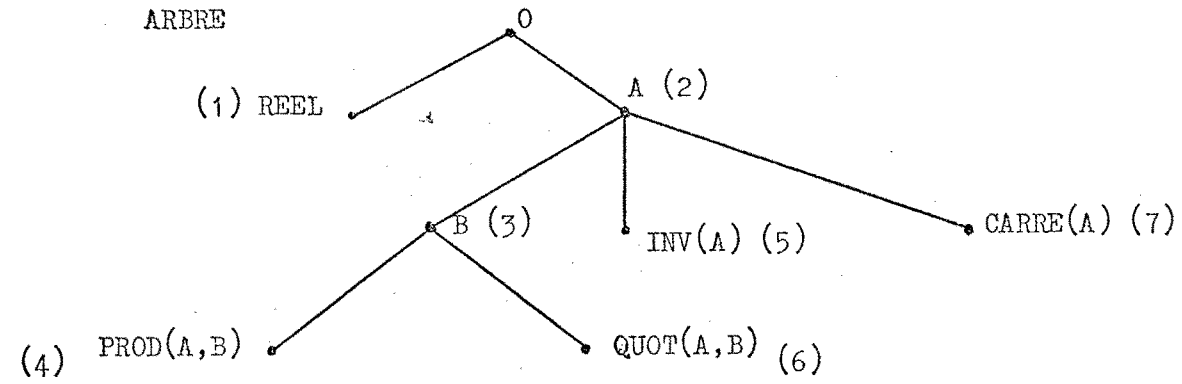
A chaque mot est donc associé une définition et une catégorie.

Les catégories sont définies comme variables ou constantes en mettant 'TYPE' en zone catégorie. Quand de nouvelles catégories sont définies, elles peuvent deve nir la catégorie d'un mot en étant placées en zone catégorie de la ligne où est défini ce mot.

L'indicateur de contexte permet d'accrocher la ligne courante à l'arbre que forme le livre. Si la ligne n ' est dans aucun bloc, son indicateur est la racine de l'arbre : 0 . Si elle est dans le bloc ouvert par la variable a , son indicate est a . Il peut y avoir plusieurs blocs imbriqués. Dans ce cas, on définit la lis des indicateurs de la ligne (LI(N)) . Cette liste correspond aux variables acceptables dans ce contexte. L'identificateur dépend de toutes ces variables.

Exemple de texte en P.A.L

N° de ligne	IN	ID	DEF	CAT
(1)	0	REEL	PN	TYPE
(2)	0	A	EB	REEL
(3)	A	B	EB	REEL
(4)	B	PROD	PN	REEL
(5)	A	INV	PN	REEL
(6)	B	QUOT	PROD(A,INV(B))	REEL
(7)	A	CARRE	PROD(A,A)	REEL



On aurait pu inverser les lignes (3) et (5), (5) et (7). Les permutations de l'arbre sont restreintes par les conditions de précédence dans la définition des objets successifs figurant aux noeuds.

Ainsi, on ne peut pas inverser (1) et (2) car A est de type REEL défini en ligne (1). De même QUOT dont la définition dépend de PROD et INV ne peut être placé plus tôt.

Voyons la liste d'indicateurs des lignes : c'est la suite des noeuds qu'il faut traverser pour remonter jusqu'à la racine. Ex pour INV : A et pour QUOT : B,A

Interprétation :

ligne 1 : la définition PN : notion primitive ou axiome, indique que l'on définit une constante appelée REEL (zone.ID) indépendante de toute variable (hors contexte car IN = 0) qui sera un type.

ligne 2 : hors contexte (IN = 0), A(ID) est une variable (DEF = 'EB') réelle (CAT).

ligne 3 : dans le bloc A , B est une variable réelle.

ligne 4 : en fonction de A et B , PROD(A,B) est une notion primitive de type REEL.

ligne 5 : de même pour INV(A).

ligne 6 : en fonction de A et de B , QUOT(A,B) = PROD(A,INV(B)) est réel.

ligne 7 : en fonction de A , CARRE(A) = PROD(A,A) est un réel.

Dans ces deux derniers cas REEL semble redondant car on peut construire la catégori à partir de la définition, QUOT = PROD( ) quotient et produit doivent donc avoir aussi le même type. Donc QUOT a le type de PROD : REEL.

Le travail se borne à vérifier que le type donné en zone catégorie correspond au type construit. Ceci est très important dans la suite car on utilise cette vérification pour constater qu'une démonstration donne le résultat voulu. La ligne prend la forme :

IN	ID	DEF	CAT
	Théorème	expression qui est la démonstration	énoncé du théorème

Remarque sur les substitutions à effectuer dans le contrôle de validité. Dans la ligne (6) on a PROD(A,INV(B)) alors que PROD est défini avec les paramètres A et B , donc le deuxième paramètre B est devenu INV(B) . Pour que la ligne 6 soit acceptable, il faut que B et INV(B) aient même type. De même en ligne (7), dans PROD,B devenant A , implique que A et B aient même type pour que la substitution soit valide.

Nous allons maintenant préciser la grammaire de P.A.L , puis son utilisation.

Grammaire et logique de P.A.L

On peut distinguer trois niveaux de structure : le livre, la ligne et l'expression.

Le livre vide étant valide, un livre valide est défini récursivement par ajout de lignes valides. Une ligne valide, à la suite du livre 1 a pour indicateur

0 ou une variable définie dans le livre 1. Son identificateur peut être un mot quelconque de l'alphabet, à condition qu'il ne soit pas déjà utilisé. Sa définition peut être 'EB' , 'PN' ou une expression valide Σ_2 . Sa catégorie peut être 'TYPE' ou une expression valide Σ_1 . On veut de plus que la catégorie de Σ_1 soit 'TYPE' et celle de Σ_2 soit Σ_1 modulo certaines substitutions.

Reste à voir la définition d'expression valide dans le contexte θ .

Une variable x est admissible si elle appartient à la liste d'indicateurs de θ c'est à dire au contexte.

Une constante est admissible si le contexte de la ligne où elle est définie est inclu dans le contexte de θ .

Maintenant par récurrence nous allons définir une expression parenthésée admissible.

Si B est une constante définie dans le contexte $x_1, x_2 \ldots x_k$ on a donc dans le livre les lignes suivantes :

$$\begin{array}{llll} 0 & x_1 & EB & \Gamma_1 \\ x_1 & x_2 & EB & \Gamma_2(x_1) \\ \hline \hline x_{k-1} & x_k & EB & \Gamma_k(x_1,\ldots,x_{k-1}) \\ x_k & B\ \Omega(x_1,\ldots,x_k) & & \Sigma(x_1,\ldots,x_{12}) \end{array}$$

Alors $B(E_1,\ldots E_k)$ est admissible si

1°) tous les E_i sont admissibles

2°) les catégories des paramètres correspondants sont égaux modulo certaines substitutions.

$$\begin{array}{ll} CAT(E_1) \overset{=}{D} \Gamma_1 & \\ CAT(E_2) \overset{=}{D} S_{x_1 \to E_1} & \Gamma_2(x_1) \\ \qquad \text{on substitue } E_1 \text{ à } x_1 \text{ dans } \Gamma_2(x_1) & \\ \hline \hline CAT(E_k) \overset{=}{D} S_{\substack{x_1 \to E_1 \\ \hline x_{k-1} \to E_{k-1}}} & \Gamma_k(x_1,\ldots,x_k) \end{array}$$

Alors la catégorie construite de l'expression $B(E_1,\ldots E_k)$ est dans ces conditions :

$$\frac{S_{\substack{x_1 \to E_1 \\ \cdots \\ x_k \to E_k}}}{\Sigma(x_1,\ldots,x_k)}$$

Note sur $A \underset{D}{=} B$.

Il se peut que dans la définition de A on utilise une constante qui soit elle même fonction d'autres constantes par sa définition

exemple : $A(x) = F(H(x),x)$ et $H(x) = G(x,x)$.

Supposons que F et G soient des notions primitives, alors on dit que A est mis sous forme développée quand

$$FD(A) = F(G(x,x),x)$$

car on ne peut plus remplacer une constante par sa définition.

$A \underset{D}{=} B$ est équivalent à $FD(A) = FD(B)$.

Comme cela, on est sûr que A et B sont logiquement équivalents, même s'ils ont des formes différentes.

Exemples de livres PAL

Démonstration de p→p

Ci-dessous se trouve la démonstration de p→p dans le système de Gentzen donnée à la page II.4 est reprise sous forme de livre PAL.
A la page suivante, la même démonstration est donnée mais dans le système de Hilbert. Elle est décrite en page II.2 .

LIVRE

	0	*P*	*EB*	*TYPE*	
	P	*Q*	*EB*	*TYPE*	
	Q	*INCLU*	*PN*	*TYPE*	définition de INCLU
P	*Q*	*PT*	*EB*	*P*	
Q	*PT*	*QT*	*EB*	*Q*	
P→Q	*QT*	*II*	*PN*	*INCLU(P,Q)*	définition de la règle de inclu-introduction
	0	*A*	*EB*	*TYPE*	
A	*A*	*AT*	*EB*	*A*	
A→A	*AT*	*TH*	*II(A,A,AT,AT)*	*INCLU(A,A)*	démonstration immédiate

définition de la logique

définitions de EBF,VRAI,INCLU

0	*EBF*	*PN*	*TYPE*
0	*P*	*EB*	*EBF*
P	*VRAI*	*PN*	*TYPE*
P	*Q*	*EB*	*EBF*
Q	*INCLU*	*PN*	*EBF*
Q	*AX1*	*PN*	*VRAI(INCLU(P,INCLU(Q,P)))* $\vdash P \to (Q \to P)$
Q	*M*	*EB*	*EBF*
M	*AX2*	*PN*	*VRAI(INCLU(INCLU(INCLU(Q,M)),INCLU(INCLU,INCLU(M))))* $\vdash (P \to (Q \to M)) \to ((P \to Q) \to (P \to M))$
Q	*ASP1*	*EB*	*VRAI(P)*
ASP1	*ASP2*	*EB*	*VRAI(INCLU(P,Q))* $\vdash P$ et $\vdash P \to Q$ alors $\vdash Q$
ASP2	*MOPO*	*PN*	*VRAI(Q)*
0	*A*	*EB*	*EBF*
A	*B*	*EB*	*EBF*
B	*H1*		*AX1(A,INCLU(B,A))* *VRAI(INCLU(A,INCLU(INCLU(B,A),A)))* $\vdash A \to ((B \to A) \to A)$
B	*H2*		*AX1(A,B)* *VRAI(INCLU(A,INCLU(B,A)))* $\vdash A \to (B \to A)$
B	*H3*		*AX2(A,INCLU(B,A),A)* *VRAI(INCLU(INCLU(A,INCLU(INCLU(B,A),A)),INCLU(INCLU(A,INCLU(B,A)),INCLU(A,* $\vdash (A \to ((B \to A) \to A)) \to ((A \to (B \to A)) \to (A \to A))$
B	*H4*		*MOPO(INCLU(A,INCLU(INCLU(B,A),A)),INCLU(INCLU(A,INCLU(B,A)),INCLU(A,A)),H1,H3)* *VRAI(INCLU(INCL* $\vdash (A \to (B \to A)) \to (A \to A)$
B	*TH*		*MOPO(INCLU(A,INCLU(B,A)),INCLU(A,A),H2,H4)* *VRAI(INCLU(A,A))* $\vdash A \to A$

7 : Insuffisances de P.A.L.

Dans le système PAL, les fonctions sont définies en extension. Une fonction f, qui a x fait correspondre y, est définie par l'ensemble des couples de constantes

$$(x,y) = (x,f(x)).$$

En d'autres termes, x étant un objet de type T1, on lui associe un objet f(x) de type T2. Mais il n'est pas possible de définir une fonction f du type T1 dans T2.

Les mathématiques modernes utilisent très fréquemment des objets du type fonction. Même dans notre schéma de correspondance objet - type nous utilisons cette notion.

Dans le chapitre suivant, nous verrons que le formalisme du lambda-calcul introduit par Church apporte une solution naturelle à ce problème de l'extension de PAL.

III / Les systèmes du lambda-calcul.

1 : Présentation du lambda-calcul de Church.

Ce système a été introduit pour expliciter les mécanismes de la substitution dans les systèmes formels. Il est donc normal d'y faire appel dans cette étude.

Ce formalisme permet de préciser la notion de fonction. En effet, dans la notation f(x) se trouvent confondues deux notions :

- la définition de la fonction f
- la valeur de f appliquée à x.

Par exemple, soit f la fonction $x \to x+2$ elle sera notée

$$f = \lambda v.v+2 \quad \text{et} \quad f(x) = (\lambda v.v+2)(x) \xrightarrow{\beta} x+2$$

λ est l'opérateur d'abstraction fonctionnelle, son argument v est le paramètre dont dépend la fonction. v+2 est le corps de la définition de la fonction, il permet le calcul effectif de la valeur de celle-ci pour une valeur donnée au paramètre. f(x) note la valeur de f lors de son évaluation avec x comme paramètre. Cette évaluation se fait suivant les règles du lambda-calcul. Dans notre exemple, $\xrightarrow{\beta}$ note la β-réduction, c'est à dire la subtitution de la valeur x au paramètre v.

Dans les systèmes de Gentzen, les règles de déductions sont des fonctions qui, appliquées à des formules (prémisses) donnent une formule (conclusion). Or le lambda-calcul est

justement une formalisation de la notion de fonction calculable, et toute fonction calculable peut être représentée par un lambda-terme. Donc si dans notre système, l'on prend comme couple de règles l'abstraction et l'application, on pourra exprimer toutes les autres règles désirables.

Un autre point à noter pour l'analogie entre le lambda-calcul et les systèmes de Gentzen, concerne l'existence d'une dualité des règles. Dans le système de Gentzen, les règles permettent d'introduire ou d'éliminer un connecteur logique, dans le lambda-calcul, l'abstraction permet d'introduire une variable liée et l'application de l'éliminer.

Dans toute sa généralité le lambda-calcul permet d'éliminer les variables liées (hypothèses) en appliquant n'importe quel terme. Or pour que dans notre interprétation cette opération ait un sens, il faut limiter la classe des termes susceptibles de lever une hypothèse.

Cette restriction est obtenue, dans un lambda-calcul typé par la condition d'applicabilité. C'est à dire, l'exigence que les arguments d'une fonction aient même type que les variables liées qu'elles vont remplacer.

Comparaison entre le lambda-calcul et le lambda-calcul typé.

. Dans le lambda-calcul, l'égalité de deux termes est indécidable.

. Dans le lambda-calcul tous les termes n'ont pas de forme normale.

Ex : $(\lambda x.xx)(\lambda x.xx)$

. Cependant, d'après le théorème de Church-Rosser, si un terme a une forme normale, alors elle est unique.

Cette propriété établit la consistance du lambda-calcul.

. Dans le lambda-calcul typé, tout terme a une forme normale On en déduit son unicité par le théorème de Church-Rosser.

La décidabilité de l'égalité de deux termes résulte alors de la comparaison des formes normales des deux termes.

2 : Un système du lambda-calcul : L.

Nous allons définir les formules de L.

Soit V un ensemble de variables u,v,w...

- les formules atomiques : ce sont les variables $V \subset L$
- les formules composées :
  - + par abstraction $v \in V$ et $w \in L$ alors $[v]w \in L$

    noté classiquement $\lambda v.w$

    v est dite liée dans la formule $[v]w$
  - + par application u et $w \in L$ alors $\langle u \rangle w \in L$

    noté classiquement wu

    u est l'argument de w.

