Un programme qui vérifie des identités en utilisant le raisonnement par récurrence

Martial Vivet

THESE Présentée

Pour l'obtention

du

diplome de Docteur de 3e cycle

à

l'Université de PARIS VI

SPECIALITE : Mathématiques appliquées

MENTION : Informatique

Par : Monsieur Martial VIVET

Sujet de la thèse : Un programme qui vérifie des identités en utilisant le raisonnement par récurrence.

Soutenue le 29 novembre 1973 devant la Commission composée de :

Président	Mr J.C. SIMON
Examinateurs	Mr J. PITRAT
	Mr M. CHEIN
Invité	Mr A. BOUCHET

THESE Présentée

Pour l'obtention

du

diplome de Docteur de 3e cycle

à

l'Université de PARIS VI

SPECIALITE : Mathématiques appliquées

MENTION : Informatique

Par : Monsieur Martial VIVET

Sujet de la thèse : Un programme qui vérifie des identités en utilisant le raisonnement par récurrence.

Soutenue le novembre 1973 devant la Commission composée de :

Président	Mr J.C. SIMON
Examinateurs	Mr J. PITRAT
	Mr M. CHEIN
Invité	Mr A. BOUCHET

Je présente à Monsieur SIMON l'expression de mes remerciements pour avoir accepté de présider le Jury qui examinera le présent travail.

Monsieur CHEIN a bien voulu s'intéresser à cette étude et accepter de participer au Jury de cette thèse. Qu'il trouve ici le témoignage de ma plus profonde reconnaissance.

Que Monsieur PITRAT trouve ici l'expression de toute la gratitude et reconnaissance que j'ai à son égard pour l'attention constante avec laquelle il a suivi le développement de cette recherche et pour les critiques, conseils et encouragements qu'il m'a prodigués.

Je remercie également Monsieur BOUCHET d'avoir accepté de se joindre à ce Jury.

Je tiens enfin à remercier les personnes du service de Mathématiques de la Faculté des Sciences du Mans qui ont participé à divers titres à la réalisation matérielle de ce travail.

UN PROGRAMME QUI VERIFIE DES IDENTITES

EN UTILISANT LE RAISONNEMENT PAR RECURRENCE

TABLE DES MATIERES

I N T R O D U C T I O N

Depuis quelques années, les ordinateurs ont dépassé le stade de purs organes de calculs. Dans leur utilisation une ère nouvelle se prépare : celle où la machine est capable de prendre en charge un certain nombre de tâches nécessitant une certaine intelligence.

Ainsi, en mathématiques, l'ordinateur n'est plus seulement une machine capable de "dévorer" des calculs numériques en suivant un programme préétabli, mais devient un instrument de manipulations formelles d'expressions capable de créer des théorèmes dans une axiomatique donnée, de vérifier des démonstrations ou d'établir certains raisonnements. On peut citer les travaux de Gelernter qui démontre des théorèmes de géométrie euclidienne (pour trouver la solution, la machine s'aide d'une figure qu'elle construit elle-même). Dans le domaine de la logique, les travaux de Newell, Shaw et Simon d'une part et de Pitrat d'autre part sont importants pour la construction de démonstrations et l'élaboration des résultats "essentiels" d'une théorie. En analyse, Bledsoe a redémontré des théorèmes sur la continuité et les limites des fonctions. Plus orientés vers la manipulation formelle d'expressions, les travaux de Slagle (calcul de primitives de fonctions) et de Laurent (détermination de limites de fonctions) sont à retenir.

D'autres domaines d'application font appel à ces possibilités de comportement intelligent des machines : citons la reconnaissance des formes qui a de nombreuses applications pratiques (lecture optique, reconnaissance et génération de la parole, photographie etc...), l'analyse des langues naturelles, les jeux (échecs etc...) le fonctionnement de robots.

Tous ces domaines de recherche se regroupent sous le terme assez flou d'Intelligence Artificielle. Ils ont tous en commun de faire en sorte que l'on puisse obtenir de la machine un comportement intelligent (proche du comportement humain ou pas). Le fonctionnement n'est plus basé sur un algorithme qui donne de façon certaine une solution en un temps fini mais sur des heuristiques qui en quelque sorte suppléent à l'intuition. On se

.../...

donne ainsi un certain nombre de critères ou de règles qui orientent la recherche de la solution vers l'endroit où elle a le plus de chances de se trouver. Ces critères peuvent être plus ou moins bien choisis et la qualité de la solution dépend de la qualité de l'heuristique. Alors qu'un algorithme conduit toujours à la solution si elle existe, l'utilisation d'heuristiques peut mener à une impasse. Une heuristique sera d'autant meilleure qu'elle conduit plus souvent à la solution.

Remarquons que l'utilisation d'heuristiques est très fréquente chez l'humain :

"L'arrivée des cigognesmarque la fin de l'hiver"
"Pour aller plus vite, prendre sa bicyclette"
"Pour intégrer un produit, essayer l'intégration par parties"
"Au cartes, faire d'abord tomber l'atout"

etc...

L'intelligence artificielle fournit un type de recherche d'un intérêt évident pour un enseignant-chercheur. L'aspect enseignant fournit en effet au chercheur un certain nombre de données sous la forme d'analyse du comportement de l'humain en train d'acquérir des connaissances (ce qui peut rendre service pour étudier les phénomènes d'apprentissage). Il permet de dégager rapidement un certain nombre d'heuristiques utilisables dans les programmes. On peut remarquer à ce sujet qu'enseigner les mathématiques par exemple, consiste pour une bonne part à faire manipuler des heuristiques de plus en plus fines à l'enseigné et en particulier lui apprendre à établir lui-même les heuristiques dont il peut avoir besoin pour travailler dans un domaine donné. Réciproquement, l'aspect chercheur oblige l'enseignant à réfléchir sur son propre comportement d'individu ayant à faire acquérir des connaissances et le fait d'être obligé de formaliser les concepts peut éclaircir les idées sur ce qu'est l'intelligence.

En guise de conclusion pour ce paragraphe, on peut dire que le chercheur en Intelligence Artificielle et l'enseignant ont des rôles semblables : le chercheur veut faire acquérir un comportement intelligent à une machine et l'enseignant veut développer l'intelligence de ses élèves. Il semble qu'ils aient de bons services à se rendre mutuellement.

.../...

Enseignant les mathématiques dans le premier cycle d'Université, il était intéressant de voir comment il est possible de faire faire à un ordinateur le travail que l'on fait faire aux étudiants. Le domaine choisi se limite à la manipulation d'expressions. On sait qu'il est assez difficile d'obtenir des étudiants une certaine aisance dans les calculs. L'entraînement et l'habitude du calcul comptent pour beaucoup mais il y a aussi un certain apprentissage qu'il est intéressant d'étudier afin d'en dégager les mécanismes. Pour cette raison, nous avons choisi de travailler non pas avec des axiomatiques fabriquées pour les besoins de la cause ou bien formalisées, mais avec des calculs que l'on peut réellement proposer à des étudiants (nous nous sommes limités à la trigonométrie, la dérivation, les suites numériques et le calcul algébrique habituel). La vérification d'égalités à l'aide du raisonnement par récurrence a été choisie parce qu'on y travaille avec une règle (l'hypothèse de récurrence) proche formellement de l'expression à démontrer.

Dans la première partie, on précise le type de problèmes en vue, les limites du sujet. Les notions essentielles sur les représentations d'expressions sous forme préfixées et l'algorithme d'unification y sont rappelées brièvement.

La deuxième partie, plus fondamentale, revient sur la notion de problème. La méthode de Siklossy pour démontrer des théorèmes dans une axiomatique est exposée ainsi que divers aménagements qui ont pu être apportés à cette méthode pour traiter les problèmes envisagés.

La troisième partie montre comment ont été résolus les problèmes liés à la manipulation de chaînes de symbôles (politique de simplification et normalisation).

La quatrième partie présente le programme qui a été réalisé pour mettre en oeuvre les principes développés dans la seconde. Les performances et résultats obtenus par le programme y sont donnés.

1ère PARTIE

A - POSITION DU PROBLEME

I - Type de problèmes en vue limites du sujet

II - Nature heuristique du problème

B - RAPPELS DE NOTIONS ESSENTIELLES - NOTATIONS

I - Polonaises préfixées -théorème du rang représentation en arbres

II- Algorithme d'unification

A - POSITION DU PROBLEME

Le but fixé pour ce travail est de vérifier des égalités à l'aide du raisonnement par récurrence.

Nous avons utilisé cette règle d'inférence sous sa forme la plus courante dans la pratique , à savoir :

$$a_{(o)}$$

$$\frac{\forall y,\ a_{(y)} \Rightarrow a_{(y+1)}}{\forall y,\ a_{(y)}}$$

(le trait sépare les antécédants de ce qui est inféré)
Cela signifie que pour vérifier qu'une propriété $a_{(y)}$ est vraie pour tout y, il suffit de vérifier qu'elle l'est pour (o), et que si elle est vraie pour y, elle l'est encore pour (y + 1) .

I - Types de problèmes en vue

Nous nous sommes limités à la vérification d'égalités afin de réduire les problèmes de représentations des objets manipulés. Ainsi la propriété générale $A_{(n)}$ à vérifier prendra toujours la forme $G_n = D_n$ (où G (resp. D) est le membre de gauche (resp. Droit) de l'égalité : n sera toujours la variable sur laquelle porte la récurrence).

Les problèmes que nous nous sommes posés peuvent être rangés dans quatre grandes classes.

1) Algèbre générale - Il s'agit là de problèmes ne faisant appel qu'à des opérations simples et aux opérateurs Σ et π (exemple typique de problème de cette classe $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$) .

2) Trigonométrie - Les problèmes sont de même nature mais l'appel à la trigonométrie alourdit beaucoup la manipulation formelle des expressions.
(exemple typique $\sum_{i=1}^{n} \cos ix = \frac{\sin n \frac{x}{2}}{\sin \frac{x}{2}} \cos (n+1) \frac{x}{2}$)

...

3) Dérivation - Deux catégories de problèmes peuvent se poser en récurrence

α - dérivée première d'une fonction dépendant du paramètre n

(exemple typique $\frac{d}{dx}\left(\frac{x^n}{(1+x)^n}\right) = \left(\frac{nx^{n-1}}{(1+x)^{n+1}}\right)$

β - Expression de la dérivée $n^{ième}$ d'une fonction

(exemple typique $\frac{d^n}{dx^n}(x e^x) = (x + n) e^x$

4) Problèmes de suites numériques - Là encore, deux grandes catégories de problèmes

α - On se donne une suite par une formule de récurrence sous la forme :

$$u_o = \alpha$$
$$u_{n+1} = f(u_n)$$

Le problème consiste à vérifier l'expression de u_n en fonction de n ($u_n = \varphi(n)$)

β - L'autre catégorie est formée par les problèmes inverses des précédents . On se donne $u_n = \varphi(n)$, il s'agit de vérifier que la suite (u_n) peut aussi être définie par $u_o = \alpha$

$$u_{n+1} = f(u_n)$$

Exemples typiques

forme α

Si $\forall n$ $\begin{cases} u_1 = 3A + 2 \\ u_{n+1} = 3u_n + 2 \end{cases}$ vérifier que $\forall m$ $u_m = 3^m(A + 1) - 1$

forme β

Si $\forall n$ $u_n = 3^n(A+1)-1$ vérifier que $\forall m$ $\begin{cases} u_1 = 3A + 2 \\ u_{m+1} = 3u_m + 2 \end{cases}$

Un domaine d'applicationsparticulièrement intéressant fait se rejoindre le problème des dérivations et celui des suites numériques et relations de récurrence : Il s'agit des familles de polynômes orthogonaux.

Remarque : D'autres domaines en mathématique font appel au raisonnement par récurrence. Ainsi certaines intégrales, des déterminants d'ordre n, des puissances de matrices etc.. peuvent se calculer ainsi. Nous n'avons pas traité d'exemples dans ces domaines. Les techniques utilisées font appel

...

à "un outillage" mathématique assez différent de celui que nous avons considéré. Notre programme devrait pouvoir travailler dans ces domaines à condition de lui donner les règles de manipulations correspondantes. Celles-ci semblent assez nombreuses et les ajouter à la centaine que nous avons dû déjà considérer alourdirait le programme.

Limite du sujet

Le programme que nous avons écrit vérifie uniquement par récurrence qu'une certaine égalité s'écrit

$$G_n = D_n$$

Il n'établit pas cette formule. Ainsi nous pouvons lui demander de vérifier que la somme des n premiers nombres pairs s'écrit n (n + 1) (ce qui se donne sous la forme $\sum_{i=1}^{n} 2i = n(n+1)$. En aucun cas on ne peut lui poser le problème sous la forme "donner une expression simple de la somme des n premiers nombres pairs" (ce qui reviendrait à poser le problème sous la forme $\sum_{i=1}^{n} 2i = ?$)

Le problème de la création de la formule semble beaucoup plus difficile. Il ne fait pas appel aux mêmes processus de pensée.

Le problème tel que nous l'avons traité fait essentiellement appel à la manipulation formelle d'expressions (traitement de chaînes de symboles). L'intelligence artificielle intervient sous forme d'utilisation d'heuristiques facilitant cette manipulation (heuristiques permettant de réduire l'utilisation de règles (très nombreuses dans ce type d'application), heuristiques permettant de faire apparaître des termes cherchés etc...). Ces heuristiques seront reprises en détail ultérieurement.

Pour faire découvrir à un programme la formule, nous sommes obligés de lui donner des possibilités de travail dans le domaine de l'induction, de la généralisation et de l'émission d'hypothèse.

Ainsi pour découvrir la formule donnant la somme des n premiers nombres pairs un programme pourrait calculer les résultats pour les premières valeurs de n. Par exemple, il obtiendrait

...

n	somme	résultat = f(n)
2	2 + 4	6
3	2 + 4 + 6	12
4	2 + 4 + 6 + 8	20
5	2 + 4 + 6 + 8 + 10	30
6	2 + 4 + 6 + 8 + 10 + 12	42
7	2 + 4 + 6 + 8 + 10 + 12 + 14	56

Il pourrait s'apercevoir alors que pour n impair, le résultat est égal au produit du nombre de nombres par le nombre médian. A partir de là, il pourrait remarquer que même pour les lignes où n pair le nombre (n) de nombres à ajouter divise le résultat. Il pourrait alors émettre l'hypothèse $f_{(n)} = n\,\varphi_{(n)}$.

Pour éclaircir $\varphi_{(n)}$, il pourrait voir que le quotient $\frac{f(n)}{n}$ est toujours égal à la moyenne du premier et du dernier nombre écrit.

Le premier nombre écrit est toujours 2, le dernier toujours 2 n . A partir de là, il est possible d'émettre l'hypothèse $\varphi_{(n)} = \frac{2 + 2n}{2}$ et par là même $f_{(n)} = n\frac{(2 + 2n)}{2}$.

C'est alors que doit entrer en jeu un programme du type de celui que nous avons écrit afin de vérifier que les hypothèses émises sont valables pour tout n.

Plusieurs cas sont possibles :

1) Si le programme de vérification se termine par un succès sur l'expression fournie par le programme de création, alors il y a succès complet de l'opération.

2) Si la vérification échoue, on peut l'aborder différemment pour tenter d'achever avec succès. Il se peut que la formule émise soit fausse et que l'on soit obligé de retourner vers l'émission d'hypothèse (on peut peut-être commencer par regarder si la formule reste vraie pour des valeurs de n plus grandes que celles qui ont servi à poser l'hypothèse).

Nous voyons alors que pour résoudre le problème dans son entier, on pourrait concevoir un système de deux programmes en interaction : l'un qui émettrait des hypothèses, l'autre qui les vérifierait en établissant complètement leur validité. Ces deux programmes pourraient se transmettre

...

un certain nombre de renseignements à chaque fois qu'ils se donneraient le contrôle.

Ces problèmes d'émission d'hypothèses restent encore largement ouverts en intelligence artificielle, mais nous pouvons avoir l'espoir de les voir se résoudre. De toute façon les mécanismes qu'utilise l'homme pour résoudre ce type de problèmes méritent d'être démontés et examinés : c'est une façon de faire un pas de plus dans la connaissance. Le fait de maîtriser ces différents processus pourrait être d'un grand intérêt pédagogique ; Nous pourrions peut-être enfin apprendre clairement aux hommes à élaborer leur "intuition".

II - Nature heuristique du problème posé

A priori, le problème paraît de nature algorithmique. Les étapes à suivre pour vérifier une égalité à l'aide du raisonnement par récurrence sont en effet parfaitement déterminées. Il suffit dans un premier temps de vérifier l'égalité pour le rang 1. Ensuite posant qu'elle est vraie au rang n, il il faut prouver qu'elle l'est encore au rang (n + 1). La finesse du mathématicien n'intervient pas au niveau de la méthode suivie mais au niveau du travail indiqué par la méthode ; aussi, c'est au niveau des preuves qu'interviennent les techniques de l'intelligence artificielle. Les heuristiques permettent de conduire les calculs à leur fin en essayant d'éviter une profusion trop grande d'expressions stériles. Nous voyons donc que la partie essentielle de ce travail n'est pas dans l'écriture d'un programme traitant de la récurrence mais dans la fabrication d'un prouveur performant. (celui-ci sera décrit en détail dans la deuxième partie).

B - RAPPELS DE NOTIONS ESSENTIELLES - NOTATIONS

I - Polonaises préfixées

Avec le but que nous nous sommes fixés, nous avons à manipuler des objets (formules mathématiques) pour lesquels, il est fondamental de trouver une représentation pratique et efficace.

Alphabet - Il comprend :

* Les variables : ensemble fini d'objets que l'on suppose de cardinal suffisamment grand pour satisfaire les besoins (ex : X, Y ,Z, U, V, W,...)

* Les constantes : représentation des nombres .

* Les connectives (ou opérateurs) : (On trouvera en annexe B la liste des connectives utilisées par notre programme). A chaque connective est associé un degré n ($n \in \mathbb{N}$). La connective est alors dite n-aire. (Pratiquement, nous n'avons utilisé que des connectives unaires et binaires).

Termes - Les objets que nous manipulons sont des termes au sens de la définition suivante :

1- Une variable est un terme

2- Une constante est un terme

3- Si C est une connective n-aire et si $a_1, a_2, \ldots, a_n$ sont n termes, alors la concaténation C $a_1, a_2, \ldots, a_n$ est un terme

Exemples 1

X
3
+ X 3
SIN + X 3
COS * Z + X 3 sont des termes

Remarque :

Cette notation redonne aux opérations leur nature de fonction (ainsi pour l'addition fonction "somme")

$+ \quad E \; X \; E \longrightarrow E$

$(x , y) \longmapsto + \; x \; y$

(somme de x et de y)

...

Nous avons défini là, la notation préfixée.(Nous ne parlerons pas ici des autres notations linéaires utilisables : suffixée, infixée, totalement parenthésée. Voir à ce sujet [IV 10]).

On y trouvera également un certain nombre d'algorithmes permettant de réaliser les passages entre ces diverses notations.

Théorème fondamental de la notation préfixée

Ce théorème permet de retrouver facilement le terme commençant en un certain symbole d'un terme. (On peut en trouver la démonstration dans [V 10] ; celle-ci est formulée sous forme de problème dans [V 6] p 105-106)

- Notion de rang d'une suite de symboles

. Si la suite de symboles est vide, son rang est 0 $(r(\Lambda) = 0)$
. Le rang d'une variable (ou constante) est -1 $(r(x) = -1)$
. Le rang d'une connective n-aire est (n - 1) $(r(C_n) = n - 1)$
. Si A est une suite de symboles et a un symbole alors

$$r(A.a) = r(A) + r(a)$$

exemples 2

$r(+) = 1$ $r(SIN) = 0$ $r(x) = r(y) = -1$
$r(+X) = 0$ $r(SIN\ X) = -1$ $r(SIN + x\ y) = -1$

-Théorème

La suite de symboles $a_1, a_2, \ldots, a_p$ est un terme si et seulement si :

1) $r(a_1, a_2, \ldots, a_p) = -1$
2) $\forall i < p \quad r(a_1, a_2, \ldots, a_i) \geq 0$

exemples 3

. SIN + X Y est un terme

$r(sin) = 0 \quad \geq 0$
$r(sin\ +) = 1 \quad \geq 0$
$r(sin+\ X) = 0 \quad \geq 0$
et $r(sin+XY) = -1$

. + X Y SIN n'est pas un terme

$r(+) = 1 \quad \geq 0$
$r(+X) = 0 \quad \geq 0$
$r(+XY) = -1 \quad < 0$
et pourtant $r(+\ XY\ SIN) = -1$

...

Ce théorème est constamment utilisé dans la pratique. Il justifie le choix de la notation préfixée ; Pour chercher le terme commençant en un certain symbole, on met au départ le rang à 0 et on ajoute au rang de la suite celui des symboles les uns après les autres en s'arrêtant dès que l'on trouve -1.

exemple 4 faisant apparaître tous les termes successivement

	/	---	SIN	+	X	*	Z	T	LOG	RAC	+	1	PUI	X	2
1	1	1	1	2	1	2	1	0	0	0	1	0	1	0	-1
2		0	0	1	0	1	0	-1							
3			0	1	0	1	0	-1							
4				1	0	1	0	-1							
5					-1										
6						1	0	-1							
7							-1								
8								-1							
9									0	0	1	0	1	0	-1
10										0	1	0	1	0	-1
11											1	0	1	0	-1
12												-1			
13													1	0	-1
14														-1	
15															-1

La ligne 1 représente tout le terme

$$\frac{-\sin(x+(z*t))}{\mathrm{LOG}(\sqrt{1+x^2})}$$

La ligne 2 fait apparaître tout le numérateur
La ligne 9 le dénominateur
La ligne 7 le z * t
La ligne 11 le $1 + x^2$

etc...

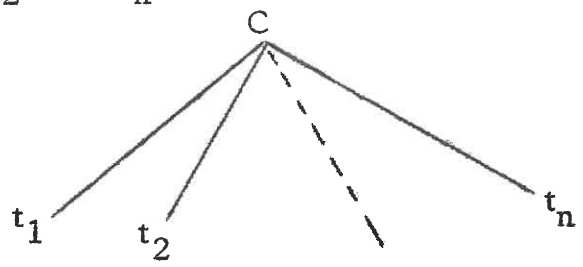
Le théorème permet de délimiter exactement chaque terme et d'éliminer tout besoin de parenthèses qui alourdissent les représentations qui en font l'usage.

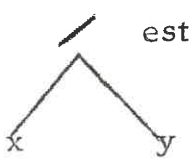
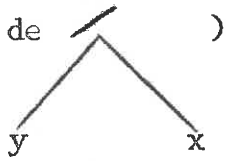
Représentation en arbre

Il peut être intéressant dans un ordinateur (où il n'y a pas de contraintes typographiques) de représenter une expression par un graphe.

(la manipulation est rendue aisée par l'utilisation de langages de listes). Chaque terme est alors représenté de la façon suivante :

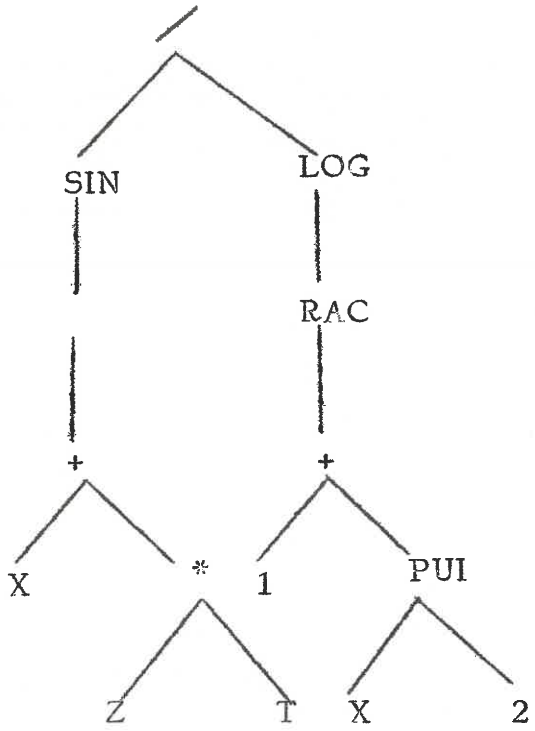
- Une variable (ou constante) est un noeud
- Si C est une connective n-aire et $t_1, \ldots, t_n$ n termes, le terme C $t_1, t_2, \ldots, t_n$ est représenté par :



Nous obtenons ainsi des arborescences dans lesquelles l'ordre des branches à de l'importance (ainsi  est différent de )

exemple 5

Le terme de l'exemple 4 précédent donne l'arborescence suivante :



Des algorithmes très rapides et très simples permettent de passer de l'arborescence à la notation préfixée et réciproquement. (on peut les trouver dans [IV 10].)