Par exemple $\lambda x.y$ se note $[x]\,y$ et $f(x)$ se note $\langle x \rangle y$.

alors $(\lambda x.y)(x)$ devient $\langle x \rangle [x]\,y$

et $\lambda xy.x(y(y))$ se note $[x][y]\langle\langle y\rangle\, y\rangle\, x$.

Conditions sur les formules.

1) On impose que dans une formule, toutes les variables liées soient distinctes.

2) Lors d'une substitution, il est entendu que le nom des variables liées est modifié de façon à éviter les collisions.

3) Toutes les formules qui ne différent que par le nom de leurs variables liées sont identifiées.

Nous avons défini les formules de L, voici maintenant les règles de conversion.

β : $\langle u\rangle[v]w \xrightarrow{\beta} S(v\backslash u)w$ ou S est la substitution.

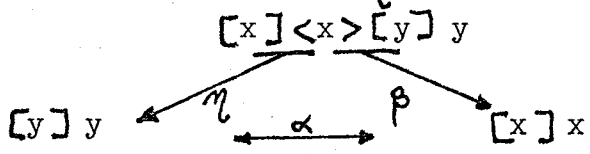
η : $[v]\langle v\rangle w \xrightarrow{\eta} w$ si $v \notin w$.

cette dernière règle peut être omise.

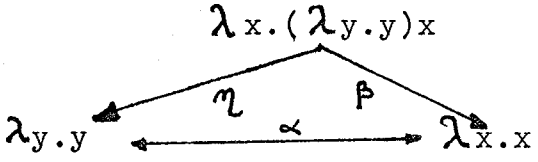
Exemples:

$\langle x\rangle[y][z]\,y \xrightarrow{\beta} [z]\,x$

$\langle[y]\langle y\rangle\,x\rangle\,z \xrightarrow{\eta} \langle x\rangle\,z$



Cette notation permet de localiser plus facilement les et réductions que la notation classique.



Une formule est sous forme normale si on ne peut pas la réduire par application des règles de conversion.

Par exemple si $u,v,w \in V$ alors u, $\langle u\rangle v$ et $[u][v]\,w$ sont sous forme normale.

Dans L, il existe des formules composées par auto-application ($\langle u \rangle u$), par exemple la formule sans forme normale

$$z = \langle [x] \langle x \rangle x \rangle [y] \langle y \rangle y$$

notée classiquement $(\lambda x.xx)(\lambda x.xx)$

On peut constater que :

. il n'y a pas de η-réduction possible sur $[x]\langle x \rangle x$ la condition $v \notin w$ n'est pas remplie.

. il reste la β-réduction $\langle u_0 \rangle [y] w$

ou $u_0 = [x]\langle x \rangle x$ et $w = \langle y \rangle y$

$$z = S(y \backsim u)w = \langle [x_1] \langle x_1 \rangle x_1 \rangle [x_2] \langle x_2 \rangle x_2$$

La substitution est faite en respectant la condition 2, la formule obtenue est conforme à la condition 1 et par la condition 3, u_1 et u_2 sont identiques à u_0.

De plus, le résultat est z lui même (condition 3)

Donc $z \xrightarrow[\beta]{} z \xrightarrow[\beta]{} z \quad \ldots$

z n'a pas de forme normale.

3 : Typification de L. Le système D.

Soit B un ensemble de types primitifs.

T l'ensemble des types sur B, est défini ainsi :

1 - B appartient à T.

2 - Si $a \in T$ et $b \in T$ alors $(a \rightarrow b) \in T$.

La typification est une application de L dans T qui a chaque formule de L associe, soit un type de T, soit $\perp$ qui note l'indétermination :

$$\tau : L \rightarrow T \cup \{\perp\}$$

Si $v \in V$ alors par définition $\tau(v) \in T$.

les variables sont classées par type lors de leur définition.

Soit $v \in V$, w et $u \in L$ tels que $\tau(v)=a$, $\tau(u)=b$, $\tau(u)=c$ alors

$$\tau([v]w) = (a \to b)$$

et $\tau(\langle u \rangle w) = d$ si $b=c \to d; \perp$ sinon.

Exemples :

Si $\tau(x)=a$, $\tau(y)=a$ et $\tau(z)=b$ quel est le type de $E = \langle x \rangle [y][z]y$?

Soit $w = [y][z]y$, $E = \langle x \rangle w$ et

$$\begin{aligned}\tau(w) &= \tau([y][z]y) \\ &= (\tau(y) \to \tau([z]y)) \\ &= (a \to (\tau(z) \to \tau(y))) \\ &= (a \to (b \to a))\end{aligned}$$

d'ou $\tau(E) = \tau(\langle x \rangle w) = (b \to a)$ puisque $\tau(x)=a$.

On peut constater que le type de E est le même que celui de sa forme normale $[z]y$.

De même, $\tau([x]\langle x \rangle [y]y) = \tau([x]x) = (a \to a)$

Cependant si $\tau(x)=a$ et $\tau(y)=b$ alors

$\tau([x]x) = (a \to a)$ mais $\tau([x]\langle x \rangle [y]y) = \perp$

car $\tau(\langle x \rangle [y]y) = \perp$ en effet $\tau(x)=a$ et $\tau([y]y) = (b \to b)$.

Pour qu'un terme et sa forme normale aient le même type, il ne faut considérer qu'un sous-ensemble de L.

Nous allons définir D comme étant la partie de L dont les termes ont un type défini.

$$D = L - \tau^{-1}(\perp)$$

D ne contient que les termes qui vérifient la condition d'applicabilité. Tous les termes de D ont une forme normale, $z = \langle [x]\langle x \rangle x \rangle [y]\langle y \rangle y$ n'appartient pas à D puisque $\tau(z) = \perp$.

Si x est de type a, $\langle x \rangle x$ est de type $\perp$ puisque $a \neq a \to b$.

4 : Isomorphisme entre D et déductions dans G.

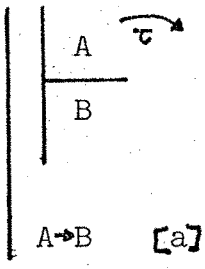
Les formules de T et de G étant les mêmes, nous allons établir la correspondance entre

- abstraction et →introduction
- application et →élimination

sur la base de l'interprétation suivante

" a pour type " se lit " est une preuve de ".

+ Règle →introduction



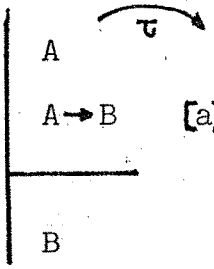
soit a une preuve de A.

b est une preuve de B dépendant de la preuve a de A.

alors [a]b est une preuve de A→B. En effet c'est une construction qui à toute preuve a de A associe une preuve b de B.

Ceci prouve que [a]b est l'équivalent de la règle →I.

+ Règle →élimination



soit a une preuve de A.

soit a b une construction qui a toute peuve u de A associe une preuve b de B.

alors on obtient une preuve de B en appliquant a à la construction [a]b.

et $\langle a \rangle [u]b \xrightarrow[\beta]{} b$

L'application suivie de la β-réduction est donc bien l'équivalent de la règle →E.

Les théorèmes de G sont les déductions qui ne dépendent d'aucune hypothèse.

L'introduction d'une hypothèse se traduit par l'apparition d'une variable. Son élimination se traduit par son abstraction, de la même façon que pour la règle →I.

Un théorème de T est donc le type d'une expression close de D. Une expression est dite close, si elle ne contient pas de variable libre.

Dans cette optique, les β-réductions correspondent à la simplification de la démonstration.

Ex : démonstration dans T de $p\to((p\to q)\to q)$.

(preuve) D $\xrightarrow{\tau}$	T (enoncé)	
a	p	Voir la similitude
b	$p\to q$	avec la démonstration
$\langle a\rangle b$	q	de la page II.5
$[b]\langle a\rangle b$	$(p\to q)\to q$	
$[a][b]\langle a\rangle b$	$p\to((p\to q)\to q)$.	

Il est clair sur cet exemple, que le lambda-calcul typé sur T, établit la formalisation des déduction dans T.

T est un calcul implicationnel très simple.

Cet isomorphisme est étendu par Stenlund au calcul des prédicats du second ordre utilisant les symboles → et ∀ .

On peut donc formaliser les démonstrations d'un système plus fort. Par exemple le lambda-calcul lui même. On obtient ainsi un lambda-calcul dont les types sont des lambda-termes. C'est l'étude d'un tel système qu'a entrepris Nederpelt.

Dans le chapitre suivant, nous étudierons un tel système.

IV / Le système de Nederpelt.

1 : Définition.

La présentation adoptée est différente de celle de la thèse de Nederpelt. Les aspects techniques de ses démonstrations étant omis, les définitions sont plus adaptées aux considérations que l'on développera dans ce chapitre.

Soit M = (A,F,R) le sytème formel suivant :

A alphabet constitué de

- l'ensemble V des variables x,y,z...
- la constante $\top$
- les symboles auxiliaires $[\ ]\ <\ >\ \in$

F ensemble des formules

- formules atomiques $V \subset F$ et $\top \in F$
- formules composées

si a et b $\in$ F alors $<a>\, b \in F$

si $x \in V$, a et b $\in$ F et $x \notin a$ alors

$[x \in a]\, b \in F$

La variable x de $[x \in a]$ b est dite liée.

Une variable non liée est dite libre.

La notation a(x) indique que x est une variable libre de a.

R ensemble des règles

1 - α-équivalence : toutes les formules qui ne différent que par le nom de leurs variables liées sont identifiées.

2 - Toutes les variables liées d'une formule doivent être distinctes entre elles et distinctes des variables libres apparaissant dans cette formule.

3 - Lors des substitutions, le nom des variables liées est modifié de façon à respecter la règle 2.

4,5 - β et η contractions.

+ La <u>contraction</u> $a \longrightarrow b$

par définition a est le redex et b le contracté.

β-contraction $\quad \langle b\rangle[x\in E]a(x) \xrightarrow[\beta]{} a(b) = S(x\backsim b)a$

β-redex $\qquad$ β-contracté

η-contraction $\quad [x\in E]\langle x\rangle a \xrightarrow[\eta]{} a \quad$ si $x \notin a$

η-redex $\qquad$ η-contracté

+ La <u>réduction</u> notée $\xrightarrow{*}$ est la clôture reflexive et transitive de la contraction.

+ La <u>1-réduction</u>, réduction en une étape : $a \xrightarrow[1]{} b$.

b est obtenue par une suite standard de contractions des redex de a. b lui même peut alors contenir de nouveaux redex qui n'apparaisaient pas dans a.

+ La <u>réduction en n étapes</u>.

Définition inductive : $a \xrightarrow[0]{} a$

$a \xrightarrow[n+1]{} c$ si $\exists b$ tq $a \xrightarrow[n]{} b \xrightarrow[1]{} c$.

<u>Propiété</u> : si $a \xrightarrow{*} b$ alors $\exists n$ tq $a \xrightarrow[n]{} b$.

Précisons la définition de la 1-réduction pour les deux types de contractions.

$a \xrightarrow[\beta_1]{} b$ est la relation engendrée par :

1 - si $a \xrightarrow[\beta]{} b$ alors $a \xrightarrow[\beta_1]{} b$

2 - si $a \xrightarrow[\beta_1]{} b$ alors $\langle c\rangle a \xrightarrow[\beta_1]{} \langle c\rangle b$

3 - si $a \xrightarrow[\beta_1]{} b$ alors $[x\in a]c \xrightarrow[\beta_1]{} [x\in b]c$

4 - si $a \xrightarrow[\beta_1]{} b$ alors $\langle a\rangle c \xrightarrow[\beta_1]{} \langle b\rangle c$

et 5 qui peut être déduit des précédents

si $a \xrightarrow[\beta_1]{} b$ alors $[x\in c]a \xrightarrow[\beta_1]{} [x\in c]b$

Les quatres dernières relations sont les règles de monotonie.

Pour $a \xrightarrow[\eta_1]{} b$ il suffit de remplacer β par η dans la définition précédente.

Un terme a est sous <u>forme normale</u> s'il ne contient pas de redex.

Un terme a est <u>normalisable</u> s'il existe une réduction $a \xrightarrow{*} b$ telle que b soit sous forme normale.

Un terme est <u>fortement normalisable</u> si TOUTE réduction se termine.

<u>La typification.</u>

C'est une application τ de M dans $M \cup \{\perp\}$ telle que :

$$\tau(T) = T$$

$$\tau(x) = \begin{cases} a \text{ si } x \text{ est liée par } [x\in a] \\ \perp \text{ si } x \text{ est libre} \end{cases}$$

$$\tau([x\in a]b) = [x\in a]\tau(b)$$

$$\tau(\langle a\rangle b) = \begin{cases} c & \text{si } \tau(b) = [x\in a]c \\ \\ \perp & \text{sinon.} \end{cases}$$

Le système N de Nederpelt est la partie de M dont l'image par τ est différente de $\perp$.

Dans N, $[x\in a]b$ est équivalent à $[x]b$ avec $\tau(x)=a$.

Comme le type d'une variable libre est $\perp$, toutes les formules de N sont closes.

Soit SFN l'ensemble des sous formules de N

$$SFN = \{(E,A)/E\in N \text{ et } A\in M\} \quad \text{notation } A_E\in SFN.$$

Une sous formule A de E peut contenir des variables libres. Cependant, on peut attribuer à ces variables libres dans A, mais liées dans E, le type qu'elles ont dans E.

Si $A_E\in SFN$, le contexte de A est l'ensemble des variables libres qu'il contient.

2 : Les résultats obtenus par Nederpelt.

§ Dans le système M.

+ Pour la β-contraction.

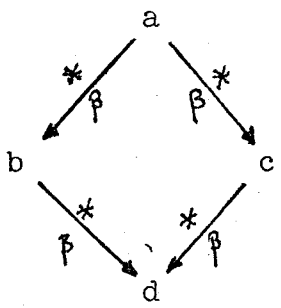
. Théorèmes de clôture

1 - Si $a \in M$ et $a \xrightarrow[\beta_1]{} b$ alors $b \in M$.

2 - Si $a \in M$ et $a \xrightarrow[\beta]{} b$ alors $b \in M$.

. Propriété de Church-Rosser

Si $a \in M$ et $a \xrightarrow[\beta]{*} b$ et $a \xrightarrow[\beta]{*} c$ alors $\exists d$ tq



+ Pour la η-contraction

. Théorème de clôture

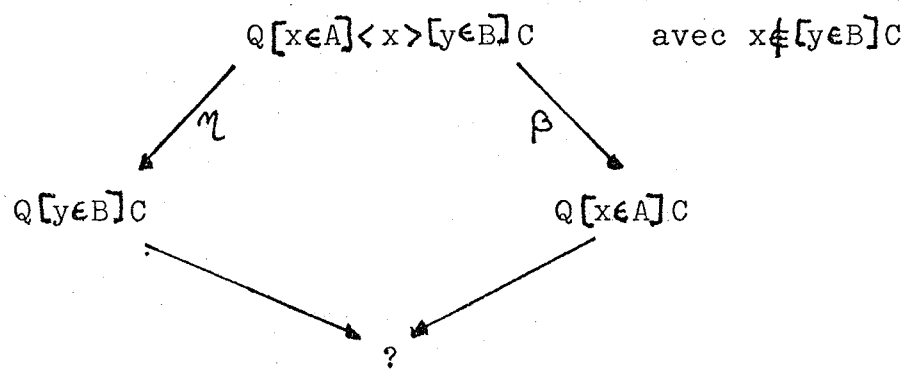
Si $a \in M$ et $a \xrightarrow[\eta_1]{} b$ alors $b \in M$.

. Théorème de séparation des réductions.

Si $a \in M$ et $a \xrightarrow{*} c$ alors $\exists b$ tq $a \xrightarrow[\beta]{*} b \xrightarrow[\eta]{*} c$.

. La propriété de Church-Rosser n'est pas vraie pour les $\beta\eta$-réductions.

Contre-exemple :



A et B sont quelconques !

§ Dans le système typé N les résultats sont plus intéressants.

. Clôture par typification

Si $a \in N$ alors $\tau(a) \in N$.

. Définition d'une norme qui est la typification itérée de la formule et de ses sous formules.

La norme étant liée à la typification, tous les termes de M n'ont pas de norme. Par contre ceux de N en ont une.

. Théorème de normalisation

Toute formule normable est normalisable.

. Forme normale dans N

Si a appartient à N alors il existe b, sous forme normale tel que $a \xrightarrow{*} b$. La démonstration donne une procédure pour obtenir b.

Nederpelt démontre de plus la forte normalisation dans N.
Il a énoncé deux conjectures qui ont été démontrées depuis par Van Daalen.

. La première est la propriété de Church-Rosser pour les $\beta\eta$-réductions dans N

Ce résultat établit la consistance du système et l'unicité des formes normales.

. La seconde conjecture est la clôture par réduction.

Si $a \in N$ et $a \xrightarrow{*} b$ alors $b \in N$.

De plus le système N est décidable pour les relations " égalité entre termes " et "être le type de".

Toutes ces propriétés font de N un objet mathématique bien adapté à notre problème.

Dans le chapitre suivant nous allons étudier la typification. Cette étude permettra de saisir comment utiliser ce système pour la formalisation de textes mathématiques.

3 : Etude de la typification et des opérateurs associés.

Dans le système N, ce qui nous intéresse au point de vue de la théorie de la démonstration, ce sont les termes sous forme normale.

Le paragraphe précédent a montré l'existence et l'unicité de la forme normale de tout terme de N. La relation "avoir même forme normale" est donc une relation d'équivalence dans N.

Soit K l'ensemble des formes normales de N.

$x \in K$ est de la forme QT

ou Q est une chaine d'abstracteurs qui peut être vide.

$Q = q_1 \ldots q_n$ ou $q_i = [x \in E_i]$ E_i est une expression de contexte $\{x_1 \ldots x_{i-1}\}$

T est une T-expression de contexte $\{x_1 \ldots x_n\}$.

Une T-expression est de la forme τ ou x_i ou $\langle E_\ell \rangle x_i$.

E_ℓ est une expression de contexte $\{x_1 \ldots x_n\}$.

Exemples :

ex1 = $[a \epsilon T][b \epsilon T][H1 \epsilon a][H2 \epsilon [x \epsilon a] b] \langle H1 \rangle H2$

ex2 = $[a \epsilon T][b \epsilon [x \epsilon T] a][c \epsilon \langle a \rangle b] c$

ex3 = $[a \epsilon T][b \epsilon [x \epsilon T] a][c \epsilon [y \epsilon a][z \epsilon y] b] \langle\langle a \rangle b \rangle c$

Pour ex3 le contexte de q_3 est : a, b

le contexte de T est : a, b, c

le contexte de q_{32} est : a, b, y.

Quelques définitions relatives à la structure des termes.

On appelle valence de x le nombre n d'abstracteurs.

$v(x) = v(q_1 \ldots q_n) = n$

On appelle ordre de x le nombre maximum d'emboîtements des abstracteurs (noté o). De façon plus précise :

$x = QT$ $o(x) = o(Q)$ et $o(T) = 0$ (chaine d'abstracteurs vide).

et $o(Q) = \max_i (o(q_i))$

avec $o(q_i) = o([x_i \in E_i]) = 1 + o(E_i)$.

On appelle pivot de x l'indice du q_p le plus à droite d'ordre maximum : $o(q_p) = o(x)$ et $o(q_j) < o(x)$ pour $p < j \leqslant v(x)$.

Exemples :

$ex4 = [a \in T][b \in [x \in T]a][c \in a]c$

	ex1	ex2	ex3	ex4
v	4	3	3	3
o	2	2	3	2
p	4	3	3	2

Après avoir précisé la structure des termes de K, voici quelques définitions de fonctions liées à la typification.

Affinons la définition de τ pour $x \in K$.

$\tau(x) = \tau(QT) = Q\tau(T)$

et pour $T = \mathbf{T}$, $\tau(\mathbf{T}) = \mathbf{T}$

$T = x_i$, $\tau(x_i) = E_i$

$T = \langle E_i \rangle^* x_i$, $\tau(\langle E_i \rangle^* x_i) = c$ ou $\tau(x_i) = q_1 \ldots q_j Q'T'$

$\langle E_i \rangle^* = \langle e_1 \rangle \langle e_2 \rangle \ldots \langle e_j \rangle$ de façon que

$\langle e_1 \rangle \langle e_2 \rangle \ldots \langle e_j \rangle q_1 \ldots q_j Q'T' \xrightarrow{*} Q'T' = c$.

Définition de la norme ρ

1 . $\rho(\mathbf{T}) = \mathbf{T}$

2 . $\rho(x_i) = \rho(E_i)$

3 . $\rho([x \in b]c) = [x \in \rho(b)]\rho(c)$

4 . $\rho(\langle b \rangle c) = e$ ou $\rho(c) = [y \in d]e$ et $d = \rho(b)$.

Comparaison de τ et de ρ .

La relation 3 donne la première différence

$x = QT \quad \tau(x) = Q\tau(T) \quad$ et $\quad \rho(x) = \rho(Q)\rho(T)$

donc ρ s'applique à la formule et à toutes ses sous formules, alors que τ ne s'applique qu'à la formule.

La relation 2 donne la seconde différence

$$\tau(x_i) = E_i \quad \text{et} \quad \rho(x_i) = \rho(E_i)$$

s'applique de façon itérée jusqu'à ne plus avoir d'occurence de variables liées. Il ne reste plus alors que la constante τ dans la formule.

Définition du degré μ

$$\mu(\tau) = 0$$
$$\mu(x) = \mu(T)$$
$$\mu(T) = 1 + \mu(\tau(T)).$$

Le degré mesure le nombre de fois que l'on peut typifier une expression avant qu'elle ne soit son propre type.

Exemples :

ex1 = [a∈τ][b∈τ][H1∈a][H2∈[x∈a]b]<H1>H2

τ(ex1) = ... b

τ^1(ex1) = ... τ

ρ(ex1) = [a∈τ][b∈τ][H1∈τ][H2∈[x∈τ]τ]τ

μ(ex1) = 2

$$ex2 = [a\in\tau][b\in[x\in\tau]a][c\in\langle a\rangle b]c$$

$$\tau(ex2) = \ldots\ldots \langle a\rangle b$$

$$\tau^2(ex2) = \ldots\ldots a$$

$$\tau^3(ex2) = \ldots\ldots \tau$$

$$\rho(ex2) = [a\in\tau][b\in[x\in\tau]\tau][c\in\tau]\tau$$

$$\mu(ex2) = 3$$

$$ex3 = [a\in\tau][b\in[x\in\tau]a][c\in[y\in a][z\in y]b]\langle\langle a\rangle b\rangle c$$

$$\tau(ex3) = \ldots [u\in\langle a\rangle b]b$$

$$\tau^2(ex3) = \ldots [u\in\langle a\rangle b][v\in\tau]a$$

$$\tau^3(ex3) = \ldots [u\in\langle a\rangle b][v\in\tau]\tau$$

$$\rho(ex3) = [a\in\tau][b\in[x\in\tau]\tau][c\in[y\in\tau][z\in\tau][w\in\tau]\tau][u\in\tau][v\in\tau]\tau$$

$$\mu(ex3) = 3$$

Les expressions de degré 0 sont de la forme $Q\tau$.

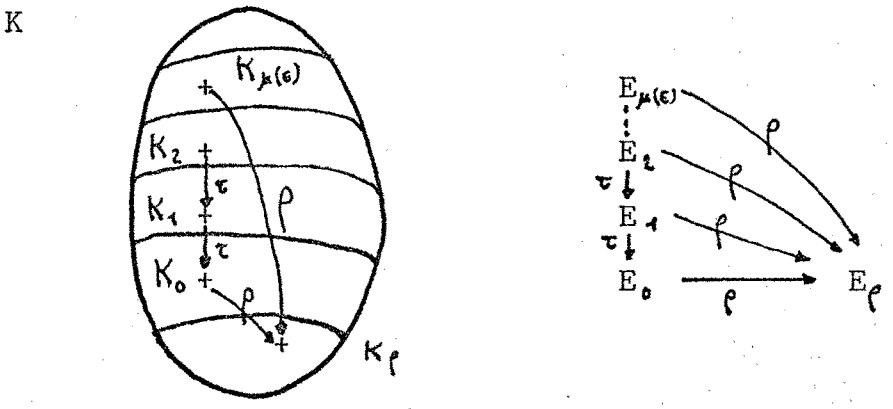
Elles sont invariantes par τ : $\tau(Q\tau) = Q\tau(\tau) = Q\tau$.

La relation "avoir même degré" est une relation d'équivalence qui partionne K en sous ensembles K_0, K_1, ...K_h ...

τ est une application de K_{h+1} dans K_h .

Définisons K_ρ comme étant le sous ensemble de K_0 tel que toutes les sous-formules des formules de K_0, appartiennent à K_0.

La norme ρ est donc la projection de K dans K_ρ.



Relations entre ρ et τ .

+ absorption de τ par ρ

$$\tau(\rho(x)) = \rho(x) \quad \text{évident.}$$

et $\rho(\tau(x)) = \rho(x)$ preuve :

$$\rho(\tau(x)) = \rho(\tau(QT)) = \rho(Q\tau(T)) = \rho(Q)\rho(\tau(T)).$$

Il faut donc que $\rho(\tau(T)) = \rho(T)$. Examinons les 3 cas possibles suivant T :

Si $T = \tau$, $\tau(\tau) = \tau$ c'est vrai.

Si $T = x_i$, $\tau(x_i) = E_i$ d'ou $\rho(\tau(x_i)) = \rho(E_i)$

et par définition $\rho(x_i) = \rho(E_i)$.

Si $T = \langle e \rangle x_i$, $\tau(T) = c$ si $\tau(x_i) = [x \in E]c$

alors $\rho(\tau(T)) = \rho(c)$

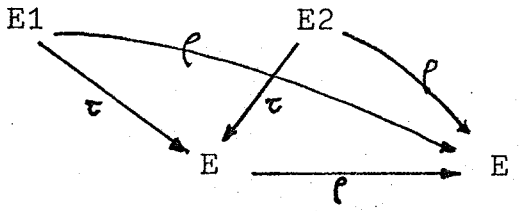
d'autre part, par définition si $\rho(x_i) = [x \in a]b$ et $\rho(e) = a$

$\rho(\langle e \rangle x) = b$ mais $\rho(x_i) = \rho(\tau(x_i)) = \rho([x \in E]c) = [x \in \rho(E)]\rho(c)$

d'ou $b = \rho(c)$, en conséquence $\rho(\tau(T)) = \rho(T)$. CQFD

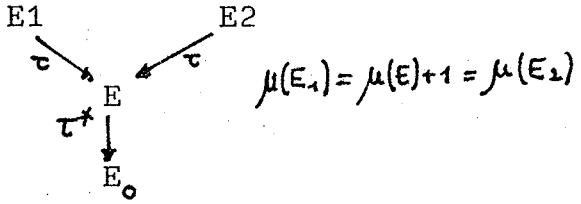
$+\tilde{\tau}$ est une relation plus fine que $\tilde{\rho}$.

Si deux expression ont même type, elles ont même norme.



$+\tilde{\tau}$ est une relation plus fine que $\tilde{\mu}$.

Si deux expressions ont même type, elles ont même degré.



Ce n'est pas vrai pour la norme car $\rho(x) = \rho(\tau(x))$
et $\mu(x) = \mu(\tau(x))-1$.

+ Invariance de l'ordre.

Si $x = QT$, $\tau(x) = Q\tau(T)$ mais $\tau(T)$ est une partie d'une sous formule de Q, donc, d'ordre strictement inférieur à $o(Q)$.
En conséquence $o(x) = o(\tau(x))$.
De plus Q étant invariant, le q.i. d'ordre maximum lui appartenant,