...

Cette représentation est pratique lorsque l'on désire substituer un terme à un autre. Il suffit de modifier le pointeur de la tête de l'expression,opération plus rapide que de recopier des termes (surtout lorsqu'ils sont longs).

Cette technique a été utilisée dans le programme sur la récurrence pour substituer 0 (ou (n + 1)) à n dans la formule à vérifier. Cela s'est avéré utile également dans la manipulation des Σ et π.

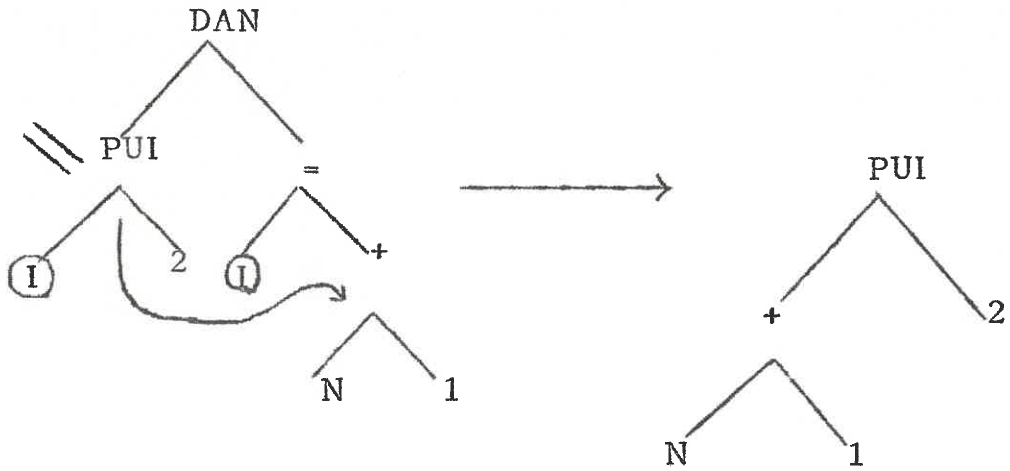
Par exemple : Soit la règle

= SIG IND I U DEA 1 + N1 + SIG IND I U DEA 1N DAN U=I+N 1

(en clair ,elle représente $\sum_{i=1}^{n+1} u_i = \sum_{i=1}^{n} u_i + u_{n+1}$)

Nous voyons que nous avons dans le terme u_i à remplacer i par (n + 1). Ceci est indiqué par le terme DAN U = I + N 1 . La présence de DAN active une procédure de traitement qui remplace ce terme par le terme u_i dans lequel i est remplace par + N 1 .

Ainsi, si U vaut PUI I 2 la procédure a juste à faire le passage,



c'est à dire détecter les I et modifier les pointeurs.

Cette représentation présente l'inconvénient d'être coûteuse en mémoires. Nous avons archivé tous nos résultats intermédiaires sous cette représentation. C'est probablement une erreur qu'il faudrait éviter si le système devait être reprogrammé. Le stockage en polonaises préfixées doit suffire. Il suffit de passer sous forme arborescente temporairement,

...

localement, pour effectuer un travail qui se fait mieux sous cette forme. A l'usage, il semble que le "gaspillage" mémoire ne soit pas équilibré par les avantages apportés.

II - Algorithme d'unification.

Lorsque l'on désire appliquer une règle à une expression, il est très rare que cela se fasse directement. Il faut en général faire dans les formules un certain nombre de substitutions . Par exemple, pour utiliser le développement du sinus de somme sur l'expression SIN (2x + 3 y) avec la formule sin (u + v) = sin u cos v + sin v cos u, il faut remplacer u par 2 x et v par 3 y pour faire coïncider l'expression donnée avec le premier membre de la règle. On trouve alors le résultat en faisant les mêmes substitutions dans le second membre.

Il existe un algorithme qui donne systématiquement les substitutions amenant le résultat le plus général possible (en ce sens que tous les autres résultats que l'on pourrait obtenir en faisant d'autres substitutions dans le théorème ou la règle se déduisent du premier par des substitutions).

Méthode : Soit l'objet T auquel on désire appliquer la règle A : = B . Ces expressions sont des chaînes de symboles (variables,constantes ou connectives).

$T : t_1, t_2, \ldots, t_n$

$A : a_1, a_2, \ldots, a_m$

$B : b_1, b_2, \ldots, b_r$

Le problème est d'amener A et T à coïncider en ne faisant que les substitutions indispensables.

1) On vérifie d'abord qu'aucune variable ne figure à la fois dans T et dans A . Si cela se produit, il suffit de substituer aux occurences de cette variable dans T une autre variable qui ne figure ni dans A : = B ni dans T .

2) Supposons que les p premiers symboles de T et A coïncident (on peut avoir p = O au départ)

...

On a le schéma $T : t_1\, t_2, \ldots, t_p, t_{p+1}, \ldots, t_n$

$A : t_1\, t_2, \ldots, t_p, a_{p+1}, \ldots, a_m$

$B : b_1, b_2, \ldots, b_r$

On compare t_{p+1} et a_{p+1}. Trois cas sont possibles :

α - t_{p+1} et a_{p+1} sont tous deux des connectives.

Si t_{p+1} et a_{p+1} sont identiques, alors les (p+1) premiers symboles étaient identiques. Augmenter p de 1 et recommencer en 2)

Si t_{p+1} et a_{p+1} sont différentes, il y a échec (1). On ne peut faire coïncider deux connectives puisque les substitutions ne portent que sur les variables (voir exemple 3 suivant).

β - t_{p+1} et a_{p+1} sont tous deux des variables.

Dans ce cas on remplace chaque occurence de a_{p+1} par t_{p+1} dans

$t_{p+2}, \ldots, t_n$

$a_{p+2}, \ldots, a_m$

$b_1 \ldots \quad \ldots b_r$

(Il est inutile de faire le remplacement dans les (p+1) premiers a_i et t_i. En effet, ces suites coïncident déjà. Elles coïncideront encore après la substitution (et seul le résultat B modifié nous intéresse).

γ - L'un est une variable, l'autre est une connective.

Supposons par exemple, t_{p+1} connective et a_{p+1} variable (l'autre cas pourrait se traiter de façon identique mais nous ne l'avons jamais utilisé et nous l'avons considéré comme cas d'échec (3)).

On détermine le terme qui commence en t_{p+1} (par exemple $t_{p+1}, \ldots, t_{p+k}$). (Il suffit pour cela d'utiliser le théorème fondamental de la notation polonaise) et on substitue ce terme à la variable a_{p+1} partout où nous l'avons rencontré dans

$t_{p+k+1}, \ldots, t_n$

$a_{p+2}, \ldots, a_m$

$b_1 \ldots \quad \ldots b_2$

...

Comme précédemment, il est inutile de faire la substitution dans $a_1, \ldots, a_{p+1}$. Il y a une difficulté si la variable a_{p+1} apparaît dans le terme $t_{p+1}, \ldots, t_{p+k}$. Il est impossible par des substitutions de faire coïncider une variable avec un terme contenant cette variable. Ce cas ne peut se produire au départ, les variables de T et A étant différentes mais après quelques substitutions, il peut se présenter. On a une deuxième cause d'échec (2). (voir exemple 4 suivant).

Si l'on n'est pas dans ce cas, le nombre de symboles identiques a augmenté et on poursuit en comparant t_{p+k+1} et a_{p+2}.

L'idée est donc d'augmenter constamment le nombre de symboles qui coïncident au début de A et de T. On poursuit le processus jusqu'à ce que les chaînes A et T soient épuisées. (elles le sont toujours et en même temps d'après les propriétés de la notation polonaise).

Ce qui précède rappelle l'essentiel de ce qu'il faut savoir pour manipuler cet algorithme. Il résume certains passages de [IV-11] dans lequel on peut trouver plus de détails, de justifications et des exemples dans le domaine de la logique.

Exemples divers dans le domaine qui nous concerne

1- exemple sans échec (ici $\frac{d}{dx} xe^x$ avec la règle $\frac{d}{dx}(u\,v) = u\frac{dv}{dx} + v\,\frac{du}{dx}$)

T : ~~DER~~ ~~IND~~ ~~1~~ ~~X~~ ~~*~~ ~~X~~ ~~EXP~~ ~~X~~

A : ~~DER~~ ~~IND~~ ~~1~~ ~~X~~ ~~*~~ ~~U~~ ~~V~~

B : + * ~~U~~ DER IND 1 X ~~V~~ * ~~V~~ DER IND 1 X ~~U~~

(~~U~~ = X ; ~~V~~ = EXP X ; ~~V~~ = EXPX)

L'algorithme réussit et fait voir que l'on peut appliquer la formule de dérivation d'un produit pour dériver $(x\,e^x)$. Les cinq premiers symboles sont identiques. Ensuite on unifie U avec X en remplaçant U par X partout où il apparaît. Puis V s'unifie avec le terme EXP X. Le résultat est obtenu dans B modifié par ces substitutions.

2 - application de $\sin^2 U + \cos^2 U = 1$ à $\sin^2 5x + \cos^2 5x$

T : ~~* PUI SIN * 5 X 2 PUI COS * 5 X 2~~
A : ~~* PUI SIN U 2 PUI COS U 2~~
*5 X

B : 1

Les trois premiers symboles coïncident, ensuite la variable U est remplacée par le terme * 5 X partout où on la rencontre. Les chaînes de symboles coïncident alors ; il y a succès. (Remarquons que B n'a pas été modifié).

3 - application de $\sin^2 u + \cos^2 u = 1$ à $\sin^2(5x) + \cos^2(5+ x)$

T : ~~* PUI SIN * 5 X 2 PUI COS~~ + 5 X 2
A : ~~* PUI SIN U 2 PUI COS~~ U 2
*5 X
B : 1

Les trois premiers symboles coïncident, la variable U s'unifie avec le terme * 5 X . L'unification s'arrête au 10 ème symbole ; en effet, nous trouvons là deux connectives différentes (+, *). Il y a échec (1) , la règle ne s'applique pas.

4 - application de 2 sin U cos U = sin 2 U à 2 sin x cos 3 x

T : * * 2 SIN ~~X~~ COS * 3 X
A : * * 2 SIN ~~U~~ COS U
X
B : SIN * 2 ~~U~~
X

Les quatre premiers symboles coïncident, nous unifions alors X et U en remplaçant U par X partout où il figure. Les COS s'unifient bien mais au septième symbole nous avons à unifier la variable X avec le terme * 3 X qui contient lui-même X. Il y a échec (2). La règle ne s'applique pas (les variables étaient pourtant bien différentes au départ dans la règle et l'expression).

5 - application de $\frac{\sin U}{\cos U} = \operatorname{tg} U$ à $\frac{\sin 2x}{\cos y}$

...

T : / SIN ~~* 2 X~~ COS y

A : / SIN ~~U~~ COS ~~U~~
* 2 X

B : TG ~~U~~
* 2 X

Les deux premiers symboles coïncident, on unifie U avec le terme * 2 X, les COS s'identifient bien mais nous devons interdire l'unification de la variable y (de T) avec le terme * 2 X (de A) - (échec (3)). Sinon la réponse TG * 2 X serait acceptée sans condition (réponse fausse en général , vraie justement si y = * 2 X).

6 - application de 2 sin U cos U = sin 2 U à 2 sin (3 + α) cos U

T : * 2 SIN ~~+ 3 α~~ COS ~~U~~
+ 3 α

A : * 2 SIN ~~U~~ COS ~~U~~
+ 3 α

B : SIN * 2 ~~U~~
+ 3 α

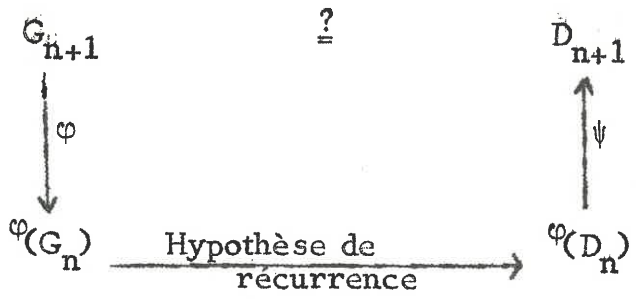
L'unification se fait très bien. Les U sont remplacés par + 3 α ; il y a succès. Mais le fait d'avoir négligé de remarquer qu'il y avait une variable identique dans la règle et l'expression suffit pour obtenir le résultat faux 2 sin (3 + α) cos U = sin 2 (3 + α) .

Raisonnement par récurrence et deuxième cas d'échec

Lorsque nous vérifions une égalité par récurrence, au moment d'appliquer l'hypothèse de récurrence, nous avons une règle ($G_n = D_n$) qui ne diffère du théorème à démontrer ($G_{n+1} = D_{n+1}$) que par la substitution ($n \rightarrow n+1$). Ceci impose pour que la preuve ne soit pas immédiate que ce soit bien la même variable n qui soit dans la règle et le théorème. En effet, si on part avec la règle $G_m = D_m$ et que l'on pose le problème $G_{n+1} = D_{n+1}$ l'unification fonctionne parfaitement et répond. Il suffit de remplacer m par (n+1). Par contre si l'on part avec la règle $G_n = D_n$ et le problème $G_{n+1} = D_{n+1}$, l'unification ne fonctionne plus car l'on est amené à remplacer la variable n par le terme n + 1 (qui contient n) . Nous sommes alors dans le 2ème cas d'échec. L'hypothèse de récurrence ne peut alors s'appliquer qu'en manipulant le $G_{n+1} = D_{n+1}$ afin d'y faire apparaître (G_n ou D_n).

...

Le schéma d'application de l'hypothèse de récurrence est alors le suivant :



Les principales difficultés sont évidemment dans la recherche des transformations φ et ψ .(φ représente les transformations à effectuer pour permettre l'application de l'hypothèse de récurrence, ψ représente les transformations à effectuer pour rendre évidente l'égalité $\varphi(D_n) = D_{n+1}$ c'est à dire pour que les expressions $\psi(\varphi(D_n))$ et D_{n+1} soient identiques).

2ème PARTIE

LE PROUVEUR

* Introduction
* Notion de problème
* La méthode de Siklossy
* Quelques aménagements à cette méthode
  - Intervention du membre droit au cours de la preuve
  - Redémarrage du Problème
    - évaluation de la distance entre deux expressions
    - évaluation de l'état d'avancement des calculs
  - Amélioration du problème posé
  - Le sélecteur de règles
  - Utilisation de règles conditionnelles

CHAPITRE I

INTRODUCTION - NOTION DE PROBLEME

L'objet de cette partie est de présenter la méthode de preuve automatique (on dira le PROUVEUR) utilisée. Dans la formulation faite des problèmes que nous avons en vue, il s'agit de vérifier que ($\forall$ n, $G_n = D_n$) c'est à dire qu'une certaine égalité est vraie pour tout n. Le raisonnement par récurrence indique que l'on doit d'abord vérifier que la formule est vraie pour $n = n_o$ (n_o = 0 ou 1, généralement). Ensuite prenant l'égalité au rang n comme hypothèse de récurrence, il s'agit de prouver que l'on a $G_{n+1} = D_{n+1}$. C'est là généralement qu'est la difficulté. Cette preuve est le coeur du problème. Elle peut nécessiter des moyens assez grands pour un mathématicien. Pour nous, elle suppose l'emploi d'un prouveur puissant, efficace (et assez général étant donnée la diversité des problèmes envisagés).

Nous pouvons remarquer dès maintenant que la variable n sur laquelle porte la récurrence doit être la même dans la règle qui traduit l'hypothèse de récurrence (HR) et dans la formule à prouver ($G_{n+1} = D_{n+1}$) En effet, si l'on prend pour hypothèse $(HR)_m$ $G_m = D_m$ et que l'on demande de prouver que $G_{n+1} = D_{n+1}$, on voit qu'immédiatement $(HR)_m$ s'applique puisque l'unification répond qu'il suffit de remplacer m par (n+1) dans la règle pour obtenir la formule cherchée. Par contre avec $(HR)_n$, l'unification échoue puisque l'on serait amené à remplacer la variable n par le terme n + 1 qui contient précisément cette variable n (2 e cas d'échec de l'algorithme d'unification - voir p. 21).

Dans ce premier chapître nous essayons de dégager la notion de problème . Le chapître II présentera la méthode de Siklossy pour résoudre des problèmes de ce type. On verra en particulier les avantages de cette méthode. Le chapître III indiquera un certain nombre d'aménagements et de prolongements apportés à cette méthode pour l'adapter aux problèmes envisagés.

...

NOTION DE PROBLEME

Un problème pour un élève, c'est quelque chose qui possède un énoncé où figurent des hypothèses et des questions figurant la conclusion à laquelle il doit arriver, ce qui lui est demandé, c'est de tracer le chemin logique qui sépare l'hypothèse de la conclusion. Il a pour cela à sa disposition un certain nombre de théorèmes ou de définitions. On voit ainsi que trois éléments fondamentaux interviennent dans la notion de problème : l'hypothèse qui est l'état de départ, la conclusion qui représente le but à atteindre, les règles qui permettent de cheminer vers le but. Résoudre le problème c'est trouver le passage qui permet en partant de l'hypothèse d'atteindre le but en n'utilisant que les règles opératives permises.

Par exemple les jeux (échecs, dames,...) représentent un problème pour chacun des adversaires : l'hypothèse est en fait la situation de départ, le but à atteindre c'est d'obtenir une situation gagnante et les règles du jeu sont les seules permises.

Un problème peut donc être présenté comme un triplé

(H, B, O) (Hypothèse, But, Opérateurs)

Résoudre le problème c'est trouver une suite d'opérateurs O_i telle que

$$H \xrightarrow{O_1} S_1 \xrightarrow{O_2} S_2 \xrightarrow{O_i} S_i \xrightarrow{O_p} B$$

Les S_i sont les situations intermédiaires issues des applications successives des O_i. La suite des S_i peut-être intéressante à différents points de vue. On peut chercher à réduire le nombre de situations intermédiaires. On peut chercher à les interpréter dans le contexte du problème, à voir si certaines ne sont pas indispensables à la solution ou au contraire si certaines peuvent être évitées avec une autre suite d'opérateurs. On peut aussi y accrocher des notions d'esthétique (une "belle" partie, une "belle" démonstration).

- Dans le cas où un certain nombre de situations intermédiaires apparaissent comme indispensables à la solution, il est intéressant d'introduire la notion de sous-problème.

...

Ainsi sur la figure 1 si l'on s'aperçoit que les différentes

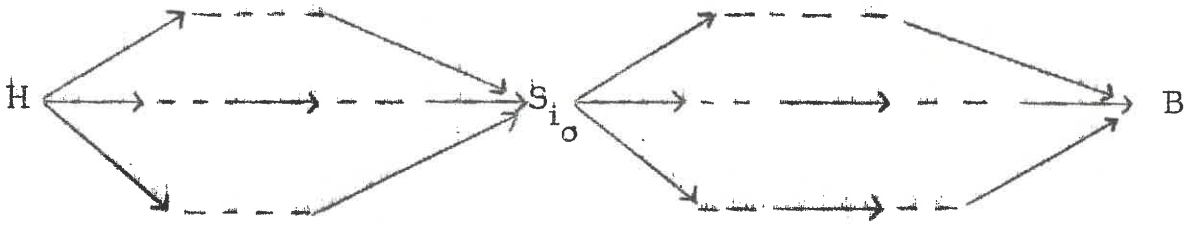


Fig. 1

suite d'O_i solutions de (H, B, O) font passer par la situation S_{i_o}. Il peut être intéressant de décomposer le problème (H, B, O) en deux sous-problèmes (H, S_{i_o}, O) et (S_{i_o}, B, O). Il est souvent aisé de s'apercevoir de cet état de chose une fois le problème résolu. Le rôle de l'intuition, de l'intelligence est peut-être de faire en sorte que l'on puisse s'en apercevoir avant la résolution afin de faciliter celle-ci.

- On peut remarquer que lors de la recherche de la suite (O_i) solution, rien n'impose de partir de H et de cheminer vers B en ne se servant que de situations issues de H c'est à dire que rien n'impose de fabriquer la suite des O_i dans l'ordre chronologique $O_1, O_2, \ldots, O_l$. Il est possible de partir de B et de remonter vers H c'est à dire de fabriquer la solution en déterminant dans l'ordre chronologique $O_l, O_{l-1}, \ldots, O_2, O_1$. Il est également possible de fabriquer la solution en descendant de H vers S_α avec les α premiers opérateurs solutions et de remonter de B vers S_α avec les $(l - \alpha)$ derniers opérateurs solution.

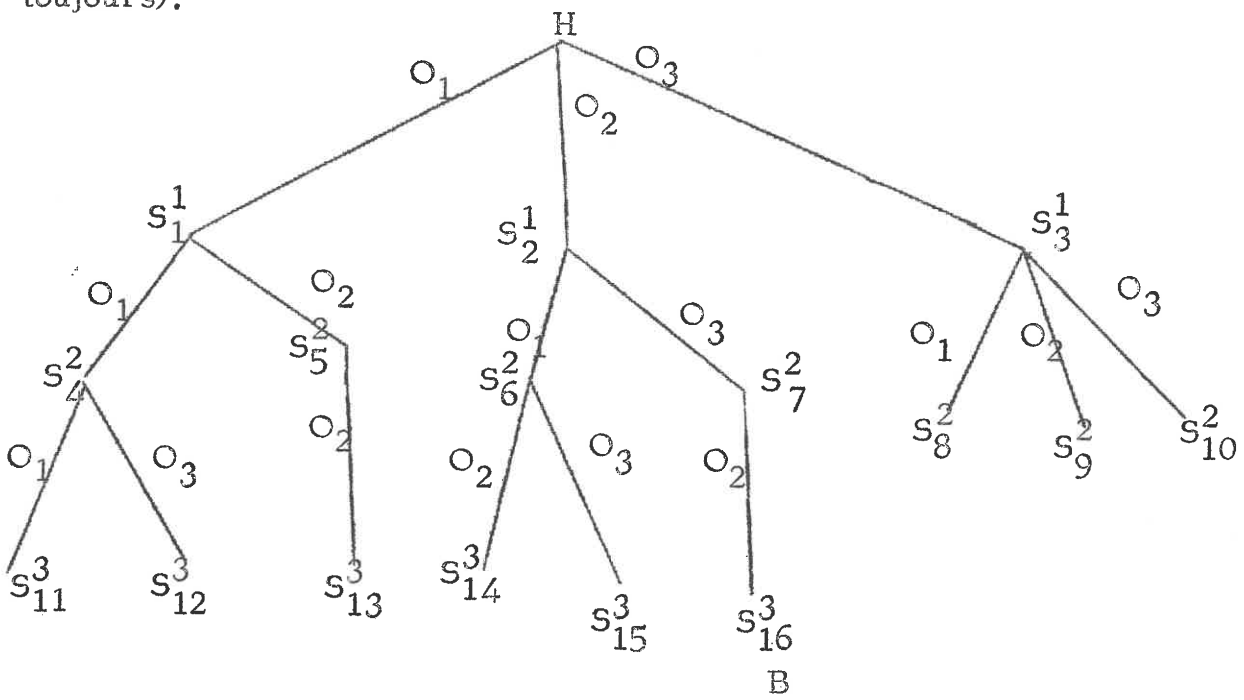
$$H \xrightarrow{O_1} \xrightarrow{O_2} \cdots \xrightarrow{O_\alpha} S_\alpha \xrightarrow{O_{\alpha+1}} \cdots \xrightarrow{O_l} B$$

L'essentiel est évidemment de "joindre les deux bouts" quelque soit la stratégie choisie.

L'ensemble des opérateurs est généralement fini. Dans ce cas, une méthode combinatoire permet théoriquement de trouver la solution si elle existe. Il suffit de fabriquer l'arborescence de toutes les situations possibles à partir de H en appliquant les opérateurs. Si B peut être obtenu à partir de H c'est à dire si la solutions existe on le trouvera dans cette arborescence.

...

Le schéma est le suivant (avec 3 opérateurs ne s'appliquant pas toujours):



S'il s'avère que S_{16}^3 est le but B alors la solution est donnée par le schéma

$$H \xrightarrow{O_2} S_2^1 \xrightarrow{O_3} S_7^2 \xrightarrow{O_2} B$$

cette méthode combinatoire présente de gros inconvénients :

L'espace de recherche (qui est l'ensemble des noeuds de cette arborescence) peut être infini ou si énorme qu'on ne peut le manipuler de façon raisonnable même avec un ordinateur puissant. Le problème des programmes de démonstration automatique ou de jeu est justement de limiter la croissance de cette arborescence. Pour cela on utilise des heuristiques qui donnent des indications pour savoir si telle branche à des chances de conduire à la solution. Les heuristiques doivent être bien choisies afin que l'arborescence reste de taille raisonnable et surtout qu'un noeud conduisant à la solution ne soit pas éliminé.

A chaque instant nous devons décider s'il faut ou non garder le noeud engendré. Les choix doivent être judicieux. C'est là qu'intervient le "flair". C'est là que nous entrons dans l'Intelligence Artificielle. Les heuristiques apparaissent comme un moyen d'orienter la recherche vers la solution.

...

Une heuristique sera donc bonne si fréquemment elle nous oriente vers la solution. Sa valeur est statistique. Il se peut qu'une heuristique bonne en général allonge la découverte d'une solution dans un cas particulier.

Par exemple : en calcul intégral une bonne heuristique pour intégrer un produit consiste à tenter l'intégration par parties (c'est préférable en général à toute manipulation algébrique : commuter le produit par exemple etc...).

On sait bien que ce n'est pas forcément la façon la plus rapide d'aboutir. De toute façon, l'utilisation d'une heuristique ne peut pas nous faire obtenir de résultat faux. Seulement si l'heuristique est "bonne", on peut gagner du temps ; si elle est mauvaise on peut aussi en perdre.

Il est intéressant en guise de conclusion de reprendre l'idée de Waldinger [II- 11] sur la possibilité d'écrire des programmes qui programment en se servant de programmes de démonstration automatique. Avec le formalisme précédent, écrire un programme, c'est résoudre le problème (D, R, I) où D sont les données, R les résultats du programme à écrire et I l'ensemble des instructions capable de faire passer des données aux résultats. Cette suite peut-être extraite d'une preuve constructive de la possibilité de résoudre le problème.

CHAPITRE II

LA METHODE DE SIKLOSSY

Nous allons dans ce chapître présenter les traits essentiels d'une méthode de démonstration proposée par L. Siklossy et V. Marinov au second IJCAI de Londres en 1971 (voir bibliographie). Les principaux avantages seront dégagés ensuite.

L. Siklossy et V. Marinov, basent leur méthode en grande partie sur le développement systématique de l'arborescence. Ils ont voulu prouver que dans certains cas les techniques heuristiques pures n'étaient pas nécessaires : ce sont des armes puissantes qu'il faut conserver pour les problèmes délicats. Dans bon nombre de problèmes où elles ont été utilisées, elles ne sont pas absolument indispensable.

DOMAINE D'UTILISATION DE LA METHODE

Pour appliquer leur méthode, Siklossy - Marinov disposent d'un certain nombre de règles de réécriture de la forme <G> : = <D> (<gauche> se réécrit <droit>) . Les deux membres de la règle sont des expressions. Un théorème a la forme d'une règle de réécriture (par exemple (a + b) + c : = (a + (b + c)).

Notation : Nous noterons G $\stackrel{?}{=}$ D le théorème à démontrer. Le signe $\stackrel{?}{=}$ indiquant que l'égalité reste à montrer.

Prouver le théorème consiste à établir la suite des règles qu'il faut appliquer au membre de gauche pour obtenir le membre de droite.

Avec le symbolisme vu au chapitre précédent, le problème a la forme (H, B, O) où H est le membre gauche de la formule, B le membre droit, O l'ensemble des opérateurs (règles de réécriture).

Une règle s'applique sur une formule partout où le membre gauche de la règle peut s'unifier. Le terme s'unifiant avec le membre gauche est alors remplacé par le membre droit (modifié par l'unification) de la règle.

...

exemple d'application d'une règle sur une formule

règle	+ U V : = + V U	[commutativité de +]
formule	* + X + Y * Z T 3	[(x + (y+zt)) * 3]
	1 2 3 4 5 6 7 8 9	

. En 1 l'unification échoue (connectives différentes)

. En 2

Formule * ~~+~~ ~~X~~ ~~+ Y * Z T~~ 3
G ~~+~~ ~~U~~ ~~V~~
D + ~~V~~ ~~U~~
+Y*ZT X

Résultat * + +Y*ZT X 3

. En 3 échec connective dans règle_variable dans la formule

. En 4

Formule * + X ~~+~~ ~~Y~~ ~~* Z T~~ 3
G ~~+~~ ~~U~~ ~~V~~
D + ~~V~~ ~~U~~
*ZT Y

Résultat * + X + *Z T Y 3

. En 5, 6, 7, 8, 9 échec de l'unification

RECHERCHE SYSTEMATIQUE

A - Règles non expansives

Dans un tel système, un théorème a la forme G : = D où G et D sont les membres gauche et droit. La recherche systématique de la preuve commence avec un espace de recherche réduit à l'expression G. De G, nous fabriquons le niveau suivant de l'arbre de recherche de la façon suivante : nous appliquons à G toutes les règles de réécriture, l'une après l'autre. L'expression résultant de l'application d'une règle à un noeud est insérée si et seulement si elle ne figure pas déjà dans l'arbre.

Exemple axiomatique 1

. Règles
R 1 A + B : = B + A
R 2 A + (B+C) : = (A+B) + C
R 3 (A + B) - B : = A
R4 (A+ B) - C : = (A - C) + B

. Théorème à démontrer

(A + B) + C : = A + (B + C)

...

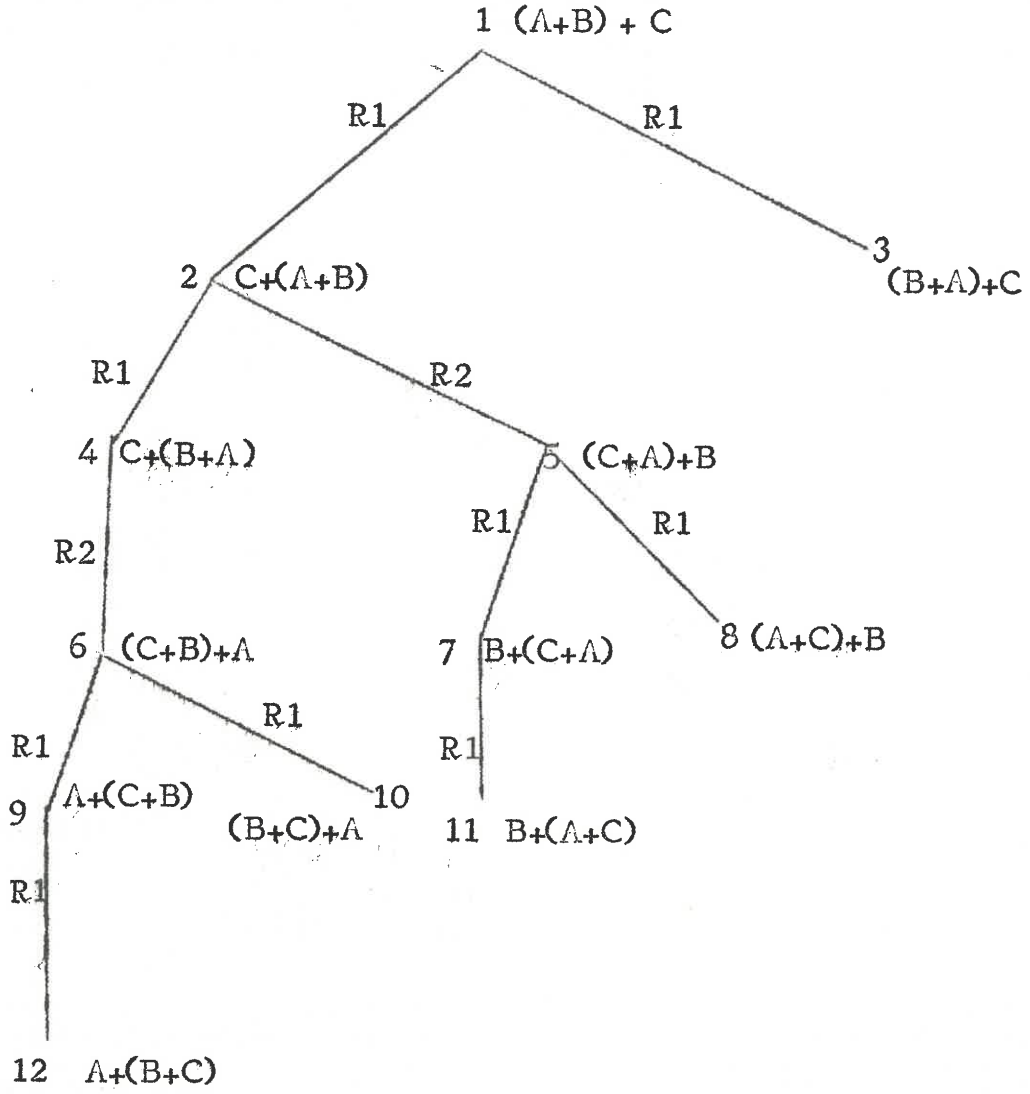


Fig. 1

On appelle noeud mort un noeud qui ne peut générer aucune nouvelle expression par application des règles (dans l'exemple précédent 3 et 8 sont des noeudsmorts).

La recherche s'arrête quand l'expression A + (B + C) est trouvée au cinquième niveau (noeud 12) . On peut remarquer qu'une fois le résultat obtenu, il est très facile d'extraire la démonstration et la suite des étapes intermédiaires.

Sur la fig. 1 on a immédiatement la preuve.

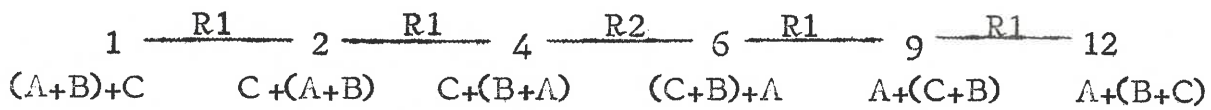


Fig. 2

...

B - Règles expansives

Dans l'axiomatique 1, la règle R 3 est dite simplifiante. Son application peut présenter l'intérêt de faire disparaître tout un terme d'une formule (ce qui peut amener un allègement substantiel de l'expression à manipuler).

Dans la pratique nous pouvons avoir à utiliser de telles règles dans l'autre sens, de façon à faire intervenir un terme que l'on souhaite voir dans le calcul (en trigonométrie il peut être intéressant de remplacer 1 par $\sin^2 u + \cos^2 u$ (en choisissant bien u) . En calcul intégral on peut avoir à remplacer x par x + a - a etc... Vous savez - tous ces "trucs" qui font dire aux élèves : "ce n'est pas difficile mais il fallait y penser ! ").

Soit par exemple la règle R'3 A : = (A + B) - B . Cette règle s'applique sur tout terme d'une formule. A s'unifie en effet avec tout terme. Le terme B qu'introduit R'3 peut être quelconque. Aussi, une règle comme R'3 génère une infinité de noeuds si l'on ne prend pas de précautions particulières.

On appelle règle expansive une règle dont l'application répétée peut engendrer un espace de recherche infini. Sont de ce type, en particulier , les règles qui introduisent une expression arbitraire.

Exemples
1 A : = (A + B) - B
2 O : = A - A
3 A : = A + O
4 A > B : = (A - B)> O

Les deux premiers exemples génèrent un espace de recherche localement infini (un noeud pouvant donner une infinité de successeurs) . Les deux exemples qui suivent donnent un espace de recherche infini mais qui est localement fini. Un noeud donne un nombre fini de successeurs mais la suite des applications possibles est infinie.

Par exemple, avec R 3 appliqué à x on peut obtenir

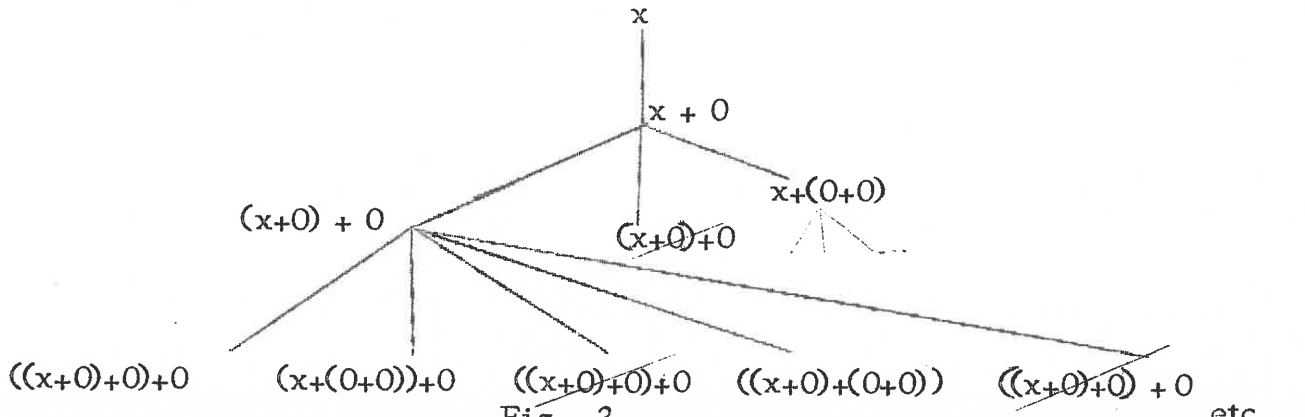


Fig. 3

Nous distinguerons désormais deux types de règles expansives :

- Les règles expansives atomiques qui ont pour membre gauche une simple variable.

- Les règles espansives non atomiques (par exemple : $A > B := (A - B) > 0$).

En général le problème de savoir si une règle de réécriture a un espace de recherche infini est indécidable. Dans le domaine qui nous intéresse, une règle R sera dite expansive si elle s'applique sur son propre second membre (c'est à dire si le membre de gauche peut s'unifier avec le membre de droite) et si l'expression obtenue en appliquant R à son membre de droite a plus de connectives que le membre de droite lui-même.

Dans ce domaine, voir qu'une règle est expansive est donc une tâche aisée.

Si le système de règles présente des règles expansives, la recherche systématique s'effectue de la façon suivante.

- D'abord, toutes les règles non expansives sont appliquées au membre gauche du théorème à démontrer. Si le membre droit est retrouvé, le système s'arrête. Sinon, tous les noeuds de l'arbre sont morts puisque par principe une règle non expansive ne peut générer qu'un espace de recherche fini. Jusqu'à ce stade les noeuds de l'arbre sont engendrés par niveau.

Pour chaque noeud ainsi engendré l'application de chaque règle expansive (comme ce sera décrit ultérieurement) ajoute des noeuds nouveaux à l'arbre.

- Nous nous limitons ensuite à n'utiliser à nouveau que les règles non expansives. Les règles expansives ne sont appelées que lorsque toutes les autres échouent. En cas d'application, elles le sont avec le maximum de précautions. La raison en est simple : les règles expansives augmentent considérablement la taille de l'arborescence.

Exemple Axiomatique 2

règles	R 1	$P \cdot Q := Q \cdot P$
	R 2	$P \cdot (Q \cdot R) := (P \cdot Q) \cdot R$
	R 3	$P \cdot (P \rightarrow Q) := Q$
	R 4	$\neg Q \cdot (P \rightarrow Q) := \neg P$

...

R 5 $\quad \neg\neg P := P$

R 6 $\quad P := \neg\neg P$ (expansive)

Théorème à démontrer

$(S \rightarrow \neg R) . R := \neg S$

L'arborescence obtenue est la suivante

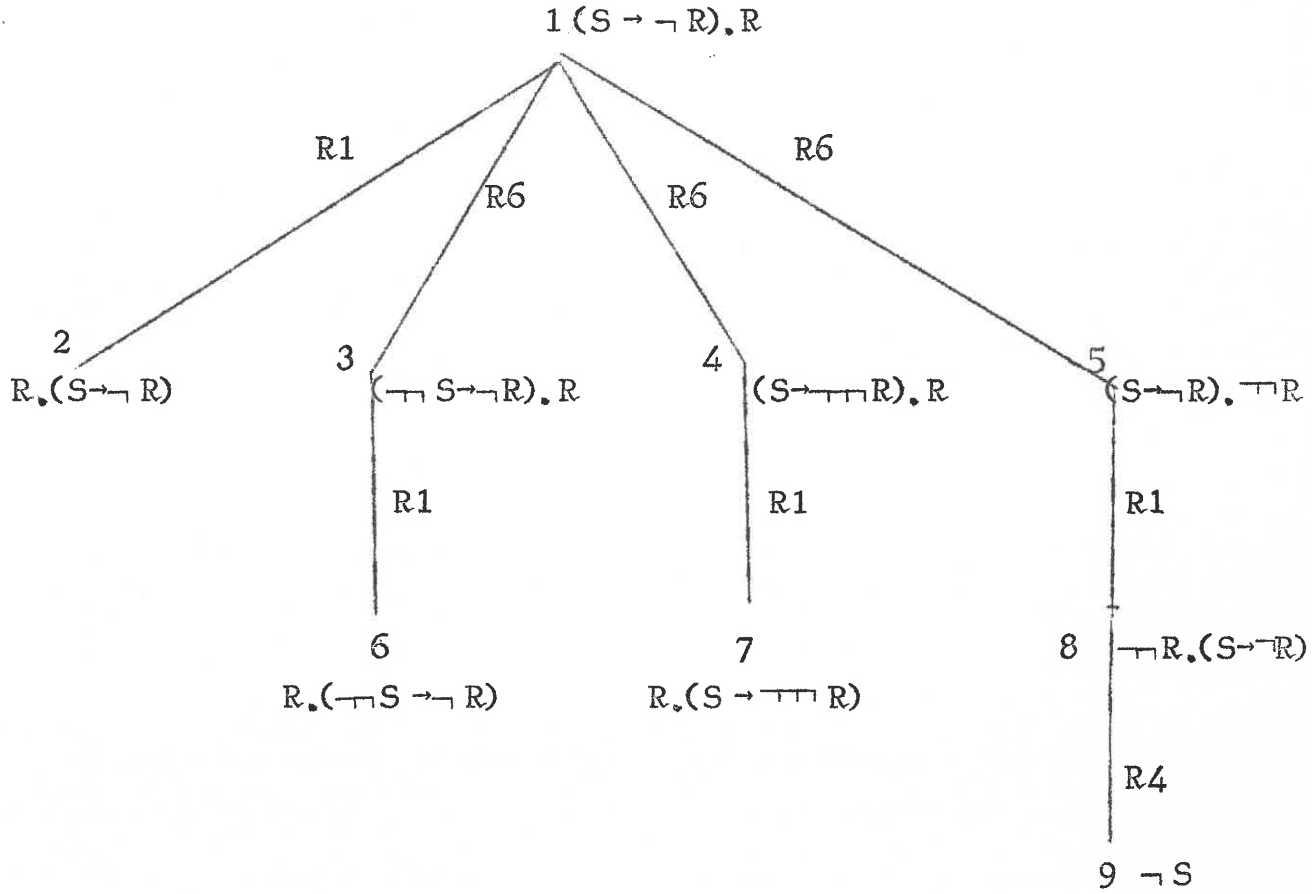


Fig. 4

Après application de la règle R 1 au membre de gauche, les deux noeuds 1 et 2 sont morts. L'unique règle expansive R 6 est appliquée au premier noeud. On obtient ainsi les noeuds 3, 4, 5 . On s'interdit alors l'utilisation de R 6 et les règles non expansives donnent rapidement la solution au noeud 9 .

La situation est plus complexe pour les règles expansives à espace de recherche localement infini qui introduisent des expressions arbitraires. Il est possible de se restreindre à celles légalement construites à l'aide des symboles trouvés dans le théorème à démontrer. Ce n'est pas là une restriction sévère. On peut noter que c'est le seul moment où le membre de droite du théorème à démontrer intervient pour autre chose que pour arrêter la recherche . Autrement cette recherche est complètement aveugle et le membre droit n'intervient pas pour l'orienter.

...

La stratégie consiste à faire croître l'arbre aussi lentement que possible. On peut atteindre ce but en limitant les expressions auxquelles on applique les règles expansives atomiques et les expressions qui peuvent être substituées aux variables arbitraires dans les règles à expansion localement infinie. La croissance de l'arborescence s'effectue en plusieurs passages. A chaque passage, les limitations sur l'utilisation des règles expansives sont moins sévères qu'au passage précédent.

Dans un premier passage, les règles expansives atomiques (qui pourraient être appliquées à toute expression ou sous-expression d'un noeud) ne sont appliquées qu'aux variables, constantes et connectives unaires. Il n'y a pas de restriction pour les règles non atomiques. Pendant ce premier passage nous limitons le choix des expressions pouvant être substituées aux variables arbitraires des règles à expansion localement infinie. On peut se limiter dans un premier temps à substituer uniquement des variables , des constantes ou des fonctions unaires.

Si le théorème n'est pas prouvé pendant le premier passage, les restrictions précédentes peuvent être levées graduellement.

On pourrait penser que dans cette méthode, on utilise des heuristiques. En fait, on n'utilise qu'une stratégie dans la mesure où l'ordre dans lequel les règles sont appliquées est prédéterminé et n'est pas modifié pendant les démonstrations.

RESULTATS DE LA METHODE

Cette méthode a été comparée avec le système FDS de Quinlan et Hunt (qui traite de problèmes bien adaptés à la méthode que l'on vient de décrire), avec le système REF-ARF de Fikes (qui résoud des problèmes (singes et bananes, missionnaires et cannibales, bidons d'eau)) enfin avec le programme de CHANG qui traite des problèmes identiques à ceux de Quinlan par une méthode basée sur la résolution. Il semble que la méthode de Siklossy prenne l'avantage en étant de 25 à 40 fois plus rapide que les méthodes avec lesquelles elle a été comparée.

Les problèmes traités sont de difficulté inégale , mais il semble qu'actuellement la méthode n'a été testée que sur des problèmes ayant au plus une trentaine de règles. Les règles et les théorèmes à démontrer n'étant pas des chaînes de symboles "très lourds" à manipuler. Les expressions proposées par Siklossy-Marinov semblent avoir rarement plus de

...

quatre noeuds, rarement plus de deux connectives différentes. La longueur des chaînes de symboles semble toujours inférieure à 20 (très souvent inférieure à 10). La profondeur des démonstrations reste de l'ordre de 5.

Les problèmes ne sont pas faciles pour autant (bon nombre d'étudiants de 1er cycle d'université seraient arrêtés devant beaucoup de ces problèmes) mais nous sommes loin de la manipulation algébrique complète dont nous pouvons avoir besoin pour résoudre des problèmes du type de ceux que nous avons envisagés. (pour manipuler des dérivées $n^{ième}$ de fonctions trigonométriques, il faut disposer de beaucoup plus de règles et les formules sont beaucoup moins maniables).

Cette méthode de recherche systématique aveugle, surprend par ses performances. Cependant Siklossy-Marinov loin de rejeter les méthodes heuristiques habituelles en intelligence artificielle mettent beaucoup d'espoir dans la combinaison de ces deux types de méthodes pour réaliser des systèmes puissants.

Un grand intérêt de cette méthode est qu'on peut disposer facilement des différentes étapes de la démonstration une fois le résultat établi. Pour les problèmes que nous avons envisagé, cela a peu d'importances mais pour certains problèmes , on sait que l'on a besoin de preuves constructives. Il semble que là encore la méthtode de Siklossy prenne le pas sur d'autres méthodes (basées sur la résolution où il semble plus délicat de dégager la construction du résultat).

CHAPITRE III

QUELQUES AMENAGEMENTS ET PROLONGEMENTS DE LA METHODE DE SIKLOSSY

Nous avons rappelé dans le chapître précédent la méthode de Siklossy, vu le type de problèmes qu'elle traite et donné un aperçu de ses performances. Nous allons maintenant voir qu'un certain nombre d'aménagements sont possibles pour traiter les problèmes envisagés dans le cadre de ce travail. Nous verrons dans un premier temps comment faire intervenir le membre droit du théorème que nous voulons démontrer. Nous verrons ensuite comment il est possible d'augmenter la profondeur d'une démonstration en itérant la méthode de Siklossy sur des sous problèmes du problème posé. La partie suivante introduira un type de règles intéressant par lui-même en intelligence artificielle. Enfin nous verrons comment il est possible de "s électer" les règles qui interviennent au cours d'une preuve. Pour un programme qui a à sa disposition un grand nombre de règles très différentes, ce sera là une façon de s'adapter au problème traité.