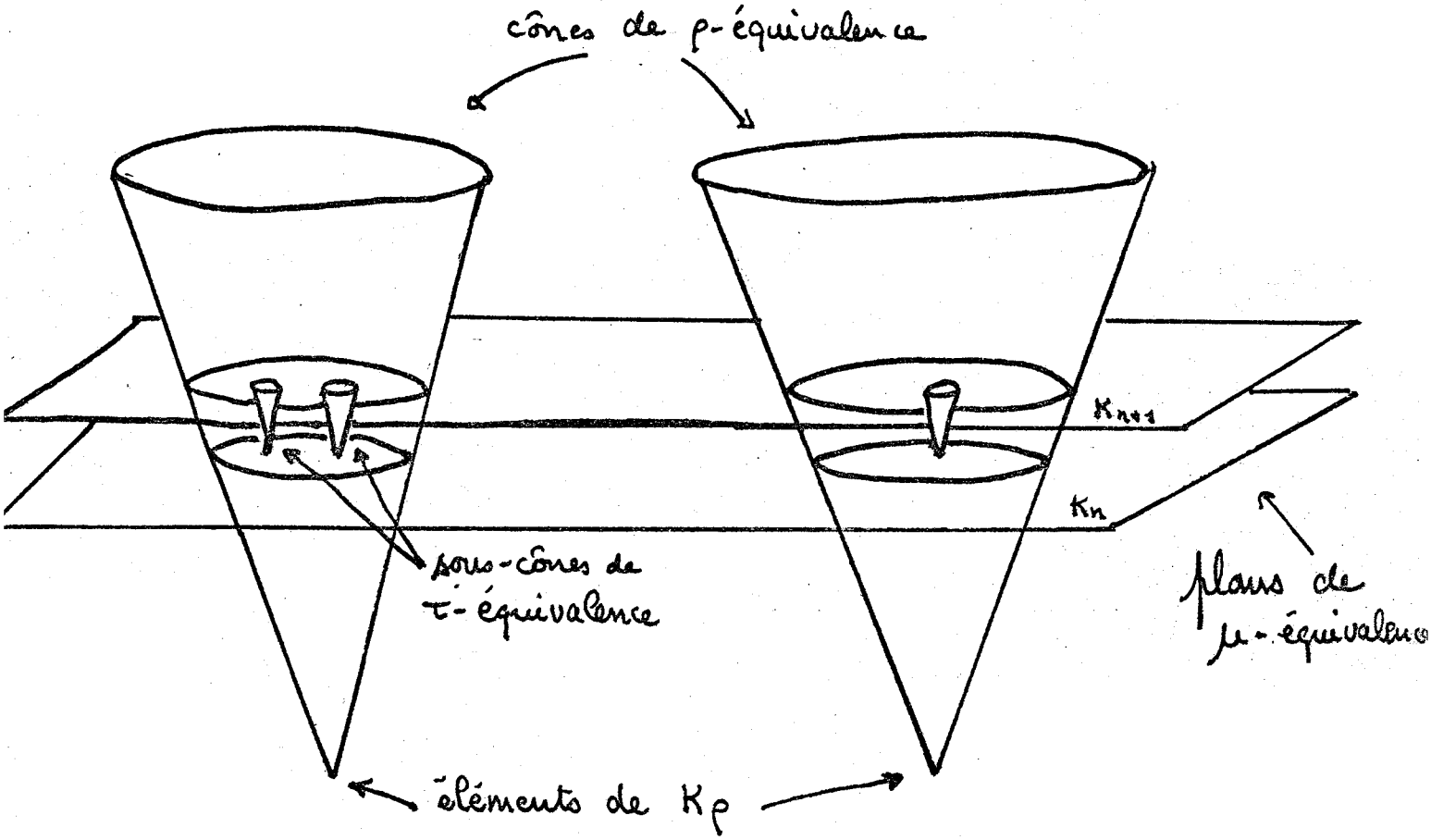
$$p(x) = p(\tau(x)).$$

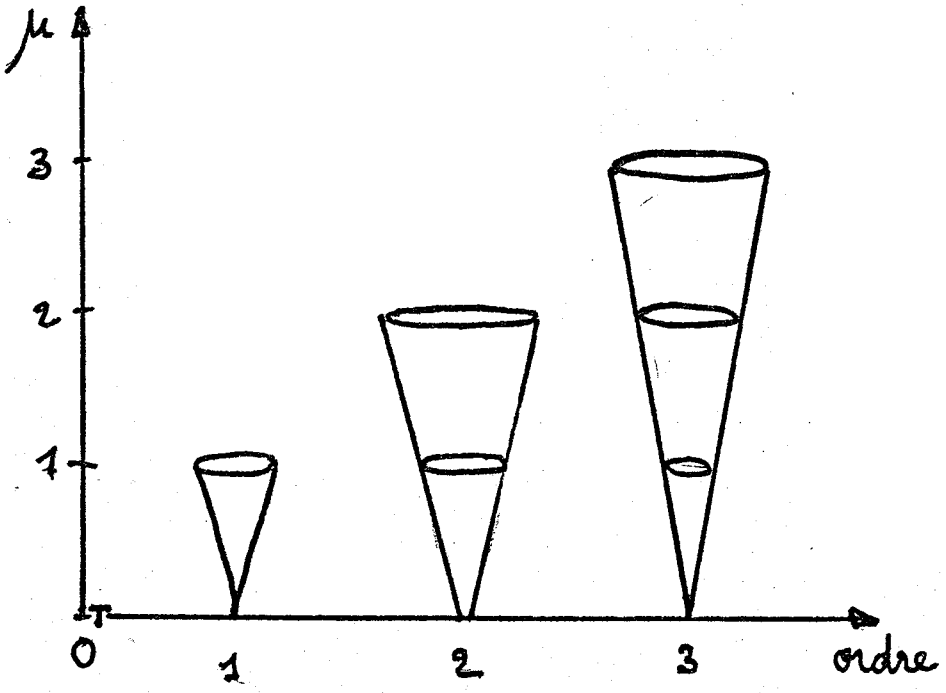
L'ordre étant invariant par typification, il en est de même pour la norme, celle-ci étant la clôture de la typification pour la formule et ses sous-formules.

$$o(x) = o(\rho(x)) \quad \text{et} \quad p(x) = p(\rho(x)).$$

+ Intervalle de variation de la valence.

$\tau(T)$ étant une formule de K, $\tau(T) = Q'T'$
donc $\tau(x) = \tau(QT) = Q\tau(T) = QQ'T'$ $\quad v(\tau(x)) \geqslant v(x)$.
La valence est indépendante des sous-formules de x,
en conséquence $v(\rho(x)) = v(\tau^{\mu(x)}(x))$.
d'ou $\quad v(\rho(x)) \geqslant v(\tau(x)) \geqslant v(x) \geqslant p(x)$


cônes de ρ-équivalence
K_{n+1}
K_n
sous-cônes de τ-équivalence
plans de μ-équivalence
éléments de Kρ


μ
3
2
1
0
1
2
3
ordre

Interprétation de la typification.

Nous avons vu au chapitre précédent que la correspondance de Howard peut se résumer par le tableau suivant :

démonstration	énoncé
objet	type
lambda-terme de D	formule de T.

T étant le langage, la formalisation des démonstrations dans T est l'objet du métalangage D sur T.

Voyons comment cette interprétation peut s'étendre à K. Dans les livres PAL, il y a deux sortes de lignes. Les lignes de définition et les lignes de démonstration.

	DEF	CAT
ligne de définition	-	τ
ligne de démonstration	D	E

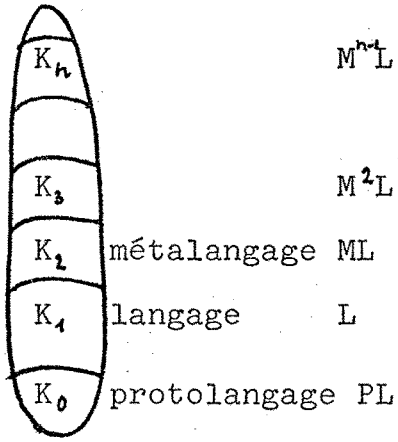
et $\tau(D) = E$ avec $\tau(E) = \tau$.

Donc les expressions de degré 0 servent plutôt pour la définition du langage qui est constitué des expressions de degré 1. Les expressions de degré 2 sont les démonstration, donc le métalangage.

On peut donc considérer la typification comme une fonction permettant de passer du métalangage au langage.

Application de cette interprétation à K.

K



Pourquoi un protolangage ?

Le langage L que l'on considère est typé, le protolangage sert à définir ce langage typé dans les systèmes AUTOMATH. K_0 ne contient que des expressions de la forme $Q\tau$. Le véritable langage est K_1, il comprend les variables de type τ

les fonctions de type $[x\epsilon\tau]\tau$ ...

les fonctionnelles ...

Une expression de K_1 étant l'énoncé d'une démonstration de degré 2, donc appartenant à K_2.

Dans AUTOMATH, les expressions sont limitées au degré 2. Mais on peut très bien considérer les expressions de degré supérieur. Dans M^2L par exemple, on peut exprimer les propriétés des démonstrations dans L...

Qu'est que trouver une démonstration ?

Trouver une démonstration de E, c'est trouver une expression D telle que $\tau(D) = E$. Il est donc intéressant de définir une fonction réciproque de τ. La fonction π de preuve de K_n vers K_{n+1} qui a E de K_n fait correspondre une classe d'équivalence pour $\tilde{\tau}$ dans K_{n+1}. Avant de définir π, donnons quelques résultats relatif à la structure de D tel que $D = \tau(E)$.

1 . $o(D) = o(E)$ l'ordre est invariant par π puisqu'il l'est par τ .

2 . $\rho(D) = \rho(E)$

3 . $p(\rho(E)) \leqslant v(D) \leqslant v(E) \leqslant v(\rho(E))$

4 . la partie $q_1 \ldots q_{\rho(E)}$ est invariante par π .

Associé à τ nous avons défini le degré μ d'une expression. de même à π nous pouvons associer ν le codegré de l'expression. Une remarque d'abord, une expression n'a pas forcement de démonstration, dans ce cas $\pi(E) = \bar{\Omega}$ (vide).

Par définition $\quad \nu(\bar{\Omega}) = 0$

$$\nu(x) = 1 + \nu(\pi(x)).$$

Expression de codegré maximum.

Soit $x_0 \in K_\rho$, $x_0 = Q\tau$ et soit i le pivot de x, alors $x_1 = Qx_i$, puis l'on considére $q_i = [x_i \in E_i]$ où $E_i \in K_\rho$ alors de même que pour x_0 on construit $E_i' = Q'x_j$

donc $x_2 = q_1 \ldots q_{i-1} [x_i \in Q'x_j] q_{i+1} \ldots q_n x_i$

$$\mu(E_i) = \mu(x_0) - 1$$

et l'on continue ainsi jusqu'à obtenir la sous expression de degré 0.

Le degré de l'expression finale x est égal à l'ordre de x .

$$\text{Donc} \quad \mu(x) \leqslant o(x)$$
$$\text{et} \quad \varphi(x) \leqslant o(x) - \mu(x)$$

Cette dernière relation permet dans bien des cas de savoir que $\pi(x) = \emptyset$ quand $\varphi(x) = 0$.

Dans le cas où $\pi(x)$ est non vide, voyons la construction de π.

Soit D(x) l'ensemble des variables liées de x.

Considérons l'ensemble des T-expressions de contexte D(x)

$$\mathcal{C} = \langle E_{D(x)} \rangle^* \; D(x)$$

Sur $\mathcal{C}$ modulo la relation d'équivalence $\overset{\sim}{\tau}$, π est l'isomorphisme inverse de τ

$$\mathcal{C}/\overset{\sim}{\tau} \underset{\pi}{\overset{\tau}{\rightleftarrows}} \mathcal{P}(D(x))$$

On peut alors définir π pour tout X par

$$\pi(X) = \{E = QT \text{ tq } X = QQ'T' \text{ où } Q'T' \in \mathcal{P}(D(Q)) \text{ et } T = \pi(Q'T')\}$$

Exemple :

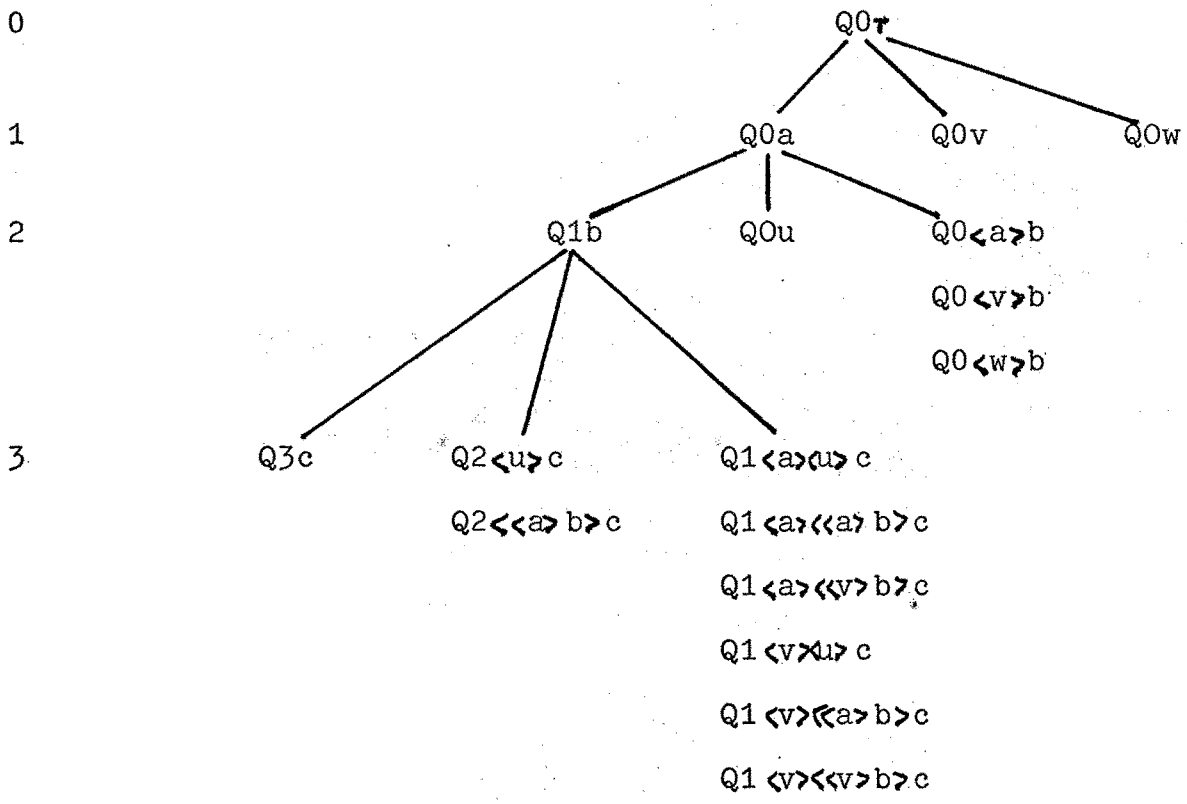
$Q3 = [a \in \tau][b \in [x \in \tau]a][c \in [y \in a][z \in \tau]b$

$Q2 = Q3[u \in a]$

$Q1 = Q2[v \in \tau]$

$Q0 = Q1[w \in \tau]$

Par application répétée de π à $Q0_\tau$, on obtient l'arbre suivant



+ La branche la plus à gauche est celle de codegré maximum.
+ Les fils qui contienent des applications produisent des β-réductions par typification.
+ Les fils d'indice de Q maximum sont ceux de contexte minimum.

Ayant un ensemble de preuves de X, le problème est de sélectionner la plus "intéressante". On peut appliquer divers critères. Par exemple :

- l'expression de codegré maximum
- l'expression de contexte minimum
- l'expression de moindre réduction
  c-à-d celle qui minimise les β-réductions lors de la typification.
- on peut aussi imposer des conditions au niveau de M^2L ce qui permet de sélectionner une démonstration plutôt que d'autres.

Les fonctions τ et π permettent de cerner de plus prés la notion de démonstration. Nous avons donné quelques résultats relatifs à la structure des démonstrations. Cependant une étude plus approfondie reste encore à faire.

V / La famille des langages AUTOMATH.

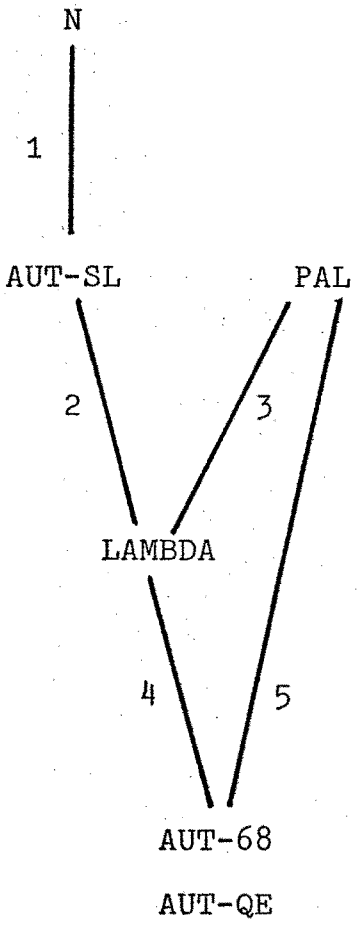
1 : Présentation historique du projet AUTOMATH.

Commencé en 1967, par une équipe de l'université d'Eindhoven, sous la direction du professeur N.G.De Bruijn, le projet AUTOMATH est popularisé par une communication au Symposium de la démonstration automatique de l'IRIA, en décembre 1968. Cette première présentation décrit PAL et une première version du langage AUTOMATH (AUT-68).

Déjà dans cet article, des possibilitées d'extensions sont envisagées. La pratique de l'écriture de livres en AUTOMATH, conduit à une version plus souple et plus concise AUT-QE. Cette version est l'objet du volume 1 du Symposium APLASM d'Orsay de décembre 1973.

En se penchant sur les fondements mathématiques de ces systèmes, Nederpelt a étudié un système LAMBDA-AUT en 1971. Les idées essentielles sont contenues dans AUT-SL de De Bruijn (fin 1971). Un approfondissement de ces idées fait l'objet de la thèse de Nederpelt (Juin 1973). Elles ont été étudiées au chapitre précedent.

2 : Relations entre ces langages.