I - Intervention du membre droit au cours de la preuve

Nous avons vu qu'un problème pour la méthode de Siklossy se présente comme la donnée d'un ensemble fini de règles et d'un théorème à démontrer.

règles $i = 1, p$ $\left\{ R_i \qquad G_i := D_i \right.$

théorème $G := D$

Résoudre ce problème consiste alors à engendrer avec les R_i l'arborescence de G afin d'y retrouver D; Ce qui pourrait correspondre au

...

schéma suivant :

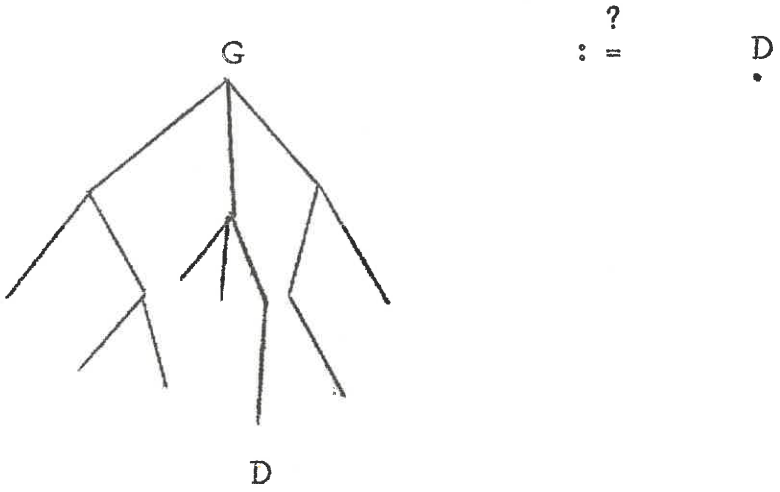


Fig. 1

La recherche est aveugle et ne fait intervenir D qu'au moment du test pour savoir si D est retrouvé.

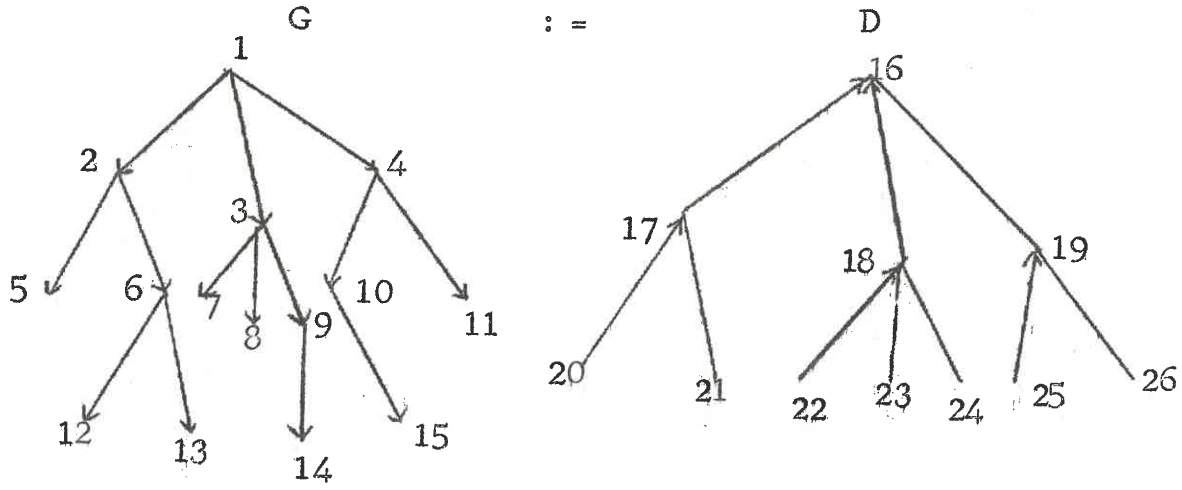
Avec cette méthode, il se peut que l'arborescence soit très grande, ou tout du moins, si l'on manipule de grosses expressions que l'espace de travail pour stocker les formules "accrochées" à chaque noeud soit saturé par un grand nombre de formules écrites "à l'aveuglette". Il se peut aussi que pour certaines d'entre elles nous ne soyons "pas très loin" de D mais qu'il manque à D juste "une petite simplification " pour être reconnu. Ce phénomène peut se produire si le problème est mal posé (chose possible lorsque c'est un programme qui pose le problème comme c'est le cas dans la vérification de formules par récurrence).

Uneidée intéressante consiste à faire intervenir D plutôt que s'éterniser dans le développement de G . Pour cela, on conserve le développement déjà fait de G et en appliquant les règles de la droite vers la gauche, on établit une arborescence de conjectures possibles pour établir D. Le principe du développement est le même que pour G. Chaque conjecture n'est inscrite que si elle ne figure pas déjà dans l'arborescence D. Mais maintenant, au lieu de vérifier si une conjecture coïncide avec G (ce qui suffit pour prouver le théorème), on compare chacune des conjectures aux noeuds de l'arborescence de G. Si l'on peut trouver dans l'arborescence issue de G, la conjecture que l'on est en train d'émettre, alors le problème

...

est terminé puisque l'arborescence D nous montre comment D se déduit de la conjecture et l'arborescence G donne la démonstration de cette conjecture.

Nous travaillons alors avec le schéma suivant :



arborescence des dérivés de G . Le chemin en trait gras donne la preuve de la conjecture

arborescence des conjectures possibles pour atteindre D. Le chemin en trait gras montre comment le résultat est obtenu à partir de la conjecture.

Fig 2

Un exemple de preuve serait si (14)et (22) étaient identiques

Cas particulier où la relation : = est symétrique

En manipulation algébrique courante, très souvent la relation "se réécrit" se réduit à l'égalité, Elle est alors symétrique. Nous disposons dans ce cas de quelques propriétés particulières intéressantes.

Il n'y a plus à tenir compte du sens des flèches sur les arborescences G et D. L'arborescence G (resp. D) met en évidence une classe d'expressions égales à G (resp. D) Il suffit de trouver un élément dans l'intersection de ces classes.

...

Dans ce cas, il n'y a plus aucune raison de privilégier le rôle de G par rapport à celui de D et l'intervention de D dans le déroulement de la recherche se justifie pleinement. Un complément intéressant consisterait même à déterminer à l'aide d'heuristiques lequel des deux membres on doit développer le premier. Il est probable qu'avec quelques heuristiques simples, on puisse améliorer les performances du programme.

Exemple

Règles = + U V + V U
= + - U V W - + U W V

démontrer = + Λ - BC - + A B C

1 + Λ - BC (? = - + Λ B C)
1 |
2 + - B C Λ
2 |
3 - + B Λ C
1 |
4 - + A B C

Fig. 3

Avec les mêmes règles démontrer le théorème symétrique.

= - + Λ B C + Λ - B C

1 - + Λ B C
1 ↓
2 - + B Λ C

noeud mort

3 + Λ - B C
1 ↑
4 + - B C Λ
2 ↑
5 - + B Λ C

Fig. 4

Il faut bien voir que, ici, la preuve consiste en le chemin (1) (5) (4) (3) et que les passages (5) (4) puis (4) (3) font intervenir les règles de la droite vers la gauche ; ceci n'est légal que parce que l'on manipule des égalités. (Il faut remarquer qu'avec une réécriture dissymétrique cette

...

méthode pourrait, au lieu de conclure à l'échec de la démonstration, indiquer quelles sont les règles à symétriser pour obtenir le résultat cherché. Ce peut-être une information importante.

Cas des règles expansives

Remarquons tout d'abord qu'une règle qui est expansive de la gauche vers la droite est simplifiante de la droite vers la gauche.

Exemples

$A := (A + B) - B$	$(A + B) - B := A$
$P := \neg\neg P$	$\neg\neg P := P$
$a > b := (a - b) > 0$	$(a - b > 0) := a > b$

Dans le cas où : = est l'égalité, pour toute règle on dispose de G : = D et D : = G . On peut alors se libérer des règles sous leur forme expansive et ne conserver que la forme simplifiante (beaucoup plus facile à manipuler). Comment interviendra alors l'expansion qui peut être nécessaire au cours d'une preuve ? Et bien sous forme de simplification dans l'arborescence D. Comme au moment de la sortie de la démonstration, dans l'arborescence D, le chemin est lu d'une feuille vers la racine, la règle se trouve considérée sous sa forme expansive.

Exemples

1 - Règles (1) = + U V + V U [U + V = V + U]

(2) = - + U V V U [(U + V) - V = U]

(3) = + - U V W - + U W V [(U - V) + W = (U + W) - V]

Démontrer

= A + - A B B [(A - B) + B = A]

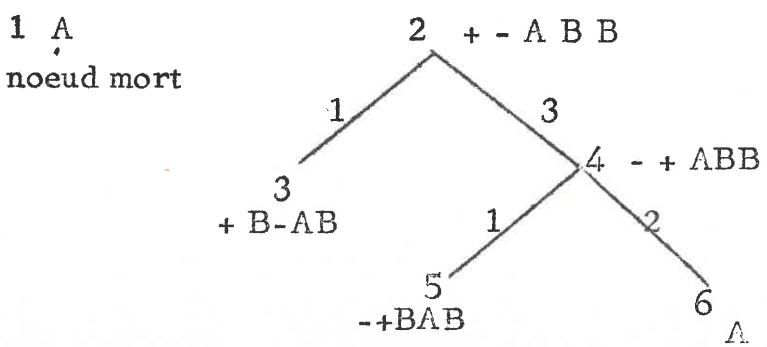


Fig. 5

La règle 2 est simplifiante de la gauche vers la droite et expansive de la droite vers la gauche. Dans l'arborescence de droite, elle est utilisée sous sa forme simplifiante mais dans la preuve $1 \xrightarrow[(G)]{2} 4 \xrightarrow[(D)]{3} 2$ elle est considérée sous sa forme expansive.

2- règles (1) = - U U 0 [U - U = 0]
(2) = + U 0 U [U + 0 = U]
(3) = - + U W V + - U V W [(U+W)-V = (U-V)+ W]
(4) = + U V + V U [U + V = V + U]

Démontrer

= - + A B C + - + A B C - C C

[(A+B)-C =((A+B)-C)+(C-C)]

On obtient

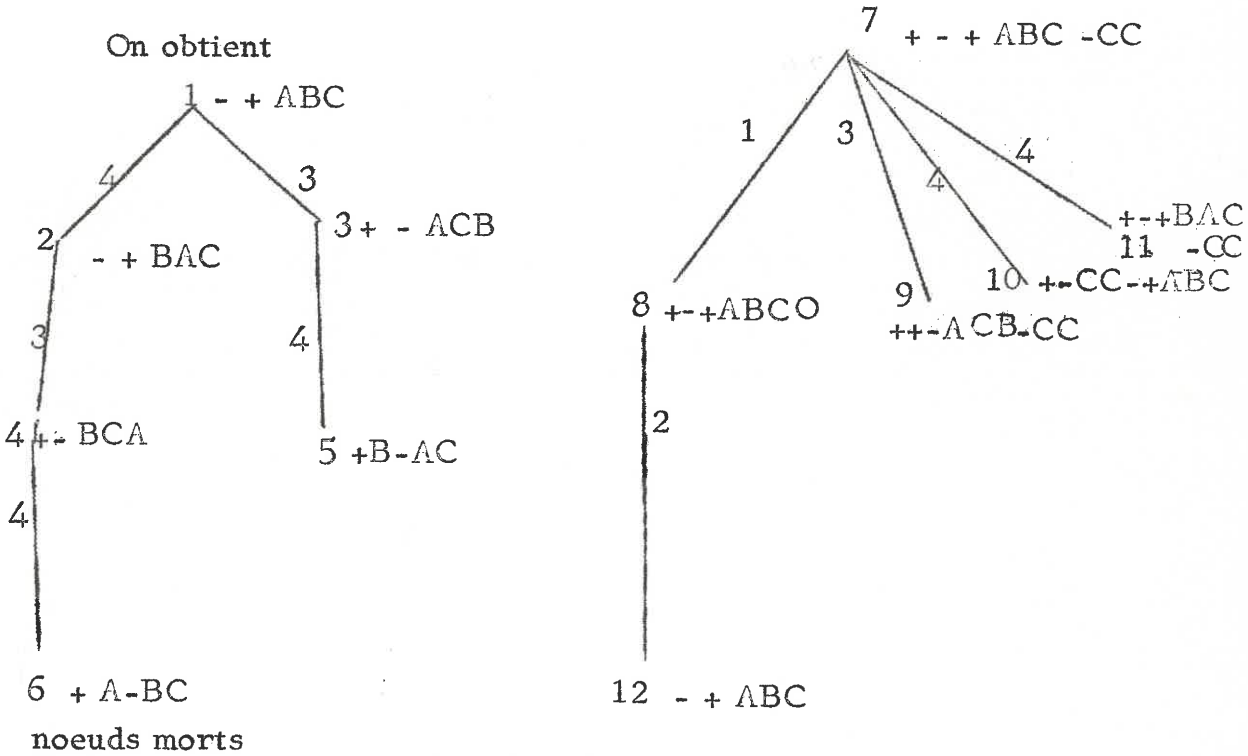


Fig. 6

nous sommes là dans le cas où le terme - C C pourrait causer beaucoup d'ennuis si nous ne pensions pas à développer D.

Il est intéressant de remarquer que faire intervenir le membre droit nous rapproche du comportement du mathématicien face à ce type de problèmes. Il n'est en effet pas du tout naturel pour un humain de travailler de façon aveugle sur le seul membre gauche de l'égalité qu'il veut démontrer. Le

...

membre droit doit rester le point de mire. On doit le garder constamment en vue. Il peut difficilement être cette expression sur laquelle on tombe au hasard en pensant à G.

II - Redémarrage du problème

Dans le paragraphe précédent, nous avons vu comment le fait de faire intervenir le membre droit du théorème à démontrer pouvait améliorer les performances de la méthode. Seulement nous avons là besoin de deux arborescences qui peuvent chacunes prendre beaucoup de place. De toute façon, si puissante que soit la machine, si grand soit l'espace de travail dont on dispose, nous serons toujours limités. Il est alors intéressant d'essayer de dégager des méthodes permettant de dépasser cette limite.

Dans la méthode qui nous intéresse, supposons l'espace de travail trop petit pour faire le développement souhaité des arborescences. La réaction la plus simple consiste à dire qu'il y a eu échec pour le problème dès que l'on constate le débordement, ce qui ne veut pas dire que l'expression proposée n'est pas un théorème dans le système donné. (En fait, on peut affirmer que l'on n'a pas affaire à un théorème uniquement lorsque les deux arborescences G et D sont complétement développées, mortes et qu'elles n'ont aucun sommet commun).

Dans le cas où le blocage a lieu uniquement pour des questions de place, et que des règles seraient encore applicables, une idée possible consiste à déterminer avec les deux arborescences G et D un couple de noeuds (MING, MIND) pour lequel une évaluation de distance entre les formules de G et de D soit minimum. Il suffit alors de reposer au programme le sous problème

$$\text{MING} \stackrel{?}{=} \text{MIND}$$

que l'on essaie de résoudre en écrasant les arborescences issues de G et de D. C'est ce que nous avons appelé "redémarrage du problème".

Ce procédé peut évidemment se poursuivre tant que le problème n'est pas terminé.

...

à
Pour le problème dans son entier, ceci revient travailler suivant le schéma

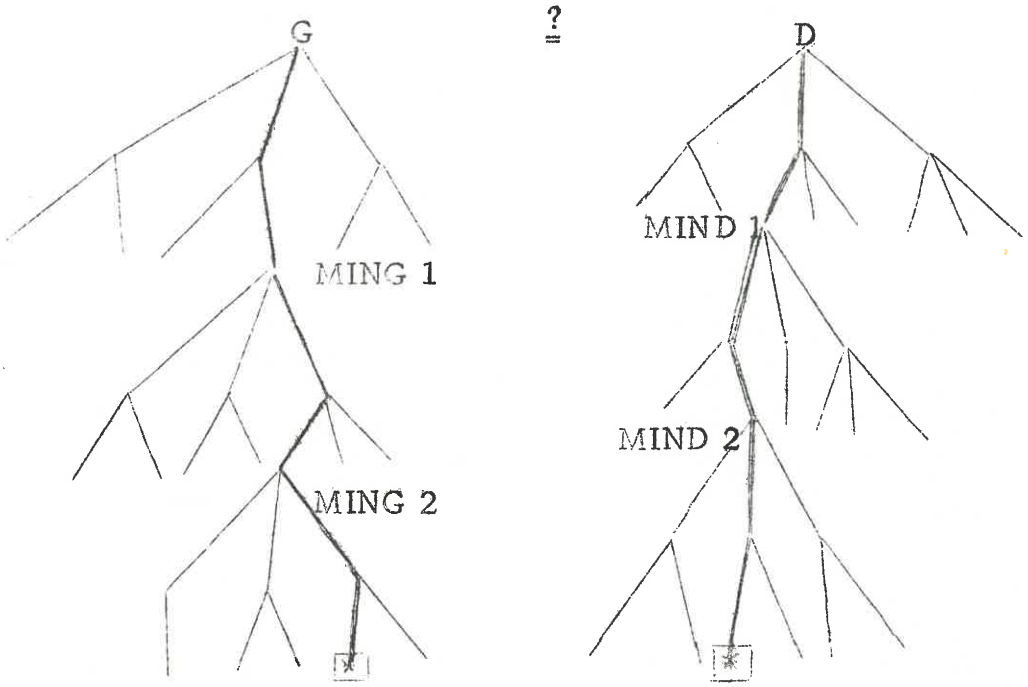


Fig. 7

Nous avons ici un redémarrage en (MING 1 $\stackrel{?}{=}$ MIND 1) et en (MING 2 $\stackrel{?}{=}$ MIND 2) . Un certain nombre de remarques s'impose :

1) Ce procédé nécessite une fonction d'évaluation capable de mesurer la distance qui sépare deux formules. (Il ne faut pas penser ici distance au sens des espaces métriques mais accepter le mot comme intuitif). Nous parlerons de cette fonction dans un paragraphe suivant.

2) L'arborescence D n'a pas besoin d'être développée et le procédé peut s'appliquer directement à la méthode de Siklossy vue précédemment. Le schéma est alors le suivant :

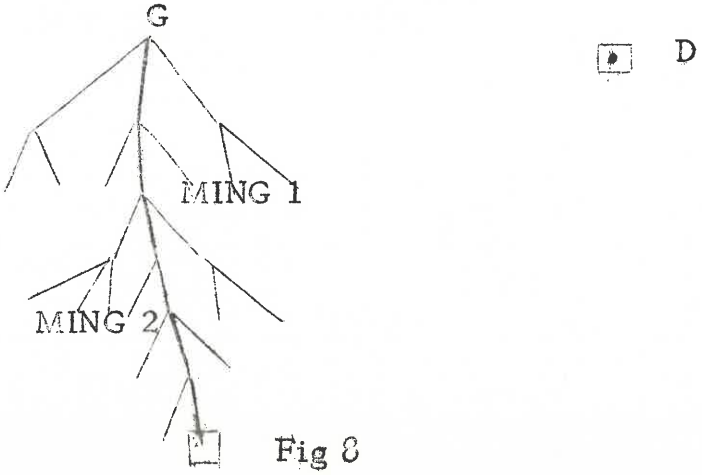


Fig 8

...

Le redémarrage consiste alors seulement à repartir du sommet MING qui semble le plus proche de D (pour la fonction d'évaluation).

3) Cette méthode permet d'atteindre avec des arborescences moins volumineuses des profondeurs de démonstration plus intéressantes. On peut même limiter artificiellement les arborescences et décider de ne travailler qu'avec moins de **s sommets** . (s étant un seuil qu'il serait peut-être intéressant d'optimiser). Ceci a été réalisé dans le programme décrit en 4 ème partie en prenant pour les arborescences G et D des seuils différents. Cela peut amener un gain de temps assez important par la diminution du nombre d'appels à la fonction d'évaluation pour déterminer (MING, MIND). Pour un nombre total donné ($s_G + s_D$) de noeuds, le nombre d'évaluation de distance à effectuer (S_G S_D) est maximum lorsque les deux nombres sont égaux. (Pour 100 noeuds, les seuils 60 - 40 pour les arborescences gauches et droites nous ont paru convenables.

4) Pour limiter le nombre d'évaluations de distances on a la possibilité de n'évaluer les distances qu'entre des feuilles. C'est en effet sur ces étapes les plus profondes qu'il est naturel de compter le plus. Par exemple; prendre comme MING un noeud au coeur de l'arborescence donne après le redémarrage au moins toute la sous-arborescence qui avait ce noeud pour racine (D'où perte de temps puisqu'on recalcule une zone qui avait déjà été calculée).

Nous aurions alors le schéma suivant

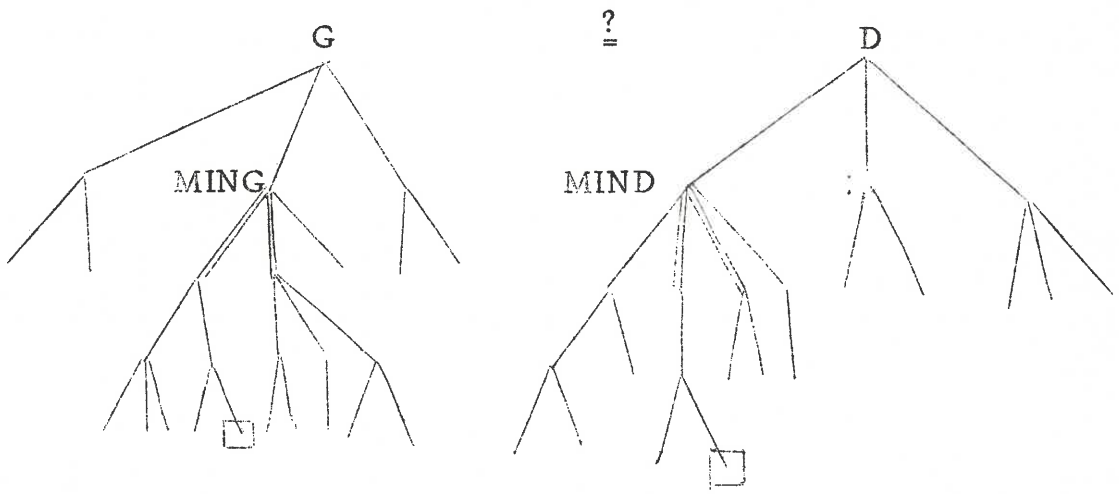


Fig. 9

...

On a même intérêt à ne prendre en considération que les feuilles les plus profondes et de ne pas tenir compte des feuilles "mortes".

5) Après un redémarage, comme nous écrasons l'arborescence du stade précédent nous ne disposons plus des formules et il peut se faire que nous retrouvions après le redémarrage des formules déjà vues et jugées non intéressantes par la fonction d'évaluation. Ce phénomène est difficile à controler puisque nous ne disposons plus en mémoire des éléments nécessaires aux comparaisons (ce qui fait que dans la figure 9, MING, par exemple, peut avoir plus de fils après le redémarage qu'avant ; Il peut avoir pour fils supplémentaire une formule qui n'avait pas été inscrite dans l'arobrescence G à cet endroit parce qu'elle figure déjà ailleurs et qui est reprise dans l'arborescence MING puisqu'elle n'y figure pas encore). Il y a indiscutablement une perte de temps ici.

Il y a plus grave puisque avec ce phénomène, on peut arriver à des bouclages : cela se produit si, par exemple, après un redémarage, on retrouve dans l'arborescence MING comme fils des antécédents du noeud MING dans l'arborescence G ; c'est un cas, où la démonstration "repart en marche arrière". Ce peut être intéressant pour rectifier les imperfections de la fonction d'évaluation.