1 - N est un système formel du lambda-calcul lambda-typé. AUT-SL est une réalisation d'un vérificateur de démonstrations en une seule ligne. Il fournit le type d'une expression et sa forme normale. Cette expression étant considérée comme la démonstration d'un théorème, il faut vérifier que le type de l'expression correspond à l'énoncé du théorème. Alors, la forme normale correspond à la démonstration la "plus simple" du théorème.

AUT-SL sera décrit au paragraphe 3.

2,3 - LAMBDA est la généralisation de AUT-SL en une version en plusieurs lignes. Elle utilise le format des lignes de PAL, son mode d'introduction des variables et des constantes et son système d'indicateur de contexte. Le système de substitution de PAL est remplcé par le couple abstraction-application qui par β-réduction, lors de la mise sous forme normale, provoque les substitutions.

Ex: si f(x,y) par changement de contexte donne f(a,b) dans PAL, alors dans LAMBDA, il s'écrit $\langle y\rangle\langle x\rangle f$

Ceci dans le contexte x, y; le contexte étant adjoint, on obtient $[x][y]\langle y\rangle\langle x\rangle f$

et l'application à a et b donne

$$\langle b\rangle\langle a\rangle[x][y]\langle y\rangle\langle x\rangle f \xrightarrow[\beta]{} \langle b\rangle\langle a\rangle f$$

La β-réduction donne bien le résultat.

LAMBDA sera décrit au paragraphe 4.

4,5 - Dans AUT-QE et AUT-68 on permet les deux formes de substitution. De plus, on limite le degré des expressions à 2 et on impose aux expressions typicales d'être de type T.

Dans AUT-68, il existe une seule expression de degré 0 : T. Pour cela on ajoute la règle $[x]T \to T$.

Dans AUT-QE, on accepte les expressions de degré 0, mais on peut aussi les remplacer par T. C'est donc un assouplissement de AUT-68.

L'utilisation des deux formes de substitution n'apporte rien de plus sur le plan de la puissance du langage. Cependant, lors de l'écriture d'un livre en AUTOMATH, AUT-QE est plus facile à utiliser. C'est donc un aspect pragmatique important.

AUT-QE a été étudié par Van Daalen et réalisé par Zandleven. Jutting a entièrement traduit en AUT-QE le livre "Grundlagen der Analysis" de Landau.

3 : Réalisation de AUT-SL.

Définition abstraite du langage.

+ Syntaxe

Les variables sont des identificateurs notés par d

la constante est τ

les symboles auxiliaires sont $[\ ]\ <\ >\ \in$

Les expressions sont engendrées par la formule

$$ex = [d \in ex]^* \begin{cases} \tau \\ d \\ <ex>ex \end{cases}$$

On note Q la partie $[d \in ex]^*$ et T la partie restante.

+ Sémantique

Elle est définie par les attributs fn et cat (noté τ). Ils correspondent respectivement à la forme normale et au type de l'expresssion.

fn(ex) : - si ex = $<a>[x \in c]d$

+ si $\tau(fn(a)) = fn(c)$ alors
  β-red : $fn(ex) = S(x \rightsquigarrow a)d$
+ sinon erreur : argument non applicable.

} §

- si ex = $[x \in a]<x>b$ et si $x \notin b$ alors
  η-red : $fn(ex) = b$
- sinon $fn(ex) = ex$.

τ(ex) : - si ex = QT alors τ(ex) = Qτ(T)

- si ex = τ alors $\tau(\tau) = \tau$
- si ex = d alors
  - + si d est définie par [dϵa] dans Q
    alors τ(d) = a
  - + sinon erreur : variable non définie.
- si ex = $\langle$a$\rangle$b alors
  - + si τ(b) = [xϵc]d alors
    - ° si τ(fn(a)) = fn(c) alors
      β-red τ(ex) = S(x$\sim$a)d
    - ° sinon erreur : argument non applicable.

  } §§

  - + sinon τ(ex) = τ(b).

Remarque : les définitions de fn et τ dépendent l'une de l'autre. fn dépend de τ pour le test §. Ce test peut être supprimé; il est redondant avec celui effectué dans la fonction τ en §§, car si b=[xϵc]e alors τ(b)=[xϵc]τ(e). On rend donc fn indépendant de τ. Lors du calcul des attributs on impose le calcul de fn avant τ .

Définition concrète du langage.

C'est une définition acceptable par le système SISGOF. Elle est équivalente à la définition abstraite.

+ Syntaxe

**** *PRODUCTIONS* *******

1	ΔDEF	::= ΔEX
2	ΔEX	::= ΔT
3	ΔEX	::= $\Delta H\ \Delta T$
4	ΔT	::= D
5	ΔH	::= $\Delta H\ \Delta HE$
6	ΔH	::= ΔHE
7	ΔHE	::= [$D \in \Delta EXT$]
8	ΔEXT	::= ΔEX
9	ΔT	::= < ΔEXA > ΔEXF
10	ΔEXA	::= ΔEX
11	ΔEXF	::= ΔEX
12	ΔT	::= T

Des symboles non-terminaux sont introduits pour otenir une grammaire de précédence.

+ Sémantique

On retrouve les attributs fn et cat.
Les attributs ex et atyp en sont les équivalents pour les sous-expressions de la chaine d'abstracteurs.
L'attribut ld mémorise les variables liées définies et permet de tester l'unicité de leur définition.
L'attribut ty sert à la synchronisation, il permet le calcul du type de l'expression, après l'avoir mise sous forme normale et avoir trouvé le type de toutes ses variables liées.

Règles sémantiques

```
***** POUR LA REGLE *****
  1          ΔDEF ::= ΔEX
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔDEF ← FN.ΔEX
CAT.ΔDEF ← CAT.ΔEX
LD.ΔDEF ← LD.ΔEX
TY.ΔEX ← APRES ( FN.ΔDEF )

***** POUR LA REGLE *****
  2          ΔEX ::= ΔT
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔEX ← FN.ΔT
CAT.ΔEX ← CAT.ΔT
LD.ΔEX ← LD.ΔT
TY.ΔT ← TY.ΔEX

***** POUR LA REGLE *****
  3          ΔEX ::= ΔH ΔT
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔEX ← ( EX.ΔH ) ETARED ( FN.ΔT )
CAT.ΔEX ← ATYP.ΔH , ( EX.ΔH ) ETATYP ( CAT.ΔT )
LD.ΔEX ← ( LD.ΔH ) UNIQ ( LD.ΔT )
TY.ΔT ← TY.ΔEX

***** POUR LA REGLE *****
  4          ΔT ::= D
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔT ← NOM
CAT.ΔT ← TY.ΔT , CATE ( NOM )
LD.ΔT ← VIDE

***** POUR LA REGLE *****
  5          ΔH ::= ΔH ΔHE
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔH ← EX.ΔH" , EX.ΔHE
ATYP.ΔH ← ATYP.ΔH" , ATYP.ΔHE
LD.ΔH ← ( LD.ΔH" ) UNIQ ( LD.ΔHE )

***** POUR LA REGLE *****
  6          ΔH ::= ΔHE
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔH ← EX.ΔHE
ATYP.ΔH ← ATYP.ΔHE
LD.ΔH ← LD.ΔHE
```

```
***** POUR LA REGLE *****
  7          ΔHE ::= [ D ∈ ΔEXT ]
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔHE ← ( NOM ) HEAD ( FN.ΔEXT )
ATYP.ΔHE ← ( NOM ) FMTYP ( FN.ΔEXT ) , APRES ( CAT.ΔEXT )
LD.ΔHE ← ( LD.ΔEXT ) UNIQ ( NOM )

***** POUR LA REGLE *****
  8          ΔEXT ::= ΔEX
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔEXT ← FN.ΔEX
CAT.ΔEXT ← CAT.ΔEX
LD.ΔEXT ← LD.ΔEX
TY.ΔEX ← APRES ( FN.ΔEXT )

***** POUR LA REGLE *****
  9          ΔT ::= < ΔEXA > ΔEXF
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔT ← ( FN.ΔEXA ) BETARED ( FN.ΔEXF )
CAT.ΔT ← ( CAT.ΔEXA ) BETATYP ( FN.ΔEXA ) BETA1 ( CAT.ΔEXF )
       BETA2 ( FN.ΔEXF )
LD.ΔT ← ( LD.ΔEXA ) UNIQ ( LD.ΔEXF )
TY.ΔEXA ← TY.ΔT
TY.ΔEXF ← TY.ΔT

***** POUR LA REGLE *****
 10          ΔEXA ::= ΔEX
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔEXA ← FN.ΔEX
CAT.ΔEXA ← CAT.ΔEX
LD.ΔEXA ← LD.ΔEX
TY.ΔEX ← TY.ΔEXA

***** POUR LA REGLE *****
 11          ΔEXF ::= ΔEX
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔEXF ← FN.ΔEX
CAT.ΔEXF ← CAT.ΔEX
LD.ΔEXF ← LD.ΔEX
TY.ΔEX ← TY.ΔEXA

***** POUR LA REGLE *****
 12          ΔT ::= T
°°°° LA SEMANTIQUE INTRODUITE EST :
FN.ΔT ← 0
CAT.ΔT ← 0
LD.ΔT ← VIDE
```

Le détail des fonctions sémantiques utilisées se trouve en annexe 2.

Voici maintemant quelques exemples d'expressions de AUT-SL.

```
                                LIGNE LUE :      T

Expression de deg  0            ... TEMPS ANALYSE LEXICALE :0 0 3
                                ooo TEMPS ANALYSE SYNTAXIQ :0 0 16
          de valence 1
                                EXPRESSION : T
          d'ordre 0
                                TYPE       : T

                                ooo TEMPS ANALYSE SEMANTIQ :0 0 47

                                LIGNE LUE :      [XєNAT]T

                                ... TEMPS ANALYSE LEXICALE :0 0 20
                                ooo TEMPS ANALYSE SYNTAXIQ :0 0 44
Expression de deg 0             ATTENTION :  NAT A POUR TYPE  T
de valence 1 et d'ordre 1       EXPRESSION : [XєNAT]T
NAT à par défaut le type        TYPE       : [XєNAT]T

                                ooo TEMPS ANALYSE SEMANTIQ :0 2 33

                                LIGNE LUE :      [Fє[XєNAT]T]T

                                ... TEMPS ANALYSE LEXICALE :0 0 34
                                ooo TEMPS ANALYSE SYNTAXIQ :0 1 11
Expression de deg 0             ATTENTION :  NAT A POUR TYPE  T
de valence 1 et d'ordre 2       EXPRESSION : [Fє[XєNAT]T]T
F est une fonction des          TYPE       : [Fє[XєNAT]T]T
entiers vers les entiers.       ooo TEMPS ANALYSE SEMANTIQ :0 4 16

                                LIGNE LUE :      [XєNAT]X

Expression de deg 2             ... TEMPS ANALYSE LEXICALE :0 0 24
                                ooo TEMPS ANALYSE SYNTAXIQ :0 0 44
de valence 1 et d'ordre 1       ATTENTION :  NAT A POUR TYPE  T
Soit X une variable de          EXPRESSION : [XєNAT]X
type NAT.                       TYPE       : [XєNAT]NAT

                                ooo TEMPS ANALYSE SEMANTIQ :0 2 44

Expression de deg 2             LIGNE LUE :      [Fє[XєNAT]NAT]F
de valence 1 et d'ordre 2       ... TEMPS ANALYSE LEXICALE :0 0 40
F est une fonction.             ooo TEMPS ANALYSE SYNTAXIQ :0 1 15
                                EXPRESSION : [Fє[XєNAT]NAT]F
                                TYPE       : [Fє[XєNAT]NAT][XєNAT]NAT

                                ooo TEMPS ANALYSE SEMANTIQ :0 4 47
```

Exemple de substitution D1 ~ Q et D2 ~ P

```
LIGNE LUE :      <P><Q>[D1∈T][D2∈T][D3∈D2]D1

... TEMPS ANALYSE LEXICALE :0 1 12
ooo TEMPS ANALYSE SYNTAXIQ :0 2 52
ATTENTION :  P A POUR TYPE  T
ATTENTION :  Q A POUR TYPE  T

EXPRESSION : [D3∈P]Q

TYPE       : [D3∈P]T

ooo TEMPS ANALYSE SEMANTIQ :0 10 55
```

Règle du modus-ponens

Soit H1 de type P et soit H2 de type P Q

alors ⟨H1⟩H2 est de type Q.

```
LIGNE LUE :      [H1∈P][H2∈[X∈P]Q]<H1>H2

... TEMPS ANALYSE LEXICALE :0 1 8
ooo TEMPS ANALYSE SYNTAXIQ :0 2 14
ETA REDUCTION POSSIBLE
1
ATTENTION :  P A POUR TYPE  T
ATTENTION :  P A POUR TYPE  T
ATTENTION :  Q A POUR TYPE  T

EXPRESSION : [H1∈P][H2∈[X∈P]Q]<H1>H2

TYPE       : [H1∈P][H2∈[X∈P]Q]Q

ooo TEMPS ANALYSE SEMANTIQ :0 9 14
```

La chaine d'abstracteurs contient les hypothèses, ⟨H1⟩H2 est la règle à appliquer pour obtenir le resultat Q.

Norme d'une fonction F de Nat vers Nat.

```
LIGNE LUE :      [Nє[Fє[XєNAT]NAT]NAT]<F>N

... TEMPS ANALYSE LEXICALE :0 1 7
ooo TEMPS ANALYSE SYNTAXIQ :0 2 15
EXPRESSION : [Nє[Fє[XєNAT]NAT]NAT]<F>N

TYPE       : [Nє[Fє[XєNAT]NAT]NAT]NAT

ooo TEMPS ANALYSE SEMANTIQ :0 9 14
```

N est une application de l'ensemble des fonctions F dans l'ensemble des entiers.

C'est une expression de degré 2 et d'ordre 3.

Fonction dyadique D : $(x,y) \xrightarrow{D} z$; x,y,z NAT

```
LIGNE LUE :      [Dє[XєNAT][YєNAT]NAT]<X><Y>D

... TEMPS ANALYSE LEXICALE :0 1 17
ooo TEMPS ANALYSE SYNTAXIQ :0 2 47

EXPRESSION : [Dє[XєNAT][YєNAT]NAT]<X><Y>D

TYPE       : [Dє[XєNAT][YєNAT]NAT]NAT

ooo TEMPS ANALYSE SEMANTIQ :0 12 5
```

C'est une expression de degré 2 et d'ordre 3.

Produit scalaire de deux fonctions F et G

```
LIGNE LUE :      [PSє[Fє[XєNAT]NAT][Gє[YєNAT]NAT]NAT]<F><G>PS

... TEMPS ANALYSE LEXICALE :0 1 52
ooo TEMPS ANALYSE SYNTAXIQ :0 3 47

EXPRESSION : [PSє[Fє[XєNAT]NAT][Gє[YєNAT]NAT]NAT]<F><G>PS

TYPE       : [PSє[Fє[XєNAT]NAT][Gє[YєNAT]NAT]NAT]NAT

ooo TEMPS ANALYSE SEMANTIQ :0 17 26
```

C'est une expression de degré 2 et d'ordre 3.