Exemples en n'évaluant que sur les feuilles les plus profondes

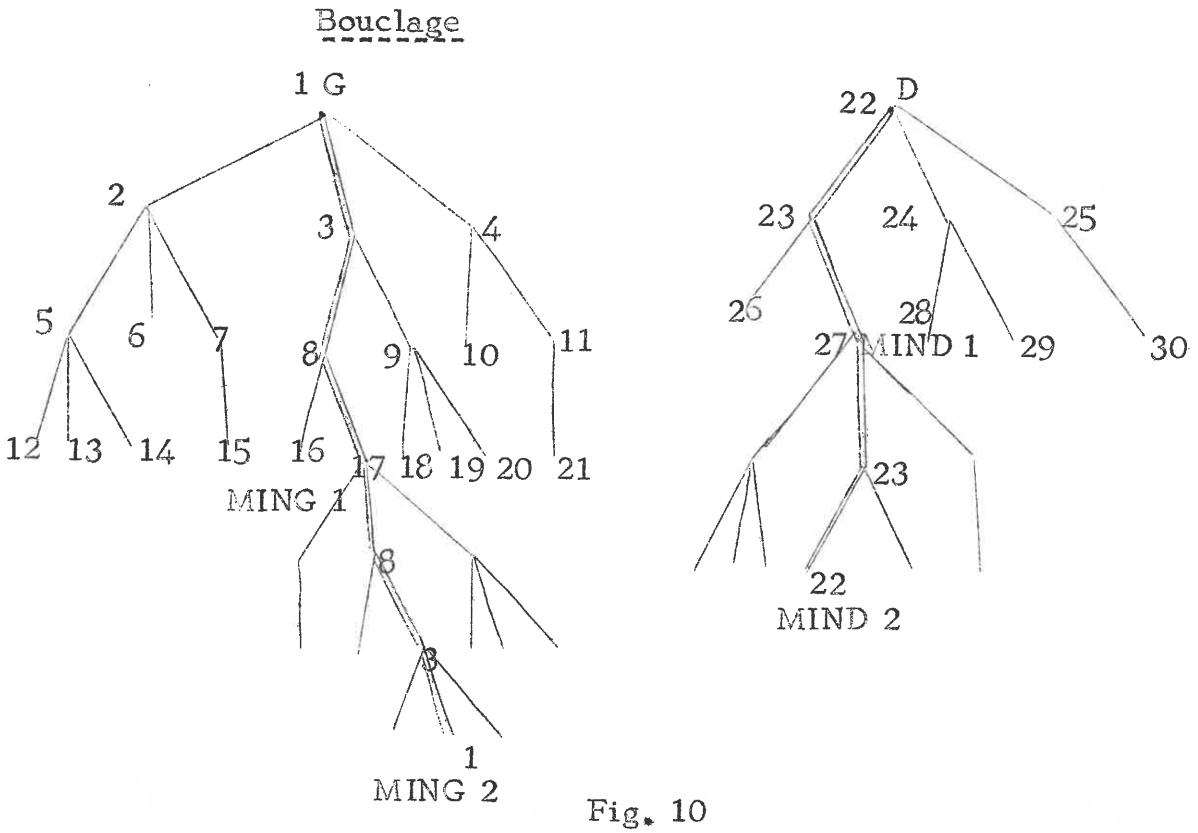


Fig. 10

...

Ce phénomène s'est produit très rarement et on n'a pas prévu de dispositif pour l'éviter. Il semble suffisant de mémoriser quelques formules intermédiairespour voir si elles réapparaissent de façon cyclique. De toute façon, il serait possible de concevoir des dispositifs permettant d'éliminer les bouclages. On pourrait par exemple imposer que la distance minimum obtenue entre MING et MIND soit strictement décroissante d'un redémarrage au suivant .

Cas pouvant rectifier une imperfection de la fonction d'évaluation

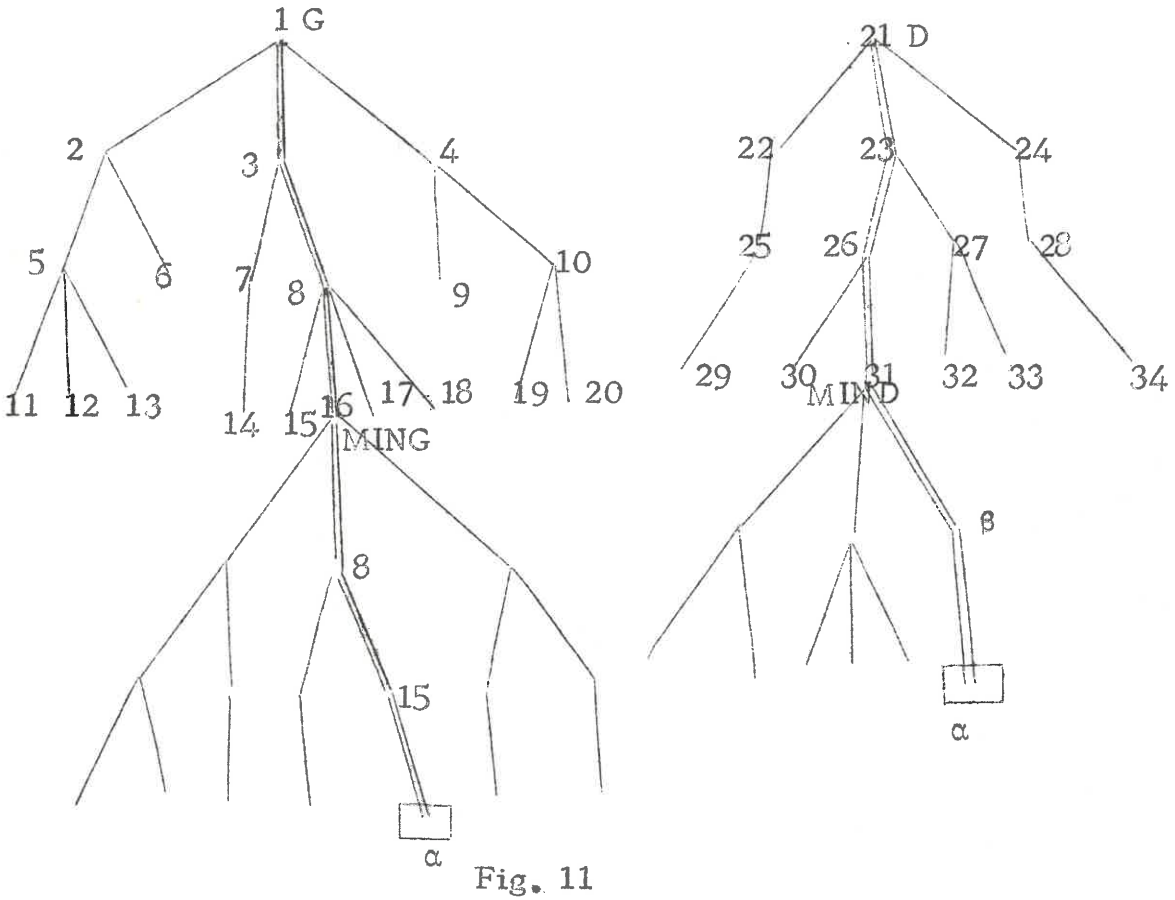


Fig. 11

La preuve est constituée par la suite.

MING

G——3——8——16——8——15——α

MIND

α——β——31——26——23——D

Nous avons succès grâce au retour arrière de 16 en 8 permettant de rattrapper 15 qui aurait pu être choisi comme MING si la fonction d'évaluation avait été plus perfectionnée. On ne doit pas être trop surpris de ce phénomène quand on sait combien il est délicat de bien régler une fonction d'évaluation . On peut d'ailleurs remarquer que là encore nous sommes assez proche du comportement de l'humain qui ne trouve pas toujours du premier

...

coup la ligne la plus directe d'une démonstration. La première évaluation n'est pas toujours la bonne. L'essentiel est sans doute de se permettre des retours en arrière.

6) Le nombre de tels redémarrages est en principe illimité tant que la liaison entre membre gauche et membre droit n'est pas établie. Pour éviter les cyclages dans le programme nous avons limité àtrois le nombre de redémarages permis dans chaque membre et nous concluons à l'échec si au bout de ce délai la preuve n'est pas achevée.

Composition de la fonction d'évaluation de distance entre deux formules

1) Complexité d'une formule

Avant de définir la distance entre deux formules, nous allons définir la notion de complexité d'une formule. Le problème est de savoir quels sont les éléments qui interviennent au moment où l'on apprécie la complexité d'une expression. Qu'est-ce qui peut nous faire dire que l'expression $x + y$ est moins complexe que $\sin^2 z + 2 y \cos x$ au seul vu de l'expression (indépendemment de tout contexte) ? On peut se rendre compte que les éléments suivant interviennent :

- nombre de symboles qui interviennent dans l'écriture (c'est à dire la longueur de la formule).
- la profondeur de parenthésage. Il semble que plus les parenthèses sont imbriquées, plus la formule est compliquée (par exemple : $\sin(x) + \sin(y) + \sin(z)$ semble plus simple que $\sin(x + \sin(y + \sin(z)))$.
- homogéneité de la formule? J'entends par là le nombre de connectives différentes qui interviennent dans la formule.

($\sin(x) + \sin(y) + \sin(z)$ peut paraître plus simple que $\sin x + \cos y * \mathrm{tg}\ z$ dans la mesure où elle fait intervenir moins de connectives différentes)

Dans notre programme nous avons utilisé la formule

compl = 3 . long + 4 . parent + 3 . Homog.

Certains critères comme l'apparition d'une loi pourraient intervenir dans la mesure de la complexité; par exemple :

$1 + x + x^2 + x^3 + x^4 + x^5$ est peut être très long mais c'est simple

...

dès que l'on a reconnu la somme des termes d'une progression géométrique. Nous n'avons pas utilisé de tels critères.

2) Distance entre deux formules

- Pour que deux formules soient proches elles doivent d'abord être de complexité comparable (attention aux compensations par exemple formules longues ayant peu de connectives différentes qui peuvent être de complexité comparable à des formules plus courtes faisant intervenir beaucoup de connectives différentes). La condition semble nécessaire sinon suffisante.

- Le nombre de connectives identiques en tête des formules intervient. (en notation polonaise bien sûr).

f_1	+	*	*	X	Y	Z	SIN	T	[(x y)z + sin (t)]
f_2	+	*	*	*	Y	Z	U	X SIN T	[(y z).u). x + sin (t)]
f_3	+	X		SIN	T				[x + sin (t)]
f_4	+	*	X	Y	SIN	T			[(x y) + sin (t)]

f_1 et f_2 peuvent être considérées comme proches dans la mesure où d'entrée avec + * * dans les deux formules on voit qu'il s'agit dans les deux cas d'une somme dont le premier terme est un double produit.

Par contre pour f_3 et f_4 on voit juste d'entrée qu'il s'agit d'une somme et que le premier terme n'est déjà pas de même nature.

- Comparaison des ensembles de connectives différentes. Soit C_1 (resp. C_2) l'ensemble des connectives intervenant dans f_1 (resp. f_2) ; la distance sera d'autant plus grande que la différence symétrique ($C_1 \Delta C_2$) sera "grosse". Je mesure ainsi le nombre de connectives qui sont dans f_1 et pas dans f_2 et réciproquement.

Exemple:

f_5	+	*	SIN x	COS y	tg	z
f_6	+	/	LOG x	EXP yRAC 2		

f_5 et f_6 doivent être considérées comme éloignées.

...

Dans le programme nous avons introduit une bonification supplémentaire afin d'écarter encore davantage deux formules qui n'ont pas la connective principale identique.

Ceci a été fait pour essayer de retenir comme MING et MIND des formules ayant même connective principale, ceci uniquement dans l'espoir de simplification du problème au moment du redémarage. (ce point sera repris ultérieurement).

La formule que nous avons utilisée est la suivante :

$$d(f_1, f_2) = 2 \cdot \begin{pmatrix}\text{valeur absolue de la} \\ \text{différence des complexités}\end{pmatrix} - 7 \cdot \begin{pmatrix}\text{Nbre connectives} \\ \text{identiques au départ}\end{pmatrix}$$

$$+ 3 \cdot (\text{card}(C_1 \Delta C_2) + \varepsilon \cdot$$

où $\begin{cases} \varepsilon = 10 & \text{si la connective principale est distincte dans les deux formules} \\ \varepsilon = 0 & \text{sinon} \end{cases}$

EVALUATION DE L'ETAT D'AVANCEMENT DES CALCULS

Les critères retenus jusqu'ici ne sont liés qu'à l'aspect des formules indépendamment de tout contexte d'utilisation. Or l'évolution d'un calcul est quelque chose de dynamique. Dans la mesure où nous avons toujours l'espoir que le calcul aboutisse rapidement, il semble intéressant dans notre problème de repartir de formules sur lesquelles "les calculs sont les plus avancés". (Ce qui revient à ne pas se contenter de la distance "à vol d'oiseau" entre les deux formules mais à essayer d'évaluer le chemin qui est à parcourir).

Le problème est donc de savoir quels sont les critères qui peuvent nous permettre de remarquer les formules dans lesquelles les calculs sont bien avancés. Ces critères dependent du contexte (ici le but à atteindre et les règles dont on dispose).

Pour ce qui est du but à atteindre, il est différent dans chaque problème et il semble difficile d'atteindre des résultats généraux intéressants. Remarquons cependant que dans l'évaluation de distance entre deux formules, faire intervenir le nombre de connectives identiques en tête de formule relève de cette idée. Si les deux formules n'ont pas même connective principale dans l'une des deux les calculs ne sont pas très avancés (puisque les calculs

...

nécessaires pour amener la "bonne " connective en tête n'ont pas été faits).

Pour ce qui est de l'avancement des calculs relativement à un jeu de règles données le problème semble plus simple.

Exemple 1 - Critère des règles privilégiées

Dans notre problème sur la récurrence une heuristique intéressante consiste à appliquer l'hypothsèe de récurrence (H R) le plus tôt possible (généralement elle ne sert qu'une fois. Mais elle sert toujours).

Au moment du redémarrage, toutes les feuilles qui ont été obtenues en utilisant H R à un moment ou à l'autre pourront être considérées comme ayant des calculs plus avancés que les autres.

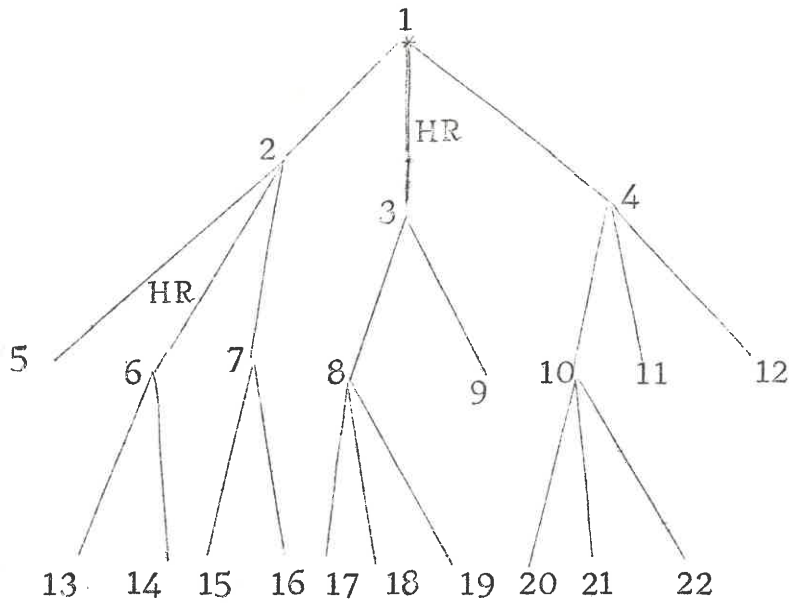


Fig 12

Les formules 20, 21, 22, 11, 12 pourront être considérées comme ayant des calculs moins avancés que celles qui sont en 13... 19.

Ce critère peut être utilisé chaque fois qu'interviennent dans le problème des règles privilégiées. C'est ainsi pour les problèmes sur les relations de récurrences où nous avons entre autre, les règles privilégiées

$$\begin{cases} U_o = a \\ U_n = f(U_{n-1}) \end{cases}$$

...

et où il faut alors prouver que $\forall n \; U_n = \varphi(n)$

Exemple 2 - Critères formels

Considérons le système suivant

R 1 : = + U + V W + + U V W [U +(V+W)=(U+V) + W]

Règles R 2 : = + / U V / W V / + U W V $\left[\frac{U}{V} + \frac{W}{V} = \frac{U+W}{V}\right]$

R 3 : = + U V + V U [U + V = V + U]

Théorème à démontrer

: = + / A D + / B D / C D / + + A B C D

$\left[\frac{A}{D} + \left(\frac{B}{D} + \frac{C}{D}\right) = \frac{(A+B+C)}{D}\right]$

Posons à 9 le seuil d'arrêt de développement de l'arborescence, on obtient

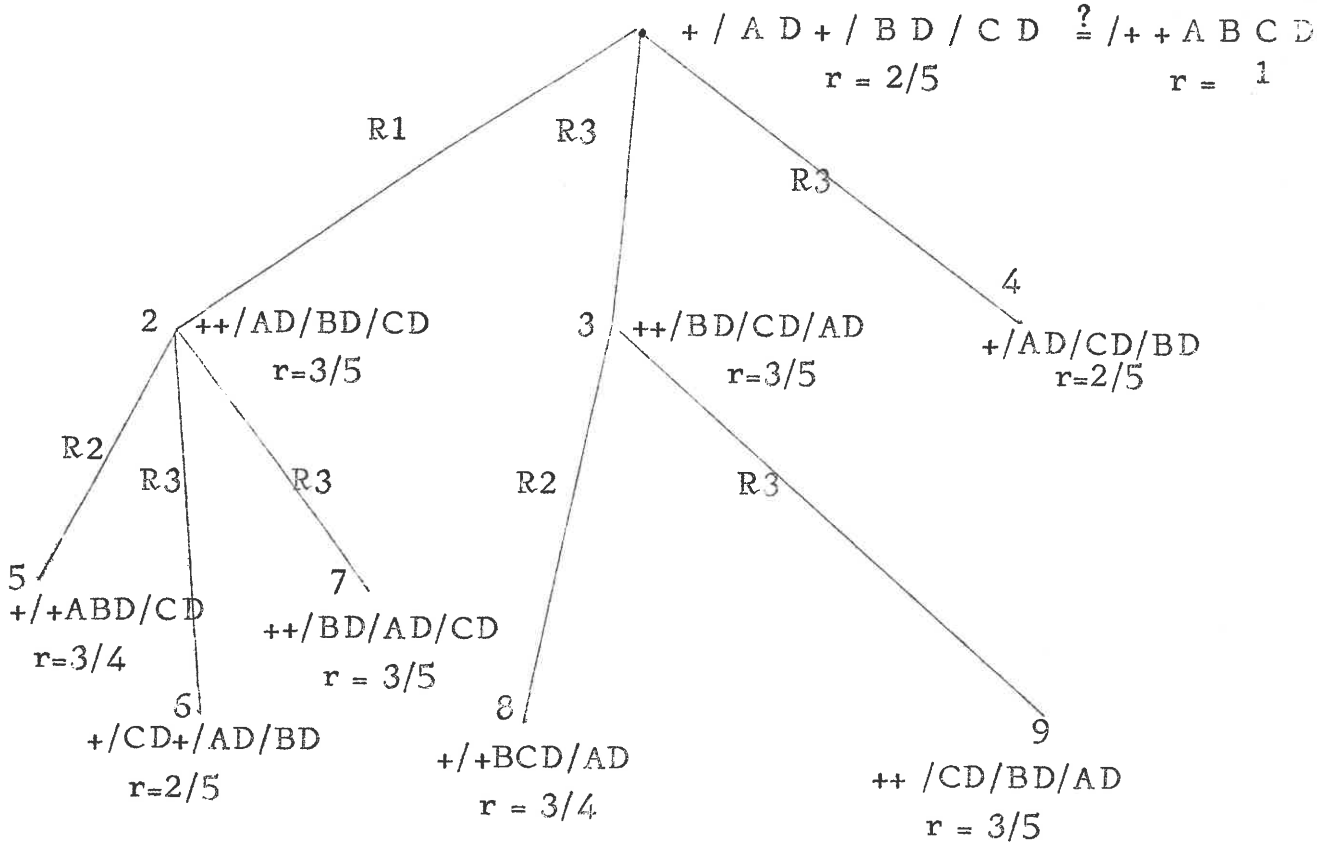


Fig. 13

...

(Quelles sont alors les formules dans lesquelles les calculs sont les plus avancés ?)

Si nous examinons le contexte règle, nous voyons que R 1 et R 2 sont des règles qui ramènent les connectives vers le début de la formule. La règle R 3 est neutre quant à ce critère. Dans le problème posé, le membre gauche à des connectives un peu partout. Le membre droit a toutes ses connectives en tête de formule.

Dans ce contexte nous pouvons définir pour chaque formule un rapport

$$r = \frac{\text{nb. de connectives en tête de formule}}{\text{nb. total de connectives}}$$

Il est alors possible de considérer que le calcul sera d'autant plus avancé que r sera proche de l'unité. (Pour le membre droit en effet, r vaut 1).

Sur la figure 13, à chaque noeud r est inscrit. Si l'on demande quelle est la formule de l'arborescence qui est la plus près de D nous avons quelques ennuis car les formules ne se ressemblent pas encore beaucoup. (la connective principale de D n'est jamais connective principale dans les formules de gauche) . Si on fait confiance au coefficient r pour mesurer l'avancement du travail, il ne reste que le choix entre 5 et 8 qui sont les formules qui semblent conduire le plus rapidement à la solution(de 5 c'est immédiat).

5 + / + A B C / C D

R2

/ + + A B C D

Remarque :

Il est certain que pour cet exemple un seuil d'arrêt un peu plus grand aurait permis de résoudre directement le problème mais n'oublions pas que le but est toujours d'essayer de dégager des méthodes valables pour un espace de travail disponible limité. (si grand soit-il !)

...

D'autres critères semblent possibles. Par exemple, dans un système où il y a beaucoup de règles simplifiantes, on pourrait considérer que les calculs sur une formule sont d'autant plus avancés que le nombre de simplifications immédiatement réalisables est réduit.

Il serait intéressant de donner au programme la possibilité de trouver lui-même les critères à utiliser dans le contexte du problème qu'on lui pose. Il y a peut-être là une voie intéressante de recherche.

III - Amélioration du problème posé

Dans le programme permettant la vérification d'égalités par récurrence nous sommes souvent amenés à poser des problèmes ou sous problèmes au Prouveur. Il y a deux problèmes principaux $\begin{cases} \text{vérifier } G_{n_o} = D_{n_o} \\ \text{vérifier } G_{n+1} = D_{n+1} \text{ si } G_n = D_n \text{ est vraie} \end{cases}$

Les sous problèmes sont ceux qui sont posés à chaque redémarrage (au sens où nous l'avons vu au paragraphe précédent).

Les problèmes qui sont ainsi présentés au prouveur ne sont pas forcément toujours posés de la façon la plus simple(ou la mieux adaptée au traitement). Ceci est surtout valable pour les sous problèmes MING $\stackrel{?}{=}$ MIND posés aux redémarrages .

Ainsi, après développement de G et D si le programme estime que les deux formules les plus proches sont :

MING : $\dfrac{(n+1)^2\ (n+2)^2}{\sin^2 x + \cos^2 y}$

MIND : $\dfrac{(n^2+3n+2)^2}{\sin^2 x + \cos^2 y}$

il est souhaitable qu'il pense à faire disparaître le dénominateur commun pour s'apercevoir qu'il lui suffit de montrer l'égalité des numérateurs : Pour cela, le programme dispose d'un certain nombre de productions simplificatric

...

Par exemple :

IMP = + U V + U W = V W [U+V $\stackrel{?}{=}$ U+W ⇒ V $\stackrel{?}{=}$ W]

IMP = / U V / U W = V W [$\frac{U}{V} \stackrel{?}{=} \frac{U}{W}$ ⇒ V $\stackrel{?}{=}$ W]

IMP = / U V / W V = U W [$\frac{U}{V} \stackrel{?}{=} \frac{W}{V}$ ⇒ U $\stackrel{?}{=}$ W]

IMP = LOG U LOG V = U V [LOG U $\stackrel{?}{=}$ LOG V ⇒ U $\stackrel{?}{=}$ V]

etc...

Ces productions ne s'appliquent que sur le problème dans son entier. Elles sont essayées systématiquement dès qu'un problème est posé. Leur utilisation est très pratique avec l'algorithme d'unification.

Exemple : Si le problème posé est

= / * + N 1 2 SIN * X 2 / + * N 2 2 SIN * X 2

la 3e production s'applique IMP { = / U V / W V
SIN * X 2
= U W
* + N 12 + * N 22

et l'égalité qui reste à vérifier est = * + N 1 2 + * N 2 2

Ce type de simplification est évidemment très intéressant.

1) Il limite la longueur des expressions que l'on manipule

2) On n'est plus tenté de travailler sur les termes qui ont été simplifiés. D'où gain de temps. (On voit mal à quoi servirait de développer le sin * X 2 de l'exemple précédent).

3) Le fait de simplifier des termes peut faire disparaître des connectives qui nécessitent pour les manipuler un grand nombre de règles. Si ces connectives disparaissent, il n'est plus nécessaire de prendre en compte les règles pour les manipuler. D'où un gain de temps appréciable puiqu'on ne perd plus de temps à essayer des règles qui ne s'appliqueront jamais. (Ce point sera repris en détail dans le paragraphe suivant consacré à la sélection des règles). Ainsi dans l'exemple précédent a priori on pouvait avoir besoin de toute la trigonométrie et des manipulations de fraction. Après application de la production seules les règles algébriques élémentaires doivent suffire pour résoudre le problème.

...

C'est pour essayer de favoriser au maximum ce type de simplifications que dans la fonction d'évaluation de distance entre deux formules, nous avons donné une bonification pour rapprocher deux formules ayant les connectives principales identiques. (voir p. 49).

IV - Le sélecteur de règles

Une heuristique grossière mais qui s'est avérée très efficace dans le programme réalisé consiste à ne pas prendre en compte les règles que l'on sait a priori être inutiles pour résoudre le problème.

Le programme écrit est assez général pour traiter de problèmes relevant de trigonométrie, de dérivées, de logarithmes,exponentielles et de tout le calcul algébrique habituel. Pour résoudre un problème particulier, il est certain que si nulle part n'apparaît le symbole de dérivation c'est perdre son temps que de balayer chaque noeud avec les règles de dérivation. De même pour la trigonométrie etc...

Rappelons qu'avec l'utilisation des productions simplificatrices, une amélioration possible du problème au moment d'un redémarrage peut amener à revoir les règles à utiliser (en général à en supprimer). Le sélecteur de règles doit donc être appelé après les essais d'amélioration du problème.

Le sélecteur que nous avons utilisé suppose que les règles sont classées par niveaux. Nous avons travaillé avec neuf niveaux de règles (en moyenne 15 règles par niveaux).

2 - manipulation algébrique
3 - Puissances - racines
4 - LOG - EXPO.
5 - Trigonométrie
6 - Σ π
7 - manipulation des fractions
8 - dérivation
9 - règles privilégiées (suites numériques par exemple)
10 - Hypothèse de récurrence

Le niveau 1 est réservé pour les productions simplificatrices

...

Pour résoudre un problème donné, on commence par extraire les différentes connectives qui figurent dans l'égalité à démontrer et on ne prend en compte que les niveaux de règles nécessaires pour manipuler ces différentes connectives. Le dernier niveau (hypothèse de récurrence) a un traitement particulier : un indicateur spécial indique au moment de la sélection s'il doit être pris ou pas.

Exemples avec ce qui précède :

. Pour montrer que $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Il suffit des niveaux 2, 6, 7, (10)

. Pour montrer que $\frac{d^n}{dx^n} xe^x = (x+n)e^x$

Il suffit de 2, 4, 8, (10)

Dans le programme, on utilise un tableau de choix quidonne le niveau à prendre en présence d'une connective donnée. Ici nous aurions :

+	-	*	---	/	PUI	RAC	SIG	PRD	SIN	COS	TG	COT	LOG	EXP	DER
2	2	2	2	7	3	3	6	6	5	5	5	5	4	4	8

Les niveaux attribués à chaque groupe de règles on été choisi de façon à ce que les règles jugées les plus efficaces se trouvent à être employées les premières. Pour cela, nous faisons un tri par ordre décroissant sur les niveaux, une fois que ceux-ci ont été déterminés. Ainsi, la règle de récurrence sera toujours la première essayée · Dans un problème de dérivation, on cherchera toujours à dériver avant de faire de la manipulation algébrique etc...

V - Introduction de règles conditionnelles

Dans la pratique, les règles que nous avons à utiliser ne sont pas toujours aussi simplesque celles que nous avons vues jusqu'à présent. Très souvent une règle ne s'applique que si certaines conditions sont remplies. Conditions pouvant porter en particulier sur les éléments manipulés par la règle.

...

Exemples :

$-\sqrt{u^2} = u$ si $u > 0$

$\sqrt{u^2} = -u$ si $u < 0$

$-\frac{du}{dx} = 0$ si x n'apparaît pas dans u .

Si nous n'assortissons pas la règle de sa condition, nous pouvons arriver à des résultats faux uniquement parce que l'unification se fait bien mais qu'elle n'est pas "légale".

Exemples :

1- T : ~~DER IND 1 X SIN X~~ [$\frac{d}{dx}$ sin x]

A : ~~DER IND 1 X U~~ [$\frac{d}{dx}$ U]

B : O [O]

Finalement on trouve 0 comme dérivée de sinx ce qui est bien sûr faux. Remarquons qu'avec la règle $\frac{du}{dx} = 0$ dans le système, on trouverait que toute fonction a une dérivée nulle.

2 - T : ~~RAC PUI~~ --- 3 ~~2~~ [$\sqrt{(-3)^2}$]

A : ~~RAC PUI U 2~~ [$\sqrt{U^2}$]

B : ~~U~~ --- 3 [U]

On trouve que $\sqrt{(-3)^2} = -3$ résultat faux évidemment . Le problème est donc de faire intervenir une condition capable de faire refuser un résultat fourni par une unification qui a réussi. Cela revient à introduire dans l'unification une cause d'échec supplémentaire (liée cette fois ci au problème traité et non due au mécanisme revu au début de ce travail). Pour résoudre ce problème, la méthode suivante donne de bons résultats.

Nous adoptons des règles ayant la structure suivante en polonaise :

: = U SI Cond ALORS V

(on peut lire : Si (cond) est vraie alors U se réécrit V)

...

Dans cette notation :

: = est une connective binaire (symbole de réécriture)

U est le membre gauche.

SI est une connective binaire qui relie la condition au second membre.

Cond est la condition

ALORS est considéré comme connective unaire. Il n'est introduit que pour rappeler le "bon schéma" SI ... ALORS ....

V est le membre droit de la réécriture.

Fonctionnement d'une telle règle

Pour utiliser une telle règle, on utilise l'unification classique. En cas de succès de l'unification, on examine la condition : si elle a la valeur vraie, alors V modifié est pris comme résultat. Si elle est fausse, on considère qu'il y a eu échec et on ne retient pas le résultat.

Exemple :

règle : = RAC PUI U 2 SI < U 0 ALORS --- U

expression RAC PUI --- 3 2

l'unification se fait avec le schéma.

T : ~~RAC PUI~~ --- 3 ~~2~~

A : ~~RAC PUI~~ ~~U~~ ~~2~~

B : SI < ~~U~~ 0 ALORS --- ~~U~~

---3 --- 3

L'unification réussit bien.

La présence du SI comme première connective dans le résultat (B) indique qu'il y a une condition (ici : < --- 3 0). L'examen de la condition (ici : - 3 inférieur à 0) répond que la condition est satisfaite. Le résultat est donc le terme qui suit ALORS (à savoir --- --- 3) .

Si avec la même règle nous avions unifié avec RAC PUI 5 2 nous aurions obtenu dans B

SI < 5 0 ALORS --- 5

La condition (5 < 0) n'étant pas satisfaite, le résultat --- 5 aurait été rejeté.

...

Remarques :

1 - L'unification se comporte très bien dans ce cas (même si la condition porte sur des éléments internes à la règle où à l'expression).

2 - L'unification doit être faite avant l'examen de la condition.

3 - L'intérêt de mettre le SI comme première connective du membre gauche est que l'on s'aperçoit très vite s'il y a lieu de valider ou pas l'unification qui vient de réussir. (toutes les règles du système ne sont pas conditionnelles). La présentation adoptée est en cela préférable à celle théoriquement équivalente

: = U V SI Cond

Evaluation de la condition

Un problème important reste ; c'est l'évaluation de la condition. Cette condition peut être représentée par n'importe quelle expression logique. Une expression logique peut être formée de propositions de base mais aussi de disjonction, conjonction ou négation de propositions de base.

Il faut entendre par proposition de base une proposition qui nécessite un calcul approprié pour connaître sa valeur de vérité.

Exemples Proposition de base

1 - VU X U (valeur "vrai " si variable X est présente dans terme U)

2 - CST U (valeur "vrai " si U est une constante)

3 - VAR U (valeur "vrai " si U est une variable)

4 - < X Y (valeur "vrai " si X inférieur à Y)

expressions logiques

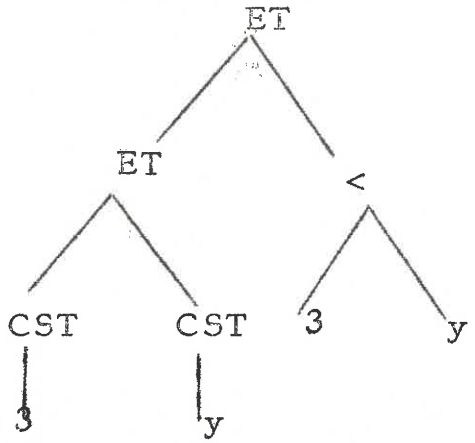
5 - NON VU X U (vrai si X n'apparait pas dans U)

6 - ET ET CST 3 CST Y < 3 Y
(vrai si 3 et y sont des constantes et si 3 est inférieur à Y).

D'une façon générale, cette expressionlogique peut être mise sous forme d'une arborescence.

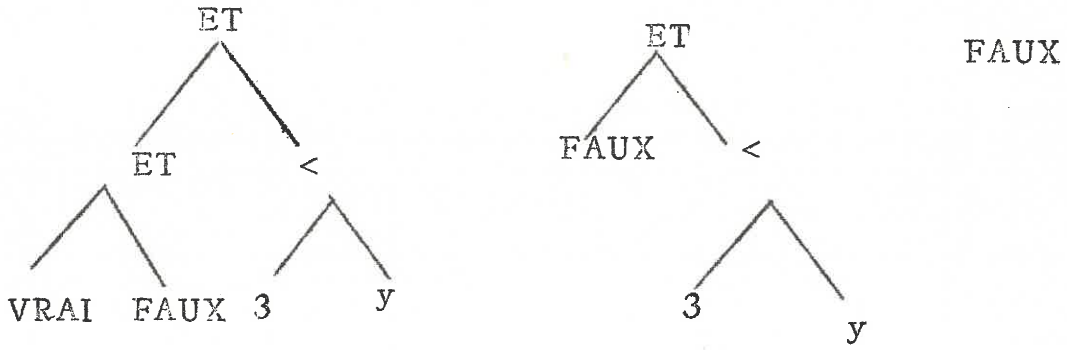
...

L'exemple 6 précédent donne



Pour calculer la valeur de vérité de cette expression, il suffit de calculer la valeur de chaque proposition de base. (chacune nécessite un traitement approprié, mais elles sont en nombre faible en général). Ensuite il suffit de faire un calcul logique.

L'arborescence précédente donne successivement :



Dans le cas où l'expression ne peut pas être évaluée pour une raison ou pour une autre, en lui attribuant la valeur FAUX d'office, nous évitons l'introduction d'expressions erronées. Il se peut que nous écartions une formule vraie. Cela ne nous est jamais arrivé.

Remarque

Le calcul de l'expression logique (e.l.) pourrait se faire en rappelant le prouveur dans lequel nous sommes.

En effet, trouver que e.l. vaut vrai (resp. faux) peut se ramener à démontrer que e. l. = VRAI (resp. FAUX) Pour cela, nous pourrions utiliser une fois les propositions de base évaluées, des règles comme :

...

= ET FAUX U FAUX

= OU VRAI U VRAI

= NON VRAI FAUX

= NON NON U U

etc...

Le processus serait alors le suivant : Pour évaluer e.l. a priori, se poser le problème el $\stackrel{?}{=}$ VRAI

S'il y a succès, attribuer VRAI à el .
sinon se poser el $\stackrel{?}{=}$ FAUX

S'il y a succès attribuer FAUX à e.l.
Sinon attribuer FAUX (avec la convention faite précédemment.

C'est à dire qu'on attribue la valeur FAUX sauf si on peut démontrer que e. l.= VRAI. Cette méthode serait sans doute rapide. Nous ne l'avons pas utilisée à cause de l'aspect récursif qu'elle introduit dans le programme. (Nous aurions en effet à rappeler le prouveur de l'intérieur de lui-même).

Exemple complet

Règle = DER IND 1 X U SI NON VU X U ALORS 0

Expression à dériver DER IND 1 X * Y Z

(IND indices de dérivation : ici dérivée première par rapport à X).

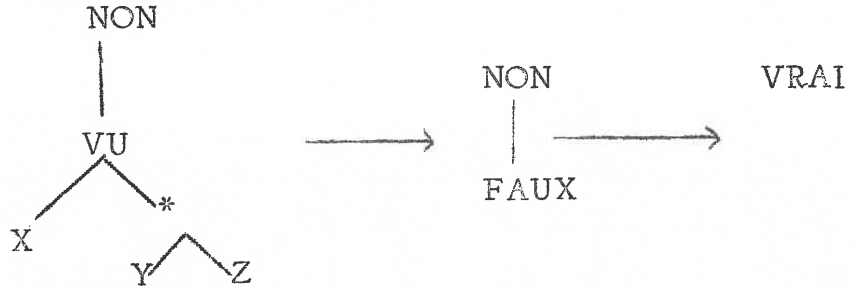
1 - L'unification donne

~~DER IND 1 X~~ * Y Z

~~DER IND 1 X U~~

SI NON VU X ~~U~~ ALORS 0
* Y Z

2 - Analyse de la condition

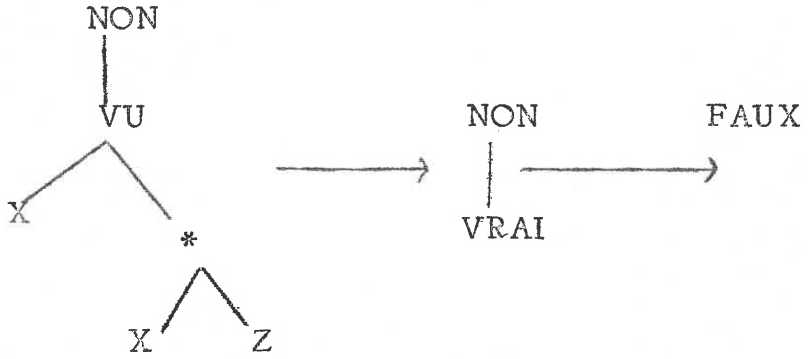


3 - Résultat 0

...

Remarque :

Si on remplace Y par X dans l'exemple, la condition devient



Le résultat est alors écarté. Tout ce passe comme si l'unification avait échoué.

Intérêt de ces règles en Intelligence artificielle

L'essentiel de l'outillage en Intelligence artificielle est constituée par les heuristiques. Certains travaux de Waterman [II - 12] ont essayés de formaliser ce travail avec les heuristiques. En particulier il est intéressant de représenter les heuristiques comme des règles.

En effet, si un programme travaille avec des heuristiques formalisées ainsi, il est très facile de changer ces heuristiques, d'en ajouter, d'en supprimer etc.. Puisqu'elles ne sont pas "intégrées dans le programme mais qu'elles apparaissent comme des données. (Le but de Waterman était à partir de mécanisme d'apprentissage, de faire découvrir au programme les heuristiques dont il peut avoir besoin).

Si nous examinons ces "règles heuristiques" nous nous apercevons qu'elles sont presque toujours conditionnelles au sens où nous l'avons vu précédemment.

Exemples

1 - Si nous voulons manipuler le binôme de Newton

$$(a + b)^n = a^n + C_n^1 \, a^{n-1} \, b + \ldots + b^n$$

il est facile de dire que les coefficients sont donnés par le triangle de Pascal mais est-il raisonnable de développer $(a + b)^{1000}$? Ce développement ne se fera que sous condition : On peut alors voir que la règle conditionnelle $(a + b)^n = a^n + C_n^1 \, a^{n-1} \, b + , \ldots + b^n$ si $n < 5$ représente l'heuristique

...

"ne faire ce développement que si n n'est pas trop grand, autrement chercher autre chose ".

Nous sommes là dans le cas d'une règle qui n'est pas expansive au sens où elle n'amène pas un espace de recherche infini. La règle brute ne serait dangereuse que parce qu'elle peut faire déborder tout l'espace de travail avec une seule expression (qui n'amènerait probablement rien de très intéressant pour le problème).

2 - Ce deuxième exemple va montrer comment on peut manipuler des règles expansives avec ce type de condition et limiter l'expansion de l'arborescence. C'est un cas où la règle contient dans sa condition une heuristique permettant une manipulation correcte.

La règle s'écrit :

= PUI U V SI VU PUI U - V 1 TOU ALORS * PUI U -V1 U

(Elle signifie que U^V peut se réécrire U^{V-1} . U si le terme U^{V-1} apparaît déjà dans l'expression que l'on manipule (le symbole TOU indique qu'il faut examiner toute l'expression et ne pas se limiter à ce qui est dans le champ d'application de la règle)).

Exemple :

Soit à montrer

= * PUI + N 1 2 PUI + N 2 2 + * PUI + N 1 2 PUI N 2 * PUI + N 1 3 4

$[(n+1)^2 (n+2)^2 \stackrel{?}{=} (n+1)^2 . n^2 + (n+1)^3 . 4]$

La règle précédente ne s'applique qu'en 24 mais cela permet d'améliorer beaucoup le problème.

En effet, nous avons à unifier :

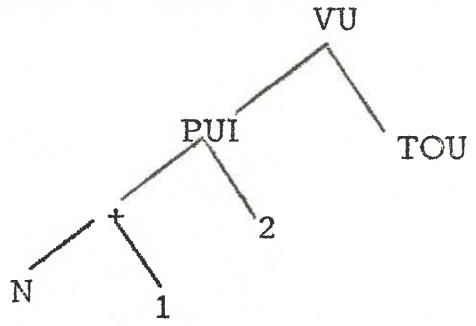
~~PUI + N 1 3~~

~~PUI U V~~

SI VU PUI ~~U~~ - ~~V~~ 1 TOU ALORS * PUI ~~U~~ - V 1 ~~U~~
+N1 3 + N 1 3 + N 1

...

La condition à examiner est



La procédure de recherche de terme trouve effectivement ce terme en 3 et 15. La condition étant vraie, le terme commençant en 24 est remplacé par * PUI + N 1 2 + N 1 . Le terme * PUI + N 1 2 se met ensuite bien en facteur dans le second membre.

Nous avons là une façon de manipuler les règles expansives qui diffère sensiblement de celle utilisée par L. Siklossy . Ici, la stratégie de limitation de l'expansion est implicite dans la condition de la règle. Elle est facilement modifiable. Chez Siklossy, la stratégie est interne au programme et indépendante des problèmes. (On utilise uniquement des termes atomiques figurant dans l'expression à la première passe etc...).

3ème PARTIE

PROBLEMES LIES A LA MANIPULATION DE CHAINES DE SYMBOLES

- Simplification

- Normalisation

Introduction

Dans un programme tel que celui que nous avons écrit, nous manipulons essentiellement des chaînes de caractères. Les objets en présence sont essentiellement repérés par leur adresse dans une table indépendamment de la valeur qu'ils représentent. Seule la nature (constante, variable, connective) de l'objet peut intervenir.

Une certain nombre de problèmes se posent dans la manipulation des expressions algébriques : Il s'agit d'une part du choix d'une politique de simplification, d'autre part d'une manipulation correcte de l'associativité et de la commutativité des opérations possédant ces propriétés.

I - Simplification

Nous pouvons d'abord dire que le problème des simplifications a été particulièrement bien exposé par MOSES [V - 8] et repris par Laurent [IV - 8]. Aussi, nous ne ferons que rappeler les traits essentiels de ce problème.

Pour définir ce que c'est qu'une simplification, nous pouvons dire que c'est une manipulation qui , appliquée à une expression algébrique, permet d'obtenir une nouvelle forme à partir de laquelle le calcul pourra être effectué plus efficacement.

Le problème vient du fait que la notion de simplicité pour une expression est directement liée au contexte dans lequel elle se place. Tout dépend du but que l'on poursuit, de l'utilisation que l'on veut faire de l'expression concernée.

Une expression peut paraître simple dans un contexte et compliquée dans un autre.
Par exemple $\frac{x^7}{x^{12}+1}$ peut paraître plus simple que $\frac{\frac{1}{4}(4x^3)\,x^4}{(x^4)^3+1}$

pourtant, si nous nous intéressons au calcul d'une primitive de cette fraction rationnelle, la deuxième forme est plus prometteuse de succès. En aucune façon, il n'y a intérêt à "simplifier" la forme 2 pour la ramener à la forme 1 si c'est ce calcul que nous avons en vue.

...

Par contre, un certain nombre de simplifications pourront être faites de façon systématique. Par exemple $x * 0$, $x * 1$, $y + 0$, $5 - 3$, etc pourront sans problème en général être remplacé par leur forme simplifiée 0, x, y, 2, ... Cependant même de telles simplifications doivent être manipulées avec prudence. Ainsi dans le problème de la recurrence, l'expression $x * 1 + x * n$ peut être transformée facilement en $x * (1 + n)$ qui a l'avantage de faire apparaître le terme $(n+1)$ qui peut être cherché. La simplification $x * 1 \rightarrow x$ pourrait gêner le programme (à moins de lui donner la règle $u + u * v := u * (1 + v)$. Il est en effet impensable de mettre dans notre système pour recréer le 1 la règle $u := 1 * u$, elle est expansive).

Nous voyons ainsi nettement qu'il y a un problème de choix fondamental dans de tels systèmes : Devons nous simplifier partout, nulle part de façon systématique. Il semble que des solutions intermédiaires puissent être trouvées.

Moses, a classé les différents systèmes possibles en "radicaux", "nouvelle gauche", "libéraux", "conservateurs", "catholiques",.

- Les systèmes "radicaux" travaillent uniquement sur des expressions mises sous forme canonique. La politique de simplification est donc très sévère : deux expressions équivalentes sont systématiquement ramenées à la même représentation.

- Les systèmes "nouvelle gauche" permettent de prendre quelques libertés par rapport à la représentation canonique. Il peut y avoir quelques inconvénients dans les systèmes radicaux (par exemple il est certainement absurde de développer $(x + 1)^{1000}$ sous forme normalisée de polynôme. Un système nouvelle gauche peut admettre de ne pas faire ce développement.

- Les systèmes "libéraux" se rapprochent plus du comportement humain. les simplifications se font de façon locale dans l'expression. Le même terme peut être modifié d'une façon à un endroit et d'une autre ailleurs. La façon de procéder est beaucoup plus naturelle mais l'efficacité est réduite . Ce sont des systèmes généralement plus coûteux en espace mémoire et en temps.

- Les systèmes "conservateurs" partent du principe qu'aucune simplification "évidente" comme $0 * x \rightarrow 0$, $1 * x \rightarrow x$ ne doit être faite de façon aveugle, sans examiner le contexte. Ces simplifications peuvent

...

détruire une information utile empêchant par exemple de reconnaître un "pattern" (par exemple $x^2 - 1^2$).

- Les systèmes "catholiques" utilisent plusieurs types de représentations et plusieurs approches au problème de simplification. Par exemple, certains calculs seront effectués avec une politique libérale alors que d'autres seront effectués de manière radicale sur des formes canoniques. Ces systèmes prétendent associer le naturel des systèmes libéraux à l'efficacité des systèmes radicaux.

Le programme que nous avons écrit peut être classé dans les systèmes "catholiques". Il utilise un certain nombre de techniques radicales. Tous les calculs de constantes sont effectués (ainsi 1^2 est systématiquement remplacé par 1 etc...). Lorsque l'expression G ou D de l'égalité à vérifier est un polynôme, elle est systématiquement normalisée (et on s'interdit tout redéveloppement de cette forme normalisée). Par contre, les polynômes internes à un terme ne sont pas normalisés. Pour le reste, les simplifications se font en suivant une politique "libérale" (par exemple , les 1 * x, 0 + U etc... peuvent être simplifiés ou gardés. En fait, dans ce cas là, les deux formules (l'une sans la simplification, l'autre avec la simplification faite) sont conservées dans l'arborescence.