Exemples d'erreurs.

```
LIGNE LUE :      [XϵT]<A>

... TEMPS ANALYSE LEXICALE :0 0 24
[DϵT]<D>
          ^
ERREUR DE SYNTAXE
ooo TEMPS ANALYSE SYNTAXIQ :0 0 49
```

```
LIGNE LUE :      [XϵT][XϵX]T

... TEMPS ANALYSE LEXICALE :0 0 35
ooo TEMPS ANALYSE SYNTAXIQ :0 1 12
VARIABLE LIEE  DEFINIE DEUX FOIS
OOO TEMPS ANALYSE SEMANTIQ :0 3 22
```

```
LIGNE LUE :      <X>[YϵX]T

... TEMPS ANALYSE LEXICALE :0 0 30
ooo TEMPS ANALYSE SYNTAXIQ :0 1 18
ATTENTION :  X A POUR TYPE  T
ATTENTION :  X A POUR TYPE  T
DANS LA BETA-REDUCTION LES TYPES NE SONT PAS COMPATIBLES?
ERREUR FONCTION IS
   QUE FAIRE ? :  EDEX A
X
   QUE FAIRE ? :  EDEX TY
T
   QUE FAIRE ? :
OOO TEMPS ANALYSE SEMANTIQ :0 3 48
```

Actions en cas d'erreur.

```
      ΔERR
ERR
'VARIABLE LIEE  DEFINIE DEUX FOIS'
'CODE 11 LIBRE'
'CODE 12 LIBRE'
'CODE 13 LIBRE'
'DANS LA BETA-REDUCTION LES TYPES NE SONT PAS COMPATIBLES'
ERSY
'CODE 16 LIBRE'
'CODE 17 LIBRE'
ERSY
'CODE 19 LIBRE'
'CODE 20 LIBRE'
'CODE 21 LIBRE'
'CODE 22 LIBRE'
ERLX
```

4 : Réalisation de LAMBDA.

Lambda est entièrement basé sur AUT-SL pour ce qui concerne le niveau expression. Il reprend les idées de PAL pour les niveaux livre et ligne.

Le principe est d'utiliser l'indicateur de contexte pour réduire la longueur des expressions introduites. La mise en évidence du contexte présente l'intérêt d'éclater la démonstration en plusieurs parties : hypothèses,lemmes et théorème.

Le second intérêt est d'éviter la duplication de la chaine d'abstracteurs dans les zones démonstration et énoncé.

La possibilité est aussi donnée d'introduire des axiomes ou constantes primitives. C'est une extension indispensable, mais sans grande influence sur les mécanismes de base.

La principale difficulté supplémentaire dans LAMBDA est due à l'existance de la δ-réduction, c'est-à-dire, le remplacement d'un objet par sa définition.

Dans la réalisation effectuée, nous avons systématiquement effectué cette δ-réduction ainsi que la mise sous forme normale des expressions. Cette approche simplifie la procédure de décisi de l'égalité de deux termes. En annexe, se trouvent les détails de la réalisation n'étant pas déjà décrits dans AUT-SL.

Voyons plutôt l'utilisation de LAMBDA et des langages AUTOMAT pour écrire des livres mathématiques.

Pour l'implication, il n'y a pas de problème, c'est l'abstractio Son élimination est la règle du modus-ponens. En introduisant un constante primitive CON pour la contradiction, on peut définir l

négation. Suivant cette définition, on obtient le calcul propositionnel classique ou intuitioniste.

0	A	EB	$\top$	Explications.
A	B	EB	$\top$	
B	IMP	$[x\in A]B$	$[x\in A]\top$	$A \rightarrow B$
0	CON	PN	$\top$	
A	NOT	$<CON><A>IMP$	$[x\in A]\top$	$NOT(A) = A \rightarrow CON$
A	H1	EB	A	
H1	H2	EB	$<A>NOT$	
H2	NOTOUT	$<H1>H2$	CON	$A, NOT(A) \vdash CON$
H1	H3	EB	CON	
H3	NOTIN	$[x\in A]H2$	$<A>NOT$	$"A", CON \vdash NOT(A)$

intuitioniste

A	I	PN	$<A><CON>IMP$	$\vdash CON \rightarrow A$

classique

A	C0	EB	$<A>NOT$	
C0	C	PN	$<A><CON>IMP$	$"NOT(A)" \vdash CON \rightarrow A$

Voyons maintemant l'introduction de $\forall$, ainsi nous aurons le calcul des prédicats du 1° ordre.

0	A	EB	$\top$
A	P	EB	$[x\in A]\top$
P	NOTP	$<CON><P>IMP$	$[x\in P]\top$
A	B	EB	$[x\in A]\top$
B	ALL	$[x\in A]B$	$[x\in A][y\in A]\top$
B	some	$<<<B>NOTP><A>ALL>NOT$	$[y\in <<B>NOTP><A>ALL]\top$

$\forall$- introduction et élimination. Explications

0	X	EB	τ	
X	P	EB	$[u \in X]\tau$	
P	H1	EB	P	
H1	ALLIN	$[u \in X]$H1	<P><X>ALL	$P(X) \vdash \forall X.P(X)$
P	H2	EB	<P><X>ALL	
H2	Y	EB	X	
Y	ALLOUT	<Y><P><X>H2	P	$\forall X.P(X) \vdash P(Y)$

Ensuite, l'on peut définir l'égalité, avec ses propriétés : réflexivité, symétrie, transitivité...
puis les entiers et les axiomes de Peano :

0	1	PN	NAT
0	S	PN	$[T \in NAT]NAT$
0	AX3	PN	$[T \in NAT]<T>S \neq 1$
0	AX4	PN	$[T \in NAT][U \in NAT](S(T)=S(U)) \rightarrow T=U$

...

Les pages suivantes contiennent un exemple de texte en LAMBDA. C'est la traduction du début du texte en AUT-QE qui se trouve dans le rapport APLASM. Ce texte de Jutting explique de façon détaillée comment passer d'un texte mathématique, au texte AUTOMATH correspondant. Il l'applique à la démonstration du théorème suivant :

> Si f est une injection de $[1,n]$ dans lui-même,
> alors f est une surjection.

LIVRE

1	0	*A*	*EB* T
2	*A*	*B*	*EB* T
3	*B*	*IMP*	[*X*∈*A*]*B* T
4	0	*CON*	*PN* T
5	*A*	*NOT*	<*CON*><*A*>*IMP* T
6	*B*	*I*	*EB* <*B*><*A*>*IMP*
7	*I*	*N*	*EB* <*B*>*NOT*
8	*N*	*CONTR*	[*X*∈*A*]<<*X*>*I*>*N* <*A*>*NOT*
9	*A*	*A*0	*EB* *A*
10	*A*0	*TH*1	[*T*∈<*A*>*NOT*]<*A*0>*T* <<*A*>*NOT*>*NOT*
11	*A*	*N*1	*EB* <<*A*>*NOT*>*NOT*
12	*N*1	*DBLNE*	*PN* *A*
13	*I*	*J*	*EB* <*B*><<*A*>*NOT*>*IMP*
14	*J*	*ANYCA*	<[*T*∈<*B*>*NOT*]<<<*T*><*I*><*B*><*A*>*CONTR*>*J*>*T*><*B*>*DBLNEG* *B*
15	*B*	*N*2	*EB* <*A*>*NOT*
16	*N*2	*TH*2	[*T*∈*A*]<[*U*∈<*B*>*NOT*]<*T*>*N*2><*B*>*DBLNEG* <*B*><*A*>*IMP*
17	*B*	*A*1	*EB* *A*
18	*A*1	*N*3	*EB* <*B*>*NOT*
19	*N*3	*TH*3	[*T*∈<*B*><*A*>*IMP*]<<*A*1>*T*>*N*3 <<*B*><*A*>*IMP*>*NOT*
20	*B*	*N*4	*EB* <<*B*><*A*>*IMP*>*NOT*
21	*N*4	*TH*4	<[*T*∈<*A*>*NOT*]<<*T*><*B*><*A*>*TH*2>*N*4><*A*>*DBLNEG* *A*
22	*N*4	*TH*5	[*T*∈*B*]<[*U*∈*A*]*T*>*N*4 <*B*>*NOT*
23	*B*	*OR*	<*B*><<*A*>*NOT*>*IMP* T
24	*A*1	*ORI*1	<<*A*1><*A*>*TH*1><*B*><<*A*>*NOT*>*TH*2 <*B*><*A*>*OR*
25	*B*	*B*0	*EB* *B*
26	*B*0	*ORI*2	[*T*∈<*A*>*NOT*]*B*0 <*B*><*A*>*OR*
27	*B*	*AU*	*EB* <*B*><*A*>*OR*
28	*AU*	*N*5	*EB* <*A*>*NOT*
29	*N*5	*NOTC*1	<*N*5>*AU* *B*
30	*AU*	*N*6	*EB* <*B*>*NOT*

Explications

Ligne

3 définition de IMP implication logique

4 introduction de la constante primitive CON

pour la contradiction, elle permet la définition

5 de la négation NOT par si A alors NOT(A) → CON

6 soit | A → B (I)

7 soit | NOT(B) (N)

8 alors | NOT(A) (CONTR)

9 soit | A

10 alors | NOT(NOT(A)) (TH1)

11 soit | NOT(NOT(A))

12 alors | A axiome de la double négation

6 soit | A → B (I)

13 soit | (NOT(A)) → B (J)

14 alors | B démontré ainsi :

1 | A → B (I)

2 | not(A) → B (J)

3 | | not(B) par [t∈<B>NOT]

4 | | not(A) par 8(1,3)

5 | | B par 4 appliqué à 2(J)

6 | not(B) → CON <B>t

et | par définition c'est not(not(B))

d'ou 7 | B par la double négation.

Extrait de l'introduction des lignes.

```
 LIGNE 22
N4 TH5 [TєB]<[UєA]T>N4  <B>NOT

EXPRESSION : [DUM11єB]CON

TYPE       : T

EXPRESSION : [DUM1єB]<[DUM2єA]DUM1>N4

TYPE       : [DUM1єB]CON

*****          N4     TH5    [TєB]<[UєA]T>N4  <B>NOT
 LIGNE 23
B OR <B><<A>NOT>IMP T

EXPRESSION : [DUM21є[DUM11єA]CON]B

TYPE       : T

*****          B      OR     <B><<A>NOT>IMP  T
 LIGNE 24
A1 ORI1 <<<A1><A>TH1><B><<A>NT
                              v
                              OT>>TH2  <B><A>OR

EXPRESSION : [DUM31є[DUM21єA]CON]B

TYPE       : T

<<<D><D>D><D><<D>D>>D
                    ^
ERREUR DE SYNTAXE
```

demande d'introduction de la ligne 22

ligne introduite

résultat de l'analyse de la zone CAT

on fournit la forme normale et le type

résultat de l'analyse de la zone DEF

ligne acceptée.

correction de la ligne lors de l'introduction

zone DEF

TH2 à trois paramètres et non un.

La forme est <a><b><c>TH2

```
<<<A1><A>TH1><B><<A>NOT>>TH2
TAPEZ UNE NOUVELLE DEFINITION

<<A1>-
     <A>TH1><B><<A>NOT>TH2
     v
```

introduction de la modification

```
EXPRESSION : [DUM31є[DUM21єA]CON]<[DUM32є[DUM43єB]CON]<DUM31>[DUM11є[DUM22єA]CON]
                                                          <A1>DUM11><B>DBLNE
TYPE       : [DUM11є[DUM21єA]CON]B

*****          A1     ORI1   <<A1><A>TH1><B><<A>NOT>TH2  <B><A>OR
```

Table de la forme normale des zones CAT du livre

```
      TCAT LFICH 'TCAT'

[1]   T
[2]   T
[3]   T
[4]   T
[5]   T
[6]   [DUM1∈A]B
[7]   [DUM1∈B]CON
[8]   [DUM1∈A]CON
[9]   A
[10]   [DUM21∈[DUM11∈A]CON]CON
[11]   [DUM21∈[DUM11∈A]CON]CON
[12]   A
[13]   [DUM21∈[DUM11∈A]CON]B
[14]   B
[15]   [DUM11∈A]CON
[16]   [DUM11∈A]B
[17]   A
[18]   [DUM11∈B]CON
[19]   [DUM21∈[DUM11∈A]B]CON
[20]   [DUM21∈[DUM11∈A]B]CON
[21]   A
[22]   [DUM11∈B]CON
[23]   T
[24]   [DUM31∈[DUM21∈A]CON]B
[25]   B
[26]   [DUM31∈[DUM21∈A]CON]B
[27]   [DUM31∈[DUM21∈A]CON]B
[28]   [DUM11∈A]CON
[29]   B
[30]   [DUM11∈B]CON
```

Table des identificateurs utilisés

```
   TYPE   NOM

[1]  V      A
[2]  V      B
[3]  C      IMP
[4]  C      CON
[5]  C      NOT
[6]  V      I
[7]  V      N
[8]  C      CONTR
[9]  V      A0
[10] C      TH1
[11] V      N1
[12] C      DBLNE
[13] V      J
[14] C      ANYCA
[15] V      N2
[16] C      TH2
[17] V      A1
[18] V      N3
[19] C      TH3
[20] V      N4
[21] C      TH4
[22] C      TH5
[23] C      OR
[24] C      ORI1
[25] V      B0
[26] C      ORI2
[27] V      AU
[28] V      N5
[29] C      NOTC1
[30] V      N6
```

correspondance position table/ligne du livre

```
      LDEF/⍳⍴LDEF
3 5 8 10 14 16 19 21 22 23 24 26 29
```

Table de la forme normale des zones DEF du livre

```
      TEXP LFICH'TEXP'
[1]   [DUM1∈A]B
[2]   [DUM1∈A]CON
[3]   [DUM1∈A]<<DUM1>I>N
[4]   [DUM1∈[DUM12∈A]CON]<A0>DUM1
[5]   <[DUM1∈[DUM12∈B]CON]<<[DUM21∈A]<<DUM21>I>DUM1>J>DUM1><B>DBLNE
[6]   [DUM1∈A]<[DUM2∈[DUM13∈B]CON]<DUM1>N2><B>DBLNE
[7]   [DUM1∈[DUM12∈A]B]<<A1>DUM1>N3
[8]   <[DUM1∈[DUM12∈A]CON]<[DUM21∈A]<[DUM22∈[DUM33∈B]CON]<DUM21>DUM1><B>DBLNE>N4><A>DBLNE
[9]   [DUM1∈B]<[DUM2∈A]DUM1>N4
[10]   [DUM21∈[DUM11∈A]CON]B
[11]   [DUM31∈[DUM21∈A]CON]<[DUM32∈[DUM43∈B]CON]<DUM31>[DUM11∈[DUM22∈A]CON]<A1>DUM11><B>DBL
[12]   [DUM1∈[DUM12∈A]CON]B0
[13]   <N5>AU
```

Conclusion.

L'étude entreprise montre bien le double intérêt du projet AUTOMATH.

Le premier est de fournir une aide au mathématicien. Les perspectives dans ce domaine sont très prometteuses. Avec le perfectionnement sans cesse croissant des machines, dans un avenir proche, le mathématicien, au lieu d'utiliser le tableau noir, utilisera un terminal d'ordinateur. Celui-ci aura l'avantage de l'avertir immédiatement de ses erreurs dans les démonstrations. Un tel système pourrait être conçu comme une machine ayant AUTOMATH comme langage de base. Il pourra comporter des bibliothèques de livres mathématiques. Il s'agira alors, partant d'un livre de géométrie, par exemple, de le continuer en introduisant de nouveaux théorèmes qui pourront être exprimés dans un langage approprié à la géométrie. Ce langage sera lui même traduit en AUTOMATH, de la même façon que les langages informatiques évolués sont traduits en langage machine.

Le deuxième aspect est d'unifier les mathématiques sur une même base. Ce qui permettra de comparer différentes théories. De plus, ce travail présente un intérêt mathématique en lui même. En effet, la théorie de la démonstration est une branche des mathématiques. Cette approche permet d'apporter une contribution importante à cette théorie encore assez jeune.