II - Normalisation

1 - Nous avons utilisé des formes canoniques pour représenter les polynômes. En fait, la forme normale n'a été imposée que lorsque tout un membre de l'égalité à vérifier est un polynôme. Les polynômes internes à un terme n'ont pas été ramenés à leur forme canonique.

Par exemple : Pour vérifier que $\sum_{i=1}^{n} 2\,i = n\,(n+1)$, le programme pour le passage $n \rightarrow n + 1$ doit vérifier que

$$\sum_{i=1}^{n+1} 2i = (n+1)\,((n+1)+1)$$

Il commence par normaliser le polynôme du second membre et se pose le problème

$$\sum_{i=1}^{n+1} 2\,i \stackrel{?}{=} n^2 + 3\,n + 2$$

...

Il est inutile de normaliser le polynôme 2 i

Forme canonique choisie

En polonaise préfixée, pour le polynôme

$$a_o\ x^n + a_1\ x^{n-1} + a_2\ x^{n-2} + \ldots a_{n-1}x + a_n$$

nous avons choisi la forme générale

$$\underbrace{++\ \ldots\ +}_{\text{n signes +}} * \text{PUI X N } a_o * \text{PUI X - N 1 } a_1 \ldots * \text{PUI X 1 } a_{n-1} * \text{PUI XO } a_n$$

C'est un terme (ausens habituel) et cela permet de rester cohérent avec les représentations en listes.

Il pourrait être intéressant de représenter un polynôme par la donnée de son degré, son indéterminée, et le vecteur de ses coefficients. Cela oblige à introduire dans le programme des objets de nature nouvelle (il y aurait les polynômes comme il y a les constantes, les variables, les connectives). En fait, nous avons utilisé ces formes canoniques uniquement pour faire des comparaisons. Nous n'avons pas eu besoin des algorithmes habituels de manipulation (somme,produit etc...) de polynômes. Aussi, la représentation choisie a été suffisante.

2 - Normalisation des sommes et produits

Dans un programme tel que celui que nous avons écrit un problème important est soulevé par les opérations associatives et commutatives. Il n'est en effet pas possible de prendre en compte toutes les formules qui se déduisent l'une de l'autre par application de l'une de ces règles.

A titre de curiosité examinons combien de formules équivalentes nous pouvons produire à partir d'une formule contenant n fois une connective ayant ces propriétés (cette connective étant la seule dans l'expression).

- En ne tenant compte que de la commutativité

Soit $C_{(n)}$ le nombre cherché. Notons . la connective

pour n = 1 nous obtenons 2 formules [a.b et b. a]

pour n = 2 nous obtenons 4 formules [(a.b).c, (b.a).c, c.(a.b), c.(b.a)]

par récurrence si $C_{(n)} = 2^n$ avec (n+1) connectives nous pouvons

fabriquer (formulesoù figure n fois la connective), x et x. (formules où figure n fois la connective)

...

soit $C_{n+1} = 2 \cdot C_{(n)} = 2^{n+1}$

finalement $\forall n_{\in N}\ C_{(n)} = 2^n$

- En ne tenant compte que de l'associativité

Soit A (n) le nombre cherché. Notons . la connective

pour n = 2 nous obtenons [(a.b).c , a.(b.c)] soit A (2) = 2

pour n = 3 (abc).d (a.b).(c.d) a.(b c.d)

2formules 1 formule 2formules soit A(3) = 5

pour n = 4 (abcd). e (abc).(de) (ab).(cde) (a.(bcde)

5 formules 2 formules 2 formules 5 formules

soit A(4) = 14

On peut voir facilement qu'en posant par convention A(0) =1 , A(1)=1, nous avons $A_{(n)} = A_{(n-1)} \cdot A_{(o)} + A_{(n-2)} \cdot A_{(1)} + \cdots + A_{(o)} \cdot A_{(n-1)}$

c'est à dire que le nombre $A_{(n)}$ de formules que l'on peut former avec (n+1) variables et n fois une connective associatives est donné par

$$A(o) = 1$$

$$A_{(n)} = \sum_{i=1}^{n} A(n-i) \cdot A(i-1)$$

- En tenant compte de l'associativité et de la commutativité

Soit C $A_{(n)}$ le nombre cherché

Pour chaque combinaison de parenthèses , nous pouvons faire toutes les combinaisons de commutativité (qui correspondent à toutes les façons dont on peut changer l'ordre des lettres sans toucher aux parenthèses et points).

$$C\ A(n) = C_{(n)} \cdot A(n)$$

Soit $C\ A(n) = 2^n \cdot \left(\sum_{i=1}^{n} A(n-i) \cdot A(i-1)\right)$

Une tabulation pour les premières valeurs de n montre immédiatement l'ampleur du problème :

...

n	$C_{(n)}$	$A_{(n)}$	CA (n)
1	2	1	2
2	4	2	8
3	8	5	40
4	16	14	224
5	32	42	1344
6	64	132	8448
7	128	429	54912

- Si les connectives sont de nature différente, le problème est plus difficile. A titre d'exemple considérons le polynôme de degré 3 sous forme normalisée :

Soit P = +++ * PUI X 3 A * PUI X 2 B * PUI X 1 C * PUI X 0 D

Si nous tenons compte de la commutativité de * , nous pouvons fabriquer 2^4 formules; Avec la commutativité de + nous pouvons introduire 2^3 formules. Ceci amène $2^4 \times 2^3 = 2^7 = 128$ formules uniquement à cause des commutativités.

L'associativité de * n'introduit pas de formules supplémentaires, les produits étant isolés un à un dans chaque terme de la somme par contre l'associativité de + permet à partir de chacune des formules précédentes d'en fabriquer 5 équivalentes : Ce qui fait qu'en tenant compte de l'associativité et de la commutativité, on peut fabriquer 5 * 128 (= 640) formules équivalentes mais d'écriture différente à partir de cette seule formule.

Pour remédier à ces problèmes de croissance des arborescences , nous avons adopté une normalisation partielle des produits et des sommes. Nous avons utilisé pour cela un système de hash-code (nous noterons . pour une opération + ou * indistinctement).

Soit le terme . t_1 t_2 . La technique consiste à définir un code pour chacun des termes t_1 et t_2 et à ne conserver que la formule pour laquelle le premier terme est celui qui à le code le plus élevé. Ainsi . t_1 t_2 est systématiquement ramené à . t_2 t_1 si code (t_2) > code (t_1).

Définition du code d'un terme

Les symboles sont représentés en machine sous forme numérique :

...

Les connectives par (100 * ORD + AD) où ORD est l'ordre de la connective (1 pour les connectives unaires, 2 pour les connectives binaires) et AD l'adresse dans la table des connectives. Les constantes (resp. variables) par leur adresse dans la table des constantes (resp. variables).

Les variables et constantes ont toujours un code inférieur à 100. Pour les connectives, il est toujours supérieur à 100.

Nous avons adopté les conventions suivantes : pour un symbole, le code est la valeur numérique représentant le symbole dans la chaîne. Pour un terme, le code est la somme des codes de chacun de ses symboles.

Nous avons ainsi une façon très simple et très rapide pour trouver le code d'un terme. Dans une représentation en liste de l'expression, il y a au plus à changer les pointeurs de . vers t_1 et t_2 pour obtenir la forme normale. Il est donc très aisé de ramener ces expressions à leur forme normale. Cette technique résoud pratiquement le cas de la commutativité. Le seul cas qui reste en suspens est celui où le code (t_1) = code de (t_2). Cela peut se produire par des compensations dans les sommes d'adresses.

- par exemple .. x y . z t
- Si x, y, z, t ont pour adresses respectives dans la table des variables 4, 1, 3, 2 il pourra arriver que l'on conserve les formes équivalentes .. x y . z t et . . z t . x y puisque code (. x y) = code (. z t).

Dans la pratique cela arrive très rarement et ne nous a jamais gêné. Il faut bien se rendre compte que cette normalisation intervient à chaque instant et qu'il faut que le dispositif reste le plus simple et le plus rapide possible.

En ce qui concerne l'associativité, le problème est moins simple. Il faut en effet pouvoir garder cette possibilité d'associativité pour isoler des termes, les réarranger . C'est elle qui prépare le travail à la distributivité qui permet de mettre les bons termes en facteur etc...

Une solution consisterait à utiliser des connectives "variaires" (c'est à dire dont l'ordre varie d'une utilisation à l'autre.) Par exemple + ne serait pas toujours binaire, mais serait n - aire si elle porte sur n termes.

...

En notant $+_n$ cette connective n-aire, nous pourrions représenter

$$a + b + c \quad \text{par} \quad +_3 \; a \; b \; c$$

sans se poser les problèmes de parenthèses qui apparaissent si on considère + comme binaire . (pour l'exemple précédent, cela donne en polonaise préfixe les deux formes ++ ab c et + a + bc) . Avec un système de Hash-code identique à ce qui a été fait pour la commutativité, il serait possible de classer de façon standard les termes a, b, c. Nous pourrions ainsi ne conserver qu'un représentant de cette classe de formules équivalentes à l'associativité près.

On peut remarquer que ce concept de connective variaire nous rapprocherait du comportement humain. En effet, pour a + b + c on dit couramment que c'est la somme de a, de b et de c et non pas que c'est la somme de a et de la somme de b et de c ou autre formule analogue.

Nous n'avons pas utilisé ce dispositif. Il aurait en effet fallu le prévoir dès le début de la programmation de l'outillage de base. Ces connectives nécéssitent des représentations adaptées et une reprogrammation complète serait nécessaire pour les introduire. (il faut en particulier introduire avec chaque connective, une information supplémentaire : son ordre alors qu'actuellement cet ordre est implicite, connu et fixe pour chaque connective. Pour les représentations en liste des expressions, il faudrait admettre des arborescences telles que chaque noeud puisse avoir n fils etc...)

Dans le programme que nous avons écrit, c'est en fait le dispositif de hash-code mis en place pour la commutativité qui règle le problème de l'associativité (cela tient au fait que + et * sont à la fois commutatives et associatives). En effet, si nous mettons toujours en premier le terme de plus fort code (en jouant sur la commutativité) nous ramenons en tête les connectives (qui ont toutes un code plus fort que les variables).

Par exemple si a, b, c sont des variables

- (a + b) + c comme a + (b+c) seront représentés par

++ a b c car code (+ ab) > code (c)

- parcontre (a + b) +((c+a)+b) sera représenté par

+++ a c b + a b

et l'application d'une fois l'associativité ramènera à + +++ a c b a b. Avec ce système nous ne gardons pas qu'un seul exemplaire des formules équiva-

...

lentes à l'associativité près, mais nous limitons beaucoup le nombre de formules pouvant être produit.

Remarque : L'avantage de ramener les connectives en début de formule est lié à l'unification. Une des causes d'échec lorsqu'on unifie une formule avec le premier membre d'une règle est de trouver deux connectives différentes à la même place dans les deux expressions. Si une unification doit échouer parce que la règle ne peut pas s'appliquer, il y a intérêt à ce que cet échec se produise le plus tôt possible (de façon à ne pas perdre de temps à substituer des termes à des variables pour finalement échouer plus loin.

Cette façon de traiter l'associativité et la commutativité impose un certain nombre de contraintes sur les règles que nous devons donner au programme. En particulier nous sommes obligés de prévoir double règle lorsqu'il y a une opération commutative.

Exemple : si nous voulons exprimer $\frac{U}{V} \cdot W = \frac{U \cdot W}{V}$

nous sommes obligésde donner pour cette règle les deux formes

$$*\ /\ U\ V\ W = /\ *\ U\ W\ V$$

et

$$*\ W\ /\ U\ V = /\ *\ U\ W\ V$$

En effet, dans les formules où nous pouvons appliquer cette règle certaines (celles où code (/U V) > code (W)) utiliseront la forme 1, les autres utiliseront la forme 2.

ainsi $\frac{x}{y} \cdot z$ [représenté par */xyz] utilisera la forme 1 pour donner $\frac{x \cdot z}{y}$

par contre $\frac{x}{y} \cdot (a+b+c)$ [représenté par * ++ a b c / x y] utilisera la forme 2 pour donner $\frac{x \cdot (a+b+c)}{y}$.

Cela amène une augmentation sensible du nombre de règles à introduire (et à appliquer à chaque noeud). C'est un facteur de ralentissement du programme.

4ème PARTIE

CARACTERISTIQUES DU PROGRAMME

- Conditions matérielles
- Performances
- Résultats

AMELIORATIONS SOUHAITABLES

Les méthodes et techniques décrites dans les parties qui précèdent ont été mises en oeuvre dans un programme. Cette partie a pour but d'exposer les caractéristiques principales de ce programme ainsi que ses résultats et performances. Enfin, quelques améliorations possibles sont envisagées.

I - CARACTERISTIQUES DU PROGRAMME

1 - Conditions matérielles de travail

Pour réaliser ce travail, nous avons pu avoir à notre disposition l'ordinateur digital PDP 15/20 de la Faculté des sciences du Mans et le CDC 3600 de l'Institut de Programmation de Paris. Le PDP 15/20 ne disposant que de 16 K mémoires de 18 bits, il a juste été possible de tester avec lui séparément toutes les procédures de service. L'assemblage du programme s'est fait sur le CDC 3600. Enfin, en septembre 1973, le programme a été reconverti pour travailler sur le terminal du centre de calcul du Mans. Ce terminal est connecté à l'IBM 370-165 du CIRCE à Orsay. Les résultats et performances donnés ici sont ceux obtenus sur l'IBM 370-165.

Le programme a été écrit en FORTRAN. Il comporte 1450 instructions Le FORTRAN utilisé est assez pauvre (c'est approximativement celui du DIGITAL PDP 15) et n'utilise pas les possibilités intéressantes offertes par les compilateurs disponibles au CIRCE. Le programma est compilé et exécuté avec le compilateur WATFIV. Ce compilateur permet de traiter facilement les chaines de caractères. Il est très rapide à la compilation mais assez lent en exécution. Il a l'avantage de donner d'excellents diagnostics d'erreurs. Après compilation, le code généré occupe 60 000 octets alors que les tableaux occupent 56 000 octets.

Représentations choisies

Pour représenter les expressions arithmétiques, nous avons utilisé soit la notation polonaise préfixée soit la représentation en listes. Tout ce qui est unification se fait sous la forme préfixée. Les comparaisons de termes, recherches de termes, substitutions etc... se font sur les listes. A l'usage, ce choix de deux types de représentations s'avère peu judicieux. Il semble en effet que le programme perde un temps non négligeable à faire des changements de représentation.

...

Performances

On peut définir plusieurs critères pour mesurer les performances d'un programme de ce type. On peut donner le temps nécessaire pour résoudre un exercice. Nous ne disposons pas des temps de calculs sur chaque exercice (WATFIV ne permettant pas l'appel des fonctions horloges). Nous pouvons définir un autre critère de mesure de performance : c'est l'évaluation du temps moyen pour un pas de démonstration. Un troisième critère ne faisant pas appel à la notion de temps peut être intéressant : c'est le rapport

$$r = \frac{\text{Nbre de pas de démonstration}}{\text{Nbre de noeuds générés}}$$

A titre d'indication, ces critères ont été évalués sur quelques exemples (les temps sont évalués à partir du temps total de JOB. Il comprend entre autre la lecture des 80 règles)

- $\sum_{i=1}^{n} 2i = n(n+1)$ ($\sim$ 20 s) — rang 1 – 1pas ; rang n – 6 pas — 60 noeuds

- $\sum_{i=1}^{n} (i^2 - i) = \frac{n(n^2-1)}{3}$ ($\sim$ 25 s) — rang 1 – 1 pas ; rang n – 8 pas — 70 noeuds

- $\sum_{i=1}^{n} (2i)^2 = n(n+1)(2n+1) * \frac{2}{3}$ ($\sim$ 40 s) — rang 1 – 1 pas ; rang n – 6 pas — 81 noeuds

- $\sum_{i=1}^{n} i(i+1)(i+2) = \frac{n(n+1)(n+2)(n+3)}{4}$ ($\sim$ 2mn)

  rang 1 – 1 pas — 86 noeuds avant 1er redémarrage
  rang n – 9 pas — 87 noeuds 2ème redémarrage puis simplification

- $\sum_{j=1}^{n} (a_j + b_j) = \sum_{k=1}^{n} a_k + \sum_{l=1}^{n} b_l$ ($\sim$ 13 s) — rang 1 – 3 pas ; rang n – 7 pas — 23 noeuds

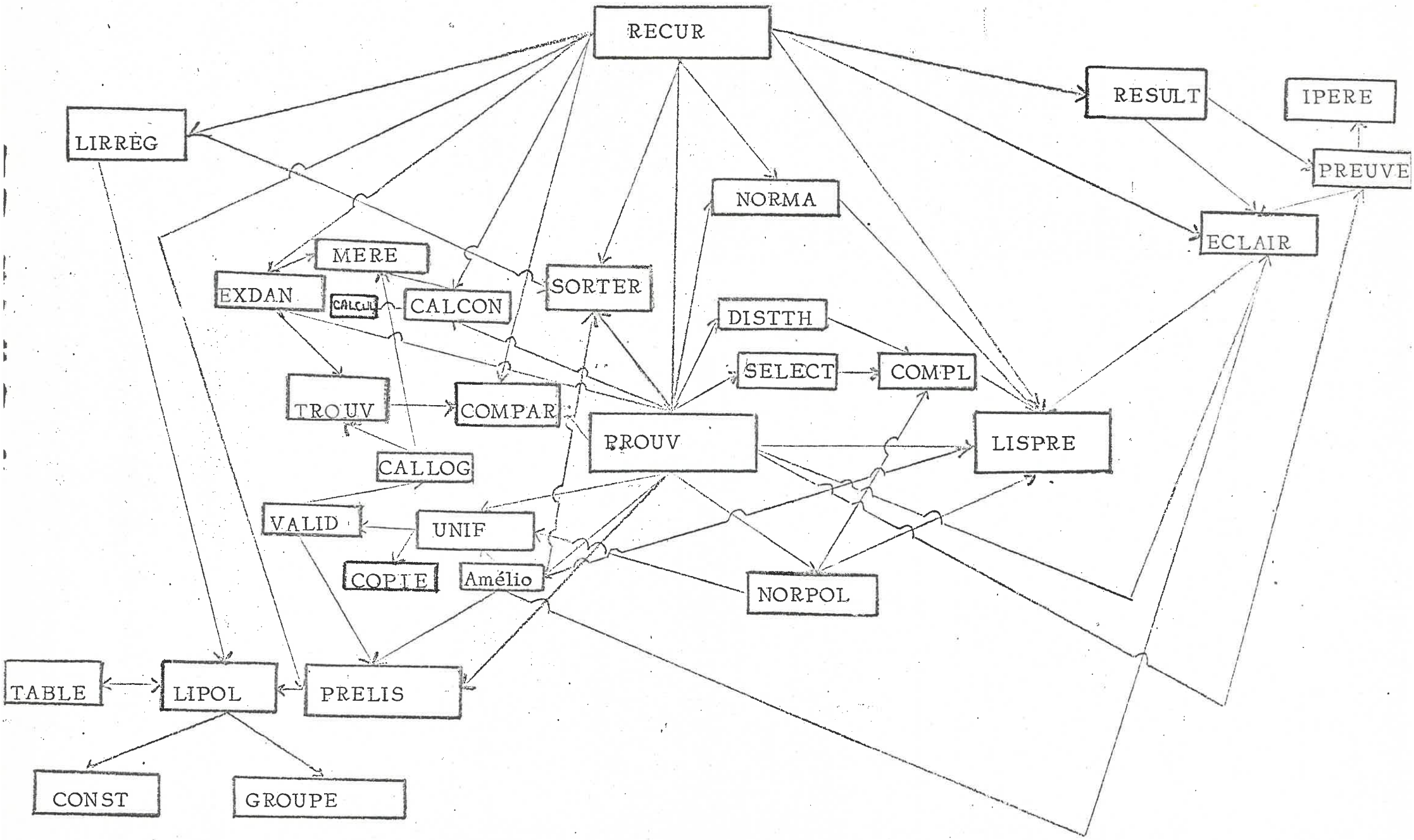
- $\frac{d^n}{dx^n}(xe^x) = (x+n)e^x$ ($\sim$ 40 s) — rang 1 – 6 pas ; rang n – 13 pas — 70 1er démarrage ; 22 2ème "

...

Ces performances du point de vue temps restent moyennes. Cela s'explique par une trop grande économie au niveau de la programmation initiale. L'exiguité de la mémoire du PDP 15 et le coût très bas du temps de calcul sur cet appareil nous ont fait prévoir des sous programmes très nombreux s'éxécutant dans un minimum de place. Ainsi, pour les arborescences (aussi bien de développement du problème que dans les expressions en listes) nous n'avons jamais gardé de pointeur pour remonter, mais nous avons définit des fonctions père. Ceci allonge évidemment les temps d'exécution. Sur le plan de la méthode, les performances sont tout à fait acceptable. On peut remarquer d'après ce qui précède que le temps de calcul sont beaucoup plus liés au volume d'arborescence developpé qu'à la profondeur des démonstrations. En particulier, les temps s'allongent beaucoup lorsqu'il y a un redémarrage dans la solution. Cela vient du fait qu'il faut examiner tous les couples de noeuds de façon à déterminer le s noeuds les plus proches au sens de la fonction d'évaluation.

Composition du Programme

Graphe des appels des sous programmes


RECUR
RESULT
IPERE
LIRREG
PREUVE
NORMA
ECLAIR
MERE
EXDAN
CALCUL
CALCON
SORTER
DISTTH
SELECT
COMPL
TROUV
COMPAR
PROUV
LISPRE
CALLOG
VALID
UNIF
COPIE
Amélio
NORPOL
TABLE
LIPOL
PRELIS
CONST
GROUPE

Rôle de chacun de ces programmes

RECUR	Programme principal, appelle la lecture des règles et du théorème à démontrer. Il génère la formule à vérifier au rang 1 puis au rang (n+1) et appelle PROUV . Il appelle l'édition des résultats.
LIRREG	Lecture des règles. Elles sont classées par niveaux
PRELIS	Transformation polonaise préfixée → listes.
LIPOL	Lecture d'une polonaise comme une chaine de caractères
CONST	Algorithme qui fabrique en entrée la valeur d'un nombre lu comme une chaîne de caractères.
GROUPE	regroupement des caractères pour les connectives qui s'écrivent sur plus d'un caractère.
RESULT	Edition des résultats : preuve s'il y a eu succès. Résultats intermédiaires s'il y a eu échec.
PREUVE	Extraction de la preuve de l'arborescence en cas de succès.
IPERE	Fonction père dans l'arborescence développée à partir d'une expression.
ECLAIR	Sortie en clair sur le papier des chaînes codées. (la représentation interne est entièrement numérique).
PROUV	Prouveur. Méthode de Siklossy aménagée.
SELECT	Sélecteur de règles.
NORMA	Normalisation des sommes et produits .
DISTTH	Fonction d'évaluation de distance entre deux formules.
COMPL	Fonction d'évaluation de compléxité d'une formule .
LISPRE	Transformation liste → polonaise préfixée.
NORPOL	Normalisation des polynômes à une indéterminée.
SORTER	Sortie du terme commençant à un symbole donné dans une polonaise préfixée.
CALCON CALCUL	Calculs sur les constantes.
MERE	Fonction permettant de remonter dans l'arborescence d'une expression représentée en liste .

...

EXDAN	Exécute DAN : substitution d'un terme à un autre dans une liste. remise en ordre des pointeurs.
COMPAR	Comparaison de deux expressions (en listes).
TROUV	Permet de trouver un terme dans un expression .
AMELIO	Amélioration du problème posé (application des productions simplificatrices).
UNIF 1	Algorithme d'unification.
VALID	Examine la validité du résultat de l'unification dans le cas des règles conditionnelles.
COPIE	Recopie de termes.
CALLOG	Calcul logique sur les conditions des règles conditionnelles.

Valeurs de certains paramètres importants dans le programme

- Les listes ont été dimensionnées à 4 500 . Cela permet de stocker l'équivalent de 200 formules de longueur moyenne 22 - 23 symboles.

- Les règles sont classées en 10 niveaux, chacun pouvant accueillir au plus 25 règles.

- Le nombre total maximum de noeuds d'arborescence de développement est fixé à 200. le développement de l'arborescence du membre G est plus poussé que celui du membre droite (D) . L'arborescence D a au plus un nombre de noeuds égal au 1/3 du nombre de noeuds de l'arborescence G . Cette règle permet de développer au plus 150 noeuds pour G et 50 noeuds pour D. En fait nous avons travaillé avec 80 noeuds pour G et 26 noeuds pour D. Les arborescences ont donc toujours gardé une taille très raisonnable.