BIBLIOGRAPHIE.

Ouvrages généraux sur la logique.

KLEENE Logique mathématique Collection U
Armand Colin

HACKSTAFF Systems of formal logic
Reigel Dordrecht Holland 1966

Sur les systèmes de Gentzen

GRIZE Logique moderne Vol 1
Gauthier Villar

PRAWITZ Natural deduction.
Almquist et Wiksell Stockolm.

Sur le lambda-calcul.

CURRY & FEY Combinatory logic
North Holland.

STENLUND Combinators,lambda-terms and proof theory
Reidel Dordrecht Holland

Sur AUTOMATH.

DE BRUIJN The mathematical langage AUTOMATH
Symposium on automatic demonstration
Versailles 1968. Lecture notes in
mathematics Vol 125 .
Springer Verlag. Berlin 1970.

APLASM 73 Le projet AUTOMATH (vol 1)
Publication de mathématique
Université PARIS-SUD 91405 ORSAY FRANCE.

KIREMITDJIAN LIMA-PAL
Note ECSTASM N°8 PARIS-SUD.

NEDERPELT Strong normalization in a type lambda calcul
with lambda structured types.
Doctoral dissertation 1972
Technological University Eindhoven.

Annexe 1

PAL

```
***** POUR LA REGLE *****
  1          ΔDEF ::= ΔEXP
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔDEF ← EX.ΔEXP
CAT.ΔDEF ← CAT.ΔEXP
CNTX.ΔDEF ← CONTEXTE (CNTX. ΔEXP )
TY.ΔDEF ← TY.ΔEXP
***** POUR LA REGLE *****
  2          ΔEXP ::= C
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔEXP ← LN.ΔEXP , NOM , ADDZ ( CNTX. ΔEXP )
CAT.ΔEXP ← TIP
CNTX.ΔEXP ← LI ( NOM )
LN.ΔEXP ← ρ CNTX.ΔEXP
TY.ΔEXP ← 0
***** POUR LA REGLE *****
  3          ΔEXP ::= V
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔEXP ← 0 , NOM
CAT.ΔEXP ← TIP
CNTX.ΔEXP ← NOM
LN.ΔEXP ← 0
TY.ΔEXP ← 1
***** POUR LA REGLE *****
  4          ΔEXP ::= C ( ΔLEXP )
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔEXP ← ( ρ LI ( NOM ) ) , NOM , ( ADDZ ( ( - LN.ΔLEXP ) +
CAT.ΔEXP ← ( EX.ΔLEXP ) SUB ( NOM )                    LI ( NOM ) ) ) , EX.ΔLEXP
CNTX.ΔEXP ← CNTX.ΔLEXP
LN.ΔEXP ← ( LN.ΔLEXP ) LONG ( NOM )
TY.ΔEXP ← 1
***** POUR LA REGLE *****
  5          ΔLEXP ::= ΔEXP
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔLEXP ← EX.ΔEXP
CAT.ΔLEXP ← CAT.ΔEXP
CNTX.ΔLEXP ← CNTX.ΔEXP
LN.ΔLEXP ← 1
***** POUR LA REGLE *****
  6          ΔLEXP ::= ΔLEXP , ΔEXP
°°°° LA SEMANTIQUE INTRODUITE EST :
EX.ΔLEXP ← EX.ΔLEXP" , EX.ΔEXP
CAT.ΔLEXP ← CAT.ΔLEXP" , CAT.ΔEXP
CNTX.ΔLEXP ← CNTX.ΔLEXP" , CNTX.ΔEXP
LN.ΔLEXP ← LN.ΔLEXP" + 1
```

LE 6/10/1975 A 14H 58MN

```
    ∇ Z←ADDZ X
[1]   Z←,0,[1.5] X
    ∇

    ∇ Z←CONTEXTE X
[1]   ⍵←X
[2]   →0 IF 1=VAL←X INCLU LI N
[3]   12 ERREUR 'CONTEXTE'
    ∇

    ∇ V←POS GET PILE
[1]   PILE←'Δ',PILE
[2]   ⍎'V←',PILE,'[POS+⍳',PILE,'[POS]]'
    ∇

    ∇ Z←A INCLU B
[1]   Z←∧/(A∈B)
    ∇

    ∇ Z←A IS B;J;AJ;BJ;PA;PB
[1]   →((⍴A)≠⍴B)/S1
[2]   →(∧/A=B)/FIN
[3]  S1:→F IF(∧/A=0)∨∧/B=0
[4]   →S2 IF A[2]=B[2]
[5]   →SA IF(⍴A)<⍴B
[6]   →F IF 0=LDEF[B[2]]
[7]   B←RED B
[8]   →(Z←A IS B)/0
[9]  SA:→F IF 0=LDEF[A[2]]
[10]  A←RED A
[11]  →(Z←A IS B)/0
[12] S2:J←Z←1
[13]  PA←2↓A
[14]  PB←2↓B
[15] B1:→S3 IF A[1]<J
[16]  AJ←TAKE PA
[17]  PA←(⍴AJ)↓PA
[18]  BJ←TAKE PB
[19]  PB←(⍴BJ)↓PB
[20]  →0 IF 0=Z←Z∧AJ IS BJ
[21]  →B1,J←J+1
[22] S3:→0
[23] FIN:Z←1
[24]  →0
[25] F:Z←0
[26]  10 ERREUR 'IS'
[27]  →0
    ∇

    ∇ Z←LI N;R
[1]   Z←0⍴1
[2]   →(0=⍴,N)/0
```

```
[3]    →(0=R←ARB[N])/0
[4]    Z←(LI R),R
       ∇


       ∇ Z←NOM
[1]    Z←,ΔVAL[V[1];1]
       ∇


       ∇ POS←V PUT PILE
[1]    POS←1+⍴⍎PILE←'Δ',PILE
[2]    ⍎PILE,'←',PILE,',(⍴,V),V'
       ∇


       ∇ Z←RED E;C;L;P;ΔSBS
[1]    Z←E
[2]    →0 IF 0=LDEF[C←E[2]]
[3]    ΔSBS←LI C
[4]    L←2↓E
[5]    P←TEXP[+/C↑LDEF]
[6]    Z←L SUBS P GET 'TEXP'
[7]    →0
[8]    ΔTEXP
       ∇


       ∇ Z←A SUB B;ΔSBS
[1]    ΔSBS←LI B
[2]    Z←A SUBS TCAT[B] GET 'TCAT'
[3]    VAL←ΔSBS TCKCAT A
       ∇


       ∇ B←A SUBS C;AI;I;POS;S;PILOT
[1]    →0 IF∧/0=B←C
[2]    I←0
[3]    PILOT←ΔSBS∘.=B×¯1⌽B=0
[4]    S←⍴ΔSBS
[5]   S1:→0 IF S<I←I+1
[6]    →0 IF 0=⍴AI←TAKE A
[7]    A←(⍴AI)↓A
[8]   S2:→S1 IF(⍴B)<POS←PILOT[I;]⍳1
[9]    B←((POS-2)↑B),AI,POS↓B
[10]   PILOT[I;POS]←0
[11]   PILOT←((S,(POS-2))↑PILOT),((S,(⍴AI))⍴0),(0,POS)↓PILOT
[12]   →S2
[13]  ER:16 ERREUR 'SUBSTITUTION'
[14]   →0
[15]   ΔTCAT
      ∇


       ∇ Z←TAKE X;I
[1]    →0 IF 0=⍴Z←,X
[2]    I←2
```

```
[3]    X←1↑X
[4]   B:→FIN IF 0=X
[5]    I←I+⍴,TAKE I↓Z
[6]    X←X-1
[7]    →B
[8]   FIN:Z←I↑Z
      ∇


      ∇ Z←ΔSBS TCKCAT LA;I;CAT;EXP;EI;L;A;CATE;SBS
[1]    →0 IF 0=I←⍴ΔSBS
[2]    A←LA
[3]    EXP←⍳0
[4]   B:→S1 IF 0=L←⍴EI←TAKE A
[5]    EXP←EXP,EI[2]
[6]    A←L↓A
[7]    →B
[8]   S1:I←I⌊⍴EXP
[9]    CAT←TCAT[ΔSBS[I]] GET 'TCAT'
[10]   CATE←TCAT[EXP[I]] GET 'TCAT'
[11]   ΔSBS←(I←I-1)↑ΔSBS
[12]   →F IF~Z←CATE IS LA SUBS CAT
[13]   →S1 IF 0<I
[14]   →0
[15]  F:11 ERREUR 'TCKCAT'
      ∇


      ∇ Z←TIP
[1]    Z←TCAT[NOM] GET 'TCAT'
      ∇


      ∇ Z←TRADUC X;NOM;PT;J
[1]    PT←1
[2]    DUM←''
[3]    Z← 2 0 ⍴0
[4]   S1:→0 IF PT>⍴,X
[5]    →S2 IF 0=NOM←ENCODE SEP SCAN X
[6]    J←DICO⍳NOM
[7]    →ER IF J>⍴DICO
[8]    Z←Z, 2 1 ⍴(VCT['VC'⍳TYPE[J]]),J
[9]   S2:→0 IF PT>⍴,X
[10]   Z←Z, 2 1 ⍴(VT[SEP⍳X[PT]]),0
[11]   PT←PT+1
[12]   →S1
[13]  ER:J←DUM⍳NOM
[14]   →S3 IF J≤⍴DUM
[15]   DUM←DUM,NOM
[16]  S3:Z←Z, 2 1 ⍴(VCT[3]),-J
[17]   →S2
      ∇


      ∇ Z←VALID EXP;ΔVAL;ΔSAT;VAL;DUM
[1]    VAL←1
[2]    Z←EXEC ΔEXP EXP
```

```
[3]    →0 IF Z
[4]    14 ERREUR 'VALID'
     ∇
```

```
      ΔERR
ERR
```

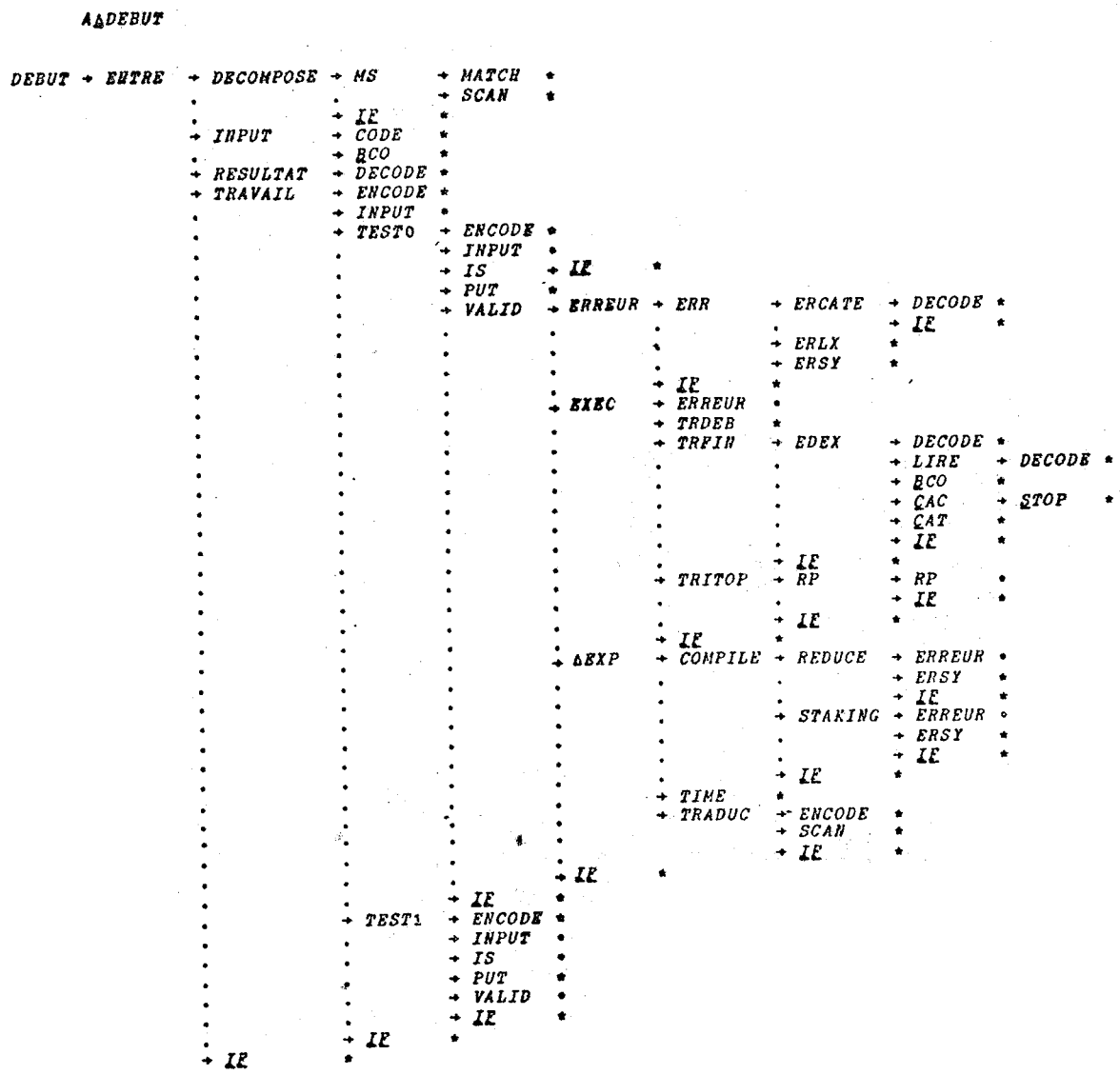
```
'DEUX EXPRESSIONS NE SONT PAS D-EGALES'

'LE PARAMETRE ',(⍕I+1),' DE ',(,DECODE DICO[X1]),' PAS COMPA
      TIBLE AVEC ',,DECODE DICO[SBS[I+1]]
'ERREUR DE CONTEXTE '

'TROP DE PARAMETRE POUR ',,DECODE DICO[X1]

'L''EXPRESSION SUIVANTE N''EST PAS VALIDE'

ERSY

'SUBSTITUTION'

'ERREUR DANS LA RECHERCHE DU PERE POUR LA DECORATION DE ARBR
     E'
ERSY

'NON DEFINIS : ',,DECODE(ISDUM=0)/DUM

'DOUBLE DEFINITION DE VAR LIEE : ',,DECODE DUM[I]
```

A LISTE DU GROUPE DE <PROCEDURES SEMANTIQUES> PAGE 1

LE 6/10/1975 A 13H 44MN

```
     ∇ Z←APRES X
[1]    Z←''
     ∇


     ∇ Z←A BETARED B;BB
[1]   DEB:→S1 IF SYMBNB[1]≠1↑B
[2]    ΔSBS←,B[2]
[3]    Z←(,A) SUBS,(SYMBNB[2 1] FRST B)↓B
[4]    →0
[5]   S1:B←RED BB←B
[6]    →DEB IF∨/B≠(ρB)↑BB
[7]    Z←SYMBNB[3],A,SYMBNB[4],B
     ∇


     ∇ Z←A BETATYP B;TY;BB
[1]   DEB:→S1 IF SYMBNB[1]≠1↑B
[2]    TY←¯1↓2↓(SYMBNB[2 1] FRST B)↑B
[3]    Z←(SYMBNB[2 1] FRST B)↓B
[4]    →ER IF~TY IS A
[5]    →0
[6]   ER:14 ERREUR 'BETATYP'
[7]    →0
[8]   S1:B←RED BB←B
[9]    →DEB IF∨/B≠(ρB)↑BB
[10]   Z←,B
     ∇


     ∇ Z←A BETA1 B
[1]    →S1 IF SYMBNB[1]≠1↑B
[2]    ΔSBS←,B[2]
[3]    Z←(,A) SUBS,B
[4]    →0
[5]   S1:Z←B
     ∇


     ∇ Z←CATE A
[1]    →S1 IF 2≠⎕NC Z←'TY',⍕A
[2]    Z←⍎Z
[3]    →0
[4]   S1:Z←0
[5]    'ATTENTION :  ',(LIRE A),' A POUR TYPE  ⍵ '


     ∇ Z←EDEX X
[1]    I←ρZ←''
[2]   B1:→F1 IF(ρX←,X)<I←I+1
[3]    →(D1,D2,D3)[2+×X[I]]
[4]   D1:→B1,ρZ←Z CAT(,DECODE DUM[⌊X[I]]),' ∈'[1+SYMBNB[1]=X[
       1⌈I-1]]
[5]   D2:→B1,ρZ←Z CAT '⍵'
[6]   D3:→D4 IF X[I]∈SYMBNB
[7]    →B1,ρZ←Z CAT(LIRE X[I]),' ∈'[1+SYMBNB[1]=X[1⌈I-1]]
```

```
[8]  D4:→B1,⍴Z←Z CAT SYMBCR[SYMBNB⍳X[I]]
[9]  F1:Z←(Z≠' ')/Z←,Z CAC ' '
     ∇


     ∇ Z←A ETARED B;LA;LD;LDA
[1]    LD←(¯1⌽A=SYMBNB[1])/A
[2]    LA←(¯1⌽B=SYMBNB[3])/B
[3]    LDA←LA INTER LD
[4]    →S1 IF 0=⍴LDA
[5]    'ETA REDUCTION POSSIBLE'
[6]    LDA
[7]   S1:Z←A,B
     ∇


     ∇ Z←A ETATYP B
[1]    Z←A,B
     ∇


     ∇ Z←A FMTYP B
[1]    Z←''
[2]    ⍎'TY',(⍕A),'←B'
     ∇


     ∇ Z←A FRST B;P
[1]    P←A∘.=B
[2]    P[1;]←-P[1;]
[3]    P←+\+⌿P
[4]    Z←P⍳0
     ∇


     ∇ Z←A HEAD B
[1]    Z←SYMBNB[1],A,B,SYMBNB[2]
     ∇


     ∇ Z←NOM
[1]    Z←,⍙VAL[1↑V;]
     ∇


     ∇ Z←RED X
[1]   ⍝FONCTION INUTILE
[2]    Z←X
     ∇


     ∇ Z←A UNIQ B
[1]    Z←A,B
[2]    →0 IF~∨/∨/(,A)∘.=,B
[3]    10 ERREUR 'UNIQDUMMY'
     ∇


     ∇ Z←VIDE
[1]    Z←''
     ∇
```


AΔDEBUT
DEBUT → ENTRE → DECOMPOSE → MS → MATCH *
SCAN *
IE *
INPUT → CODE *
BCO *
RESULTAT → DECODE *
TRAVAIL → ENCODE *
INPUT *
TEST0 → ENCODE *
INPUT *
IS → IE *
PUT *
VALID → ERREUR → ERR → ERCATE → DECODE *
IE *
ERLX *
ERSY *
IE *
EXEC → ERREUR *
TRDEB *
TRFIN → EDEX → DECODE *
LIRE → DECODE *
BCO *
CAC → STOP *
CAT *
IE *
IE *
TRITOP → RP → RP *
IE *
IE *
IE *
ΔEXP → COMPILE → REDUCE → ERREUR *
ERSY *
IE *
STAKING → ERREUR *
ERSY *
IE *
IE *
TIME *
TRADUC → ENCODE *
SCAN *
IE *
IE *
IE *
TEST1 → ENCODE *
INPUT *
IS *
PUT *
VALID *
IE *
IE *
IE *

LE 14/5/1975 A 10H 16MN

```
    ∇ DEBUT
[1]   DICO←0ρ''
[2]   TTYYPP←0ρ'
[3]   BOOK←0ρ1
[4]   ARB←0ρ1
[5]   TEXP←0ρ1
[6]   TCAT←0ρ1
[7]   ΔTEXP←⍳0
[8]   ΔTCAT←⍳0
[9]   LDEF←⍳0
[10]  ENTRE 1
    ∇
```

```
    ∇ ENTRE N;W;X;ID;IN;DEF;CAT
[1]   BOOK←BOOK,'-'
[2]  B1:'ENTREZ LA LIGNE ';N←N
[3]   'POUR FINIR  TAPEZ: FIN'
[4]   X←INPUT,1
[5]   →(∧/(3↑X)= 7 10 15)/FIN
[6]   →B1 IF DECOMPOSE X
[7]   TRAVAIL
[8]   RESULTAT
[9]   N←N+1
[10]  →1
[11] FIN:'POUR AVOIR LA COMPOSITION DU LIVRE, TAPER: LIVRE'
[12]  'POUR AVOIR LA TABLE DES CONSTANTES ET CELLE DES VARIABLES, TAPE
      R: TABLES'
[13]  →0
    ∇
```

```
    ∇ Z←DECOMPOSE X;PT
[1]   Z←PT←1
[2]   →ER IF 0=ρIN←1 MS X
[3]   →ER IF 0=ρID←1 MS X
[4]   →ER IF 0=ρDEF←1 MS X
[5]   →ER IF 0=ρCAT←1 MS X
[6]   →Z←0
[7]  ER:' NB DE CHAMPS ERRONNE'
    ∇
```

```
    ∇ RESULTAT;Z
[1]   'LIGNE ACCEPTEE'
[2]   50⍴' ';Z←(DECODE IN),'  ',(DECODE ID),'  ',ALPHA[DEF],'  ',ALPH
      [CAT]
[3]  ⍝CONSTRUCTION DU LIVRE
[4]   BOOK←BOOK,Z
    ∇
```

```
    ∇ Z←A MS B
[1]   Z←A MATCH B
[2]   Z←A SCAN B
    ∇
```

```
    ∇ Z←DECODE NB;V
[1]   V←V+0=V←(5⍴47)⊤NB
[2]   Z←⍞(ALPHA,'?')[V]
    ∇
```

```
    ∇ Z←A MATCH B
[1]   PT←PT+⍴Z←(¯1+((A⍳B←(PT-1)↓B)⍳1+⍴A←,A))↑B
    ∇
```

```
    ∇ Z←REPARTIR M;P
[1]   →(M≤N)/S1
[2]   'IMPOSSIBLE, NOUS EN SOMMES A LA LIGNE ';N
[3]   →0
[4]  S1:BOOK←(((+\('-'∘.=BOOK))⍳M)-1)↑BOOK
[5]   TCAT←(M-1)↑TCAT
[6]   ΔTCAT←(P+ΔTCAT[P←¯1↑TCAT])↑ΔTCAT
[7]   LDEF←(M-1)↑LDEF
[8]   →S2 IF 0≠⍴TEXP←(+/LDEF)↑TEXP
[9]   →S3,ΔTEXP←''
[10] S2:ΔTEXP←(P+ΔTEXP[P←¯1↑TEXP])↑ΔTEXP
[11] S3:ARB←(M-1)↑ARB
[12]  DICO←(M-1)↑DICO
[13]  TTYYPP←(M-1)↑TTYYPP
[14]  ENTRE M
    ∇
```

```
    ∇ LIVRE;I;B;POS;BOK
[1]   I←1
[2]   POS←(+/B)↑⍒B←'-'∘.=BOK←1↓BOOK
[3]   POS←0,POS
[4]   B←'-'∘.=BOOK
[5]  RET:''
[6]   5 0 ⍕I;'      ';POS[I]↓(POS-1)[I+1]↑BOK
[7]   I←I+1
[8]   →(I<+/B)/RET
    ∇
```

```
    ∇ TABLES;L
[1]   ''
[2]   ''
[3]   '  TYPE   NOM'
[4]   ''
[5]   (NUMER⍳L),(RHO TTYYPP),(((L←⍴TTYYPP),7)⍴' '),DECODE DICO
[6]   ''
[7]   →0
    ∇
```

LE 14/5/1975 A 10H 27MN

```
    ∇ Z←TRAVAIL;M;CEXP
[1]  ⍝VERIFICATION DE L'IDENTIFICATEUR
[2]  B1:M←DICO⍳ID←ENCODE ID
[3]   →(M>⍴DICO)/B2
[4]   'IDENTIFICATEUR DEJA UTILISE EN LIGNE ',⍕M
[5]   'ENTREZ EN UN AUTRE'
[6]   ID←INPUT
[7]   →B1
[8]  ⍝VERIFICATION DE L'INDICATEUR
[9]  B2:M←DICO⍳IN←ENCODE IN
[10]  →S2 IF M>⍴DICO
[11]  →S1 IF 'V'=TTYYPP[M]
[12] S2:M←0
[13]  →(IN=180548197)/S1
[14]  'INDICATEUR DOIT ETRE UNE VARIABLE DEF OU 0'
[15]  'ENTREZ EN UN AUTRE'
[16]  IN←INPUT
[17]  →B2
[18] ⍝CONSTRUCTION DE L'ARBRE'
[19] S1:ARB←ARB,M
[20]  TEST0
[21]  TEST1
    ∇
```

```
    ∇ Z←CODE EXP
[1]   ALPHA←' ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890,()[]<>;T∈'
[2]   Z←ALPHA⍳EXP
    ∇
```

```
    ∇ ΔARBRE←COMPILE TEXT;STAK;G;S;PIL;N
[1]   PIL←⍳~G←S←1
[2]   ΔARBRE←(0,2+NBF)⍴0
[3]   ΔVAL←(0,NBTR)⍴0
[4]   TEXT←TEXT,STAK← 2 1 ⍴TRUC,0
[5]  S1:STAKING
[6]   →0 IF~VAL
[7]   REDUCE
[8]   →0 IF~VAL
[9]   →S1 IF~(TEXT[1;G]=TRUC)∧(STAK[1;S-1]=TRUC)∧STAK[1;S]=AXIOM
[10]  ΔARBRE←⊖ΔARBRE
[11]  ΔARBRE[;1+⍳1+NBF]←N|(N←1+1↑⍴ΔARBRE)-ΔARBRE[;1+⍳1+NBF]
    ∇
```

```
    ∇ Z←ENCODE A
[1]   Z←47⊥5↑A
    ∇
```

```
    ∇ V←POS GET PILE
[1]   PILE←'Δ',PILE
[2]   ⍎'V←',PILE,'[POS+⍳',PILE,'[POS]]'
    ∇
```

```
    ∇ Z←A INCLU B
[1]   Z←∧/(A∈B)
    ∇
```

```
    ∇ Z←INPUT;L
[1]  ⍝SUPPRESSION DES BLANCS DE FIN DE LIGNE
[2]  B1:L←BCO ⍞
[3]  ⍝TOUS LES CARACTERES SONT ILS VALIDES
[4]   →(~(∧/((Z←CODE L)∈⍳⍴ALPHA)))/S1
[5]   →0
[6]  S1:'CARACTERE NON VALIDE'
[7]   'RECOMMENCEZ'
[8]   →B1
    ∇
```

```
    ∇ Z←A IS B
[1]   Z←1
[2]   →0 IF(0=¯1↑B)∧0=¯1↑A
[3]   →×
    ∇
```

```
     ∇ Z←LI N;R
[1]    Z←0⍴1
[2]    →(0=⍴N)/0
[3]    →(0=R←ARB[N])/0
[4]    Z←(LI R),R
     ∇
```

```
     ∇ POS←V PUT PILE
[1]    POS←1+⍴⍎PILE←'Δ',PILE
[2]    ⍎PILE,'←',PILE,',(⍴,V),V'
     ∇
```

```
     ∇ Z←TEST0;ΔEXP;ΔCAT
[1]   ⍝VERIFICATION DE LA CATEGORIE
[2]    CEXP←0
[3]   B1:→(224465326=ENCODE CAT)/S3
[4]    →S2 IF~VALID CAT
[5]    →S2 IF~ΔCAT IS 0
[6]    CEXP←ΔEXP
[7]    →S3
[8]   S2:⎕←ALPHA[CAT]
[9]    'TAPEZ UNE NOUVELLE CATEGORIE'
[10]   CAT←INPUT
[11]   →B1
[12]  ⍝CONSTRUCTION DE LA TABLE DES CATEGORIES
[13]  S3:TCAT←TCAT,CEXP PUT 'TCAT'
[14]   →0
[15]   ΔTCAT
     ∇
```

```
     ∇ Z←TEST1;ΔEXP;ΔCAT
[1]   ⍝VERIFICATION DE LA DEFINITION
[2]   B1:→(29589555 84511922 ∈ENCODE DEF)/S1,S2
[3]    →S3 IF~VALID DEF
[4]    →S3 IF~ΔCAT IS CEXP
[5]   ⍝CONSTRUCTION DE LA TABLE DES EXPRESSIONS
[6]    TEXP←TEXP,ΔEXP PUT 'TEXP'
[7]   ⍝CONSTRUCTION DE LA TABLE DES CONSTANTES
[8]    LDEF←LDEF,1
```

```
[9]   DICO←DICO,ID
[10]  TTYYPP←TTYYPP,'C'
[11]  →0
[12] S2:DICO←DICO,ID
[13]  TTYYPP←TTYYPP,'C'
[14]  LDEF←LDEF,0
[15]  →0
[16] ⍝CONSTRUCTION DE LA TABLE DES VARIABLES
[17] S1:DICO←DICO,ID
[18]  TTYYPP←TTYYPP,'V'
[19]  LDEF←LDEF,0
[20]  →0
[21] S3:⍞←ALPHA[DEF]
[22]  'TAPEZ UNE NOUVELLE DEFINITION'
[23]  DEF←INPUT
[24]  →B1
[25]  →0
[26]  ΔTEXP
    ∇
```

```
    ∇ Z←TIME;T
[1]   Z← 60 60 60 ⊤(T←⌶21)-TIMER
[2]   TIMER←T
    ∇
```

```
    ∇ Z←TRADUC X;NOM;PT;J
[1]   PT←1
[2]   DUM←''
[3]   Z← 2 0 ⍴0
[4]  S1:→0 IF PT>⍴,X
[5]   →S2 IF 0=NOM←ENCODE SEP SCAN X
[6]   J←DICO⍳NOM
[7]   →ER IF J>⍴DICO
[8]   Z←Z, 2 1 ⍴(VCT['VC'⍳TTYYPP[J]]),J
[9]  S2:→0 IF PT>⍴,X
```

```
[10]  Z←Z, 2 1 ρ(VT[SEP⍳X[PT]]),0
[11]  PT←PT+1
[12]  →S1
[13] ER:J←DUM⍳NOM
[14]  →S3 IF J≤ρDUM
[15]  DUM←DUM,NOM
[16] S3:Z←Z, 2 1 ρ(VCT[3]),-J
[17]  →S2
     ∇
```

```
     ∇ Z←VALID EXP;ΔVAL;ΔSAT;VAL
[1]    VAL←1
[2]    Z←EXEC ΔEXP EXP
[3]    →0 IF Z
[4]    14 ERREUR 'VALID'
     ∇
```

```
     ∇ Z←ΔEXP X;T;ΔARBRE
[1]    T←TIME
[2]    Z←COMPILE TRADUC X
[3]    '**SY***TEMPS MIS : ';⍕TIME
     ∇
```