- Le nombre maximum de redémarrages a été fixé à 3. Sur la méthode préconisée, il serait intéressant d'examiner la taille optimale pour qu'un redémarrage soit profitable. Une étude systématique dans ce sens permettrait sans doute de savoir si l'on a intérêt à faire beaucoup de redémarrages avec de petites arborescences ou à faire peu de redémarrages avec de grosses arborescences. Nous sommes là amenés à faire un choix espace-temps (dans la mesure où l'arborescence est coûteuse en espace et le redémarrage couteux en temps (il faut déterminer les noeuds les plus proches)). C'est aussi un choix combinatoire-heuristiques dans la mesure où le développement arborescent est dans notre cas combinatoire et où le redémarrage se fait

...

sur des noeuds évalués essentiellement à partir d'heuristiques. Il semble qu'un compromis a été trouvé afin d'équilibrer ces différents paramètres.

Formules effectivement vérifiées par le programme

Afin de se rendre compte de l'efficacité de la méthode (en particulier de la nécéssité de développer le membre droit) et de l'intérêt de la notion de redémarrage, nous accompagnerons chaque formule de la structure de la preuve fournie.

Notations

S indiquera une simplification du problème par application d'une production simplificatrice.

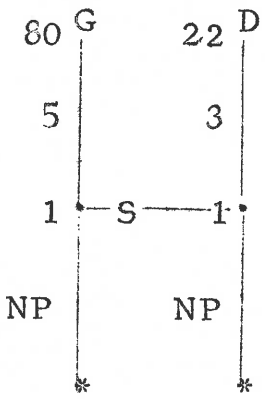
NP indiquera normalisation de polynôme.

n n sur l'arc indiquera le nombre de pas effectués entre deux redémarrages.

p. un gros point indique un démarrage ou un redémarrage (c'est la tête d'une arborescence à un moment donné. Le nombre p écrit à coté indique le nombre de noeuds de cette arborescence (quand p = 1, on l'omettra).

* Formule retrouvée dans l'autre membre.

Exemple



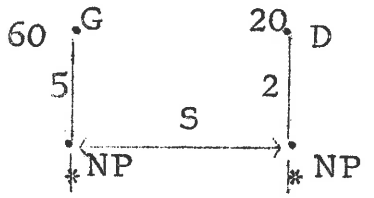
indique que pour montrer que G = D le programme a développé 80 noeuds à partir de G, 22 à partir de D. Il a alors simplifié le problème au moment du redémarrage. Il a normalisé le membre gauche, puis le membre droite et il a pu conclure à l'identité. La démonstration fait alors 10 pas (en comptant pour 1 la normalisation d'un polynôme).

- Sauf précision supplémentaire sur les indices des sommes, nous écrirons Σ pour $\sum_{i=1}^{n}$

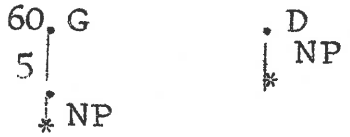
...

Remarque : Généralement, la vérification au rang 1 ne pose pas trop de problèmes aussi nous n'en donnerons les résultats que pour les cas où la preuve n'est pas immédiate.

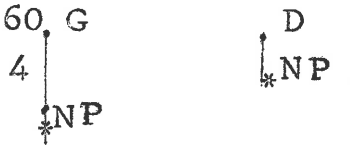
1 - $\Sigma i = \frac{n(n+1)}{2}$



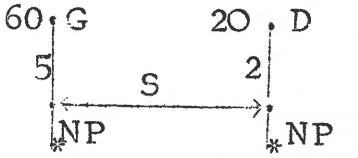
2 - $\Sigma 2i = n(n+1)$



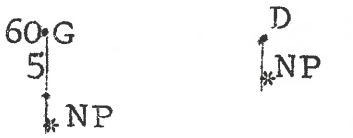
3 - $\Sigma 2i - 1 = n^2$



4 - $\Sigma i^2 = \frac{n(n+1)(2n+1)}{6}$

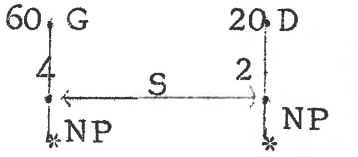


5 - $\Sigma 4i - 1 = n(2n+1)$

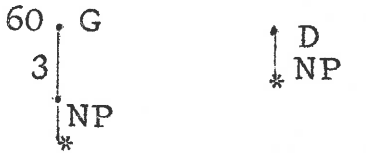


6 - $\Sigma 4i - 3 = n(2n - 1)$

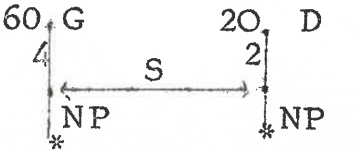
7 - $\Sigma i^2 - i = \frac{n(n^2 - 1)}{3}$



8 - $\Sigma 3(i+4)(i-3) = n(n+1)(n+2) - 36n$

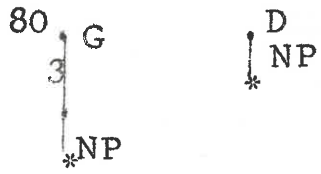


9 - $\Sigma i(i+1) = \frac{n(n+1)(n+2)}{3}$

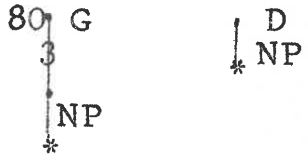


...

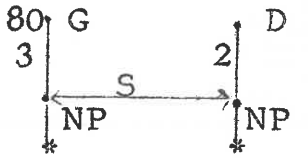
10 - $\Sigma (2i-1)^3 = n^2(2n^2-1)$



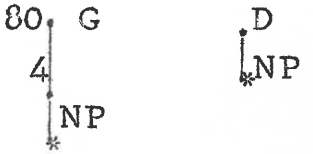
11 - $\Sigma (2i)^3 = 2n^2(n+1)^2$



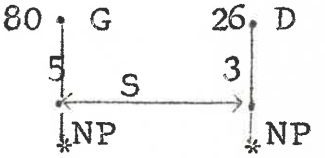
12 - $\Sigma (2i-1)^2 = \frac{n(2n-1)(2n+1)}{3}$



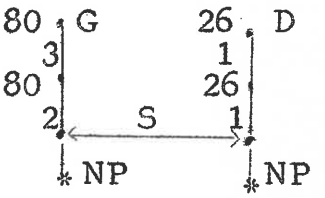
13 - $\Sigma (2i)^2 = n(n+1)(2n+1).\frac{2}{3}$



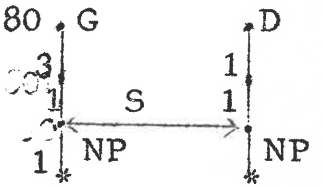
14 - $\Sigma i^3 = \frac{n^2(n+1)^2}{4}$



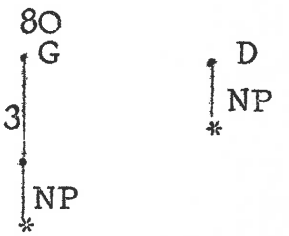
15 - $\Sigma i(i+1)(i+2) = \frac{n(n+1)(n+2)(n+3)}{4}$



16 - $\Sigma i(i+2)(i+4) = \frac{n(n+1)(n+4)(n+5)}{4}$



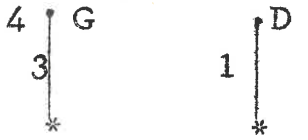
17 - $\Sigma 4(i+1)(i+3)(i+5) = n(n+1)(n^2+13n+52)+60n$



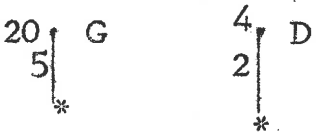
...

18 - $\Sigma a_i = \sum_{j=1}^{n} a_j$

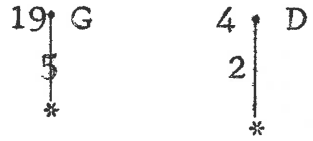
19 - $\Sigma\left(\sum_{j=1}^{m} a_{ij}\right) = \sum_{j=1}^{m}\left(\Sigma a_{ij}\right)$



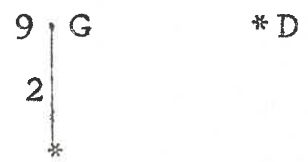
20 - $\Sigma(a+b) = \Sigma a + \Sigma b$



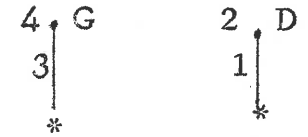
21 - $\Sigma a_i + b_i = \sum_{k=1}^{n} a_k + \sum_{l=1}^{n} b_l$



22 - $(\Sigma a_i)\left(\sum_{j=1}^{n} b_j\right) = \Sigma\left(\sum_{j=1}^{m} a_i\, b_j\right)$



23 - $(\Sigma a_i).b = \Sigma(a_i.b)$



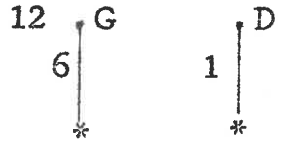
24 - $(\Sigma a_i)\left(\sum_{j=1}^{m} b_j\right) = \Sigma\left[\left(\sum_{j=1}^{m} b_j\right).a_i\right]$

25 - $(\Sigma a_i)\left(\sum_{j=1}^{m} b_j\right) = \Sigma\left[a_i \cdot \sum_{j=1}^{m} b_j\right]$

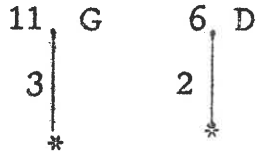
26 - $\frac{d^n}{dx^n}(KA) = K\frac{d^n}{dx^n}A$

...

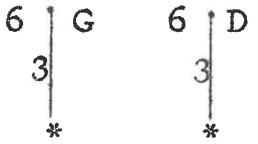
27 - $\frac{d^n}{dx^n}(KA_{(x)}) = K\frac{d^n}{dx^n}(A_{(x)})$



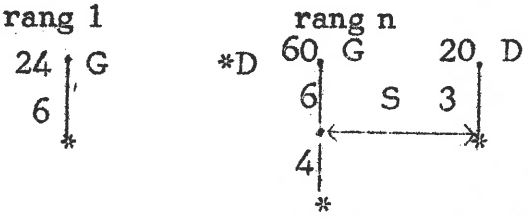
28 - $\frac{d^n}{dx^n}(A + B) = \frac{d^n}{dx^n}A + \frac{d^n}{dx^n}B$



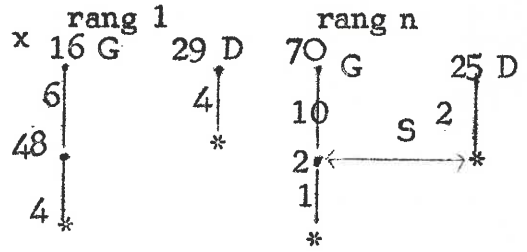
29 - $\frac{d^n}{dx^n}(A_{(x)} + B_{(x)}) = \frac{d^n}{dx^n}A_{(x)} + \frac{d^n}{dx^n}B_{(x)}$



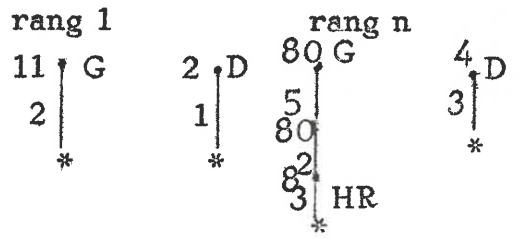
30 - $\frac{d^n}{dx^n}(x e^x) = (x + n)e^x$



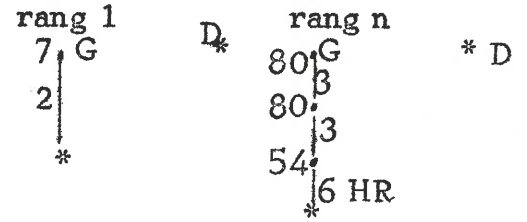
31 - $\frac{d^n}{dx^n}(x + A)e^x = (x + A + n)e^x$



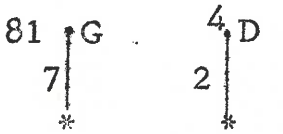
32 - $\frac{d^n(x^n)}{dx^n} = n\,!$



33 - $\frac{d^n}{dx^n}(x^{n-1}) = 0$

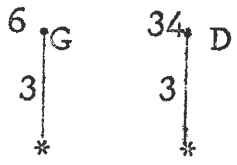


34 - $\begin{cases} u_1 = 3A + 2 \\ u_{n+1} = 3u_n + 2 \end{cases} \Rightarrow u_n = 3^n(1+A) - 1$

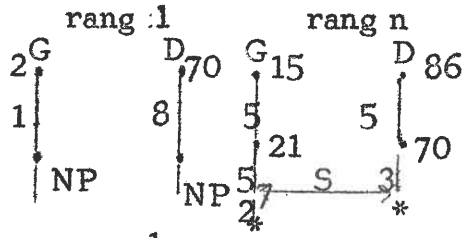


...

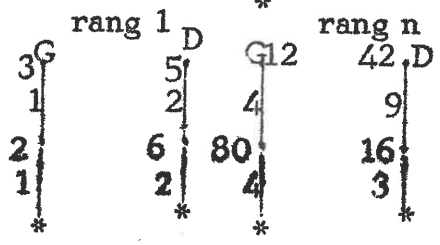
35- $\begin{cases} u_1 = A \\ u_{n+1} = B\,u_n \end{cases} \Rightarrow u_n = A.B^{n-1}$



36- $u_m = 3^m(1+A)-1 \Rightarrow u_n = 3u_{n-1}+2$



37- $u_m = A.B^{m-1} \Rightarrow u_{n+1} = B\,u_n$



Tous ces exercices n'ont pas été traités avec exactement le même jeu de règles. En effet notre programme, est capable de vérifier par récurrence un certain nombre d'égalités que nous sommes obligés de lui donner comme règle pour d'autres exercices. Pour établir une telle égalité, nous devions enlever la règle correspondante sinon la preuve aurait été immédiate avec une unification. Cette contrainte qui consiste à être obligé de fournir comme règle une formule que le programme sait vérifier par ailleurs ne doit pas surprendre : Cela tient essentiellement au fait que le programme dispose de deux dispositifs d'inférences parfaitement indépendants. D'une part dans RECUR il dispose de la récurrence, d'autre part dans PROUV, il utilise l'inférence par substitution.

Cette indépendance totale entre les deux modes d'inférence empêche le programme de résoudre un certain nombre de problèmes.

Exemple : Soit à résoudre $\sum_{i=1}^{n} (i^3) = (\sum_{i=1}^{n} i)^2$

- au rang 1:pas de problème
- passage de n à (n+1)

(HR) $(\sum_{i=1}^{n} i^3) = (\sum_{i=1}^{n} i)^2$

...

$$\left(\sum_{i=1}^{n+1} i^3\right) \stackrel{?}{=} \left(\sum_{i=1}^{n+1} i\right)^2$$

$$\sum_{i=1}^{n} i^3 + (n+1)^3 \qquad \left(\sum_{i=1}^{n} i + (n+1)\right)^2$$

HR

$$\left(\sum_{i=1}^{n} i\right)^2 + (n+1)^3 \qquad \left(\sum_{i=1}^{n} i\right)^2 + 2(n+1)\sum_{i=1}^{n} i + (n+1)^2$$

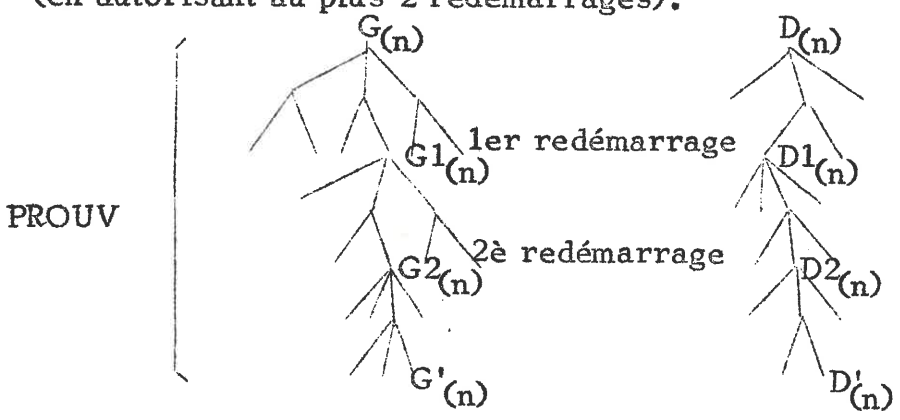
après simplification le problème prend la forme

$$(n+1)^2 \stackrel{?}{=} 2 \cdot \sum_{i=1}^{n} i + (n+1)$$

Des productions de modification de la forme du problème peuvent ramener à $\sum_{i=1}^{n} i \stackrel{?}{=} \frac{n(n+1)}{2}$

mais ce problème ne pourra pas être résolu par PROUV si on ne lui donne pas justement comme règle $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Or, il n'est pas naturel de donner au programme des règles aussi particulières. Et pourtant le programme dans son entier sait résoudre ce problème. On pourrait penser qu'il est suffisant dansle cas où PROUV échoue de renvoyer le problème à RECUR. Dans l'exemple ci-dessus, il est probable que le programme pourrait alors aboutir. Ce renvoi vers RECUR en cas d'échec de PROUV n'est cependant pas toujours souhaitable. En effet lorsque PROUV échoue, c'est souvent parce que le problème est compliqué et il est très probable que le problème qui serait renvoyé à RECUR serait plus compliqué encore que le problème initial. On travaillerait alors sur le schéma suivant: (en autorisant au plus 2 redémarrages).



...

Il resterait à démontrer après l'échec de PROUV que $G'_{(n)} = D'_{(n)}$ (en supposant que $G'_{(n)}$ et $D'_{(n)}$ soient choisis avec la même fonction d'évaluation). Ce problème pourrait à son tour être envoyé à RECUR pour être tenté par récurrence. L'expérience nous a montré que lorsqu'il y a eu ainsi échec, le problème ($G' \stackrel{?}{=} D'$) qui reste est complexe. Il serait peut être possible de filtrer à l'aide de quelques heuristiques les problèmes renvoyés. Nous n'avons pas expérimenté ces possibilités.

Démonstrations complètes obtenues par le programme

Le programme redonne la carte lue en donnée après le message "THEOREME A DEMONTRER" (le $ indique la fin de la chaîne de caractères. Le programme sort également chaque problème qui est envoyé à PROUV après le message "IL FAUT PROUVER" . On voit ainsi apparaître la formule au rang 1, au rang (n+1) et les formules sur lesquelles il repart à un redémarrage après le message : "IL RESTE A PROUVER " . Le message "IL SUFFIT DE PROUVER indique qu'à un redémarrage une production a pu simplifier le problème. Le message "DEVELOPPEMENTS : A GAUCHE XX NOEUDS - A DROITE YY NOEUDS " permet de connaître la volume des arborescences crées au moments de la sortie d'une démonstration.

Au moment du redémarrage, l'information

MING α MIND β MINDIS γ

est inscrite. Elle signifie que l'on est reparti du noeud α dans l'arborescence de G, du noeud β dans l'arborescence de D et que la distance donnée par la fonction d'évaluation était γ à ce moment là. Cette information γ est intéressante car on peut examiner comment la distance décroit d'un redémarrage au suivant.

Le message "SYSTEME BLOQUE XX NOEUDS" indique qu'une arborescence a eu tous ses noeuds morts avant d'avoir atteint la taille maximumpermise . XX précise le nombre de noeuds de cette arborescence. Le message "DEBORDEMENT" indique que les listes ont débordé l'espace alloué. Dans ces cas là, on relance sur l'autre membre si possible. Sinon, on effectue un redémarrage. La règle niveau 10 est l'hypothèse de récurrence.

Nous donnerons d'abord un exemple de jeu de règles utilisées. Les

...

justifications sont données en précisant la règle (niveau et numéro dans le niveau) utilisée et l'"endroit" où cette règle a été appliquée. (l'endroit donne le rang dans la chaîne de symboles de la connective principale du terme qui va être modifié par la règle).

Jeu des Règles utilisées.

(le $ indique la fin de la chaîne de caractères en entrée)

```
          $ENTRY
REGLES
 NIVEAU =   1
 IMP = / U W / V W = U V $
 IMP = * U W * V W = U V $
 IMP = + U W + V W = U V $
 IMP = / U V / T W = * U W * T V $
 $
 REGLES A INVERSER
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 NIVEAU =   2
 = * 1 U U $
 = * U 1 U $
 = + U 0 U $
 = * 0 U 0 $
 = * U 0 0 $
 = - U V + U * --- 1 V $
 = --- * U V * --- 1 * U V $
 = --- + U V + * --- 1 U * --- 1 V $
 = * + U V W + * U W * V W $
 = * U + V W + * U V * U W $
 = + U + V W + + U V W $
 = * U * V W * * U V W $
 = + U U * 2 U $
 = - + U V W + U - V W $
 = + + U V W + + U W V $
 = + + U V W + + V W U $
 = + 0 U U $
 = - + U V W + V - U W $
 $
 REGLES A INVERSER
  9 10 11 12 13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 NIVEAU =   3
 = PUI / U V W / PUI U W PUI V W $
 = PUI * U V W * PUI U W PUI V W $
 = PUI U - V W / PUI U V PUI U W $
 = PUI U + V W * PUI U V PUI U W $
 = PUI U 2 * U U $
 = PUI U 0 1 $
 = PUI U 1 U $
 = PUI U 3 * * U U U $
 = PUI U 4 * * * U U U U $
 $
 REGLES A INVERSER
  1  2  3  4  5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 NIVEAU =   4
 = LOG * U V + LOG U LOG V $
 = LOG / U V - LOG U LOG V $
 = EXP - U V / EXP U EXP V $
 = EXP + U V * EXP U EXP V $
 $
 REGLES A INVERSER
  1  2  3  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Productions simplificatrices

(indique les règles qu'il faut également considérer de droite à gauche

```
NIVEAU =    5
 = SIN + U V + * SIN U COS V * SIN V COS U $
 = SIN - U V - * SIN U COS V * SIN V COS U $
 = COS + U V - * COS U COS V * SIN U SIN V $
 = COS - U V + * COS U COS V * SIN U SIN V $
 = TG U / SIN U COS U $
 = COT U / COS U SIN U $
  $
REGLES A INVERSER
   1  2  3  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

NIVEAU =    6
 = SIG IND P U DEA V V DAN U = P V $
 = SIG IND P T DEA U + 1 V + SIG IND P T DEA U V DAN T = P + 1 V $
 = PRD IND P U DEA V V DAN U = P V $
 = PRD IND P T DEA U + 1 V * PRD IND P T DEA U V DAN T = P + 1 V $
 = * U SIG IND P V DEA R S SI NON DAN P U ALO SIG IND P * U V DEA R S $
 = SIG IND P + U V DEA K L + SIG IND P U DEA K L SIG IND P V DEA K L $
 = * SIG IND P V DEA R S U SI NON DAN P U ALO SIG IND P * U V DEA R S $
 = SIG IND P U DEA R S SI NON DAN P U ALO * - S - R 1 U $
 = SIG IND P * U V DEA R S SI NON DAN P U ALO * SIG IND P V DEA R S U $
 = SIG IND P * U V DEA R S SI NON DAN P V ALO * SIG IND P U DEA R S V $
 = PRD IND I I DEA 1 0 1 $
 = PRD  IND I U DEA 1 2 * DAN U = I 1 DAN U = I 2 $
 = PRD  IND I U DEA P + 2 T * * PRD IND I U DEA P T DAN U = I + 1 T
DAN U = I + 2 T $
  $
REGLES A INVERSER
   6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

NIVEAU =    7
 = + / U V / W T / + * U T * V W * V T $
 = - / U V / W T / - * U T * V W * V T $
 = + / U V W / + * V W U V $
 = - / U V W / - U * V W V $
 = * / U V / W T / * U W * V T $
 = / * V U * W U / V W $
 = / / U V / W T * / U V / T W $
 = / U / V W / * U W V $
 = / / U V W / U * V W $
 = / * U V U V $
  = * / U V W / * U W V $
 = * U / V W / * U V W $
  $
REGLES A INVERSER
   1  2  3  4  5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```
 NIVEAU =    8
= DER IND 1 X X 1 $
 = DER IND 1 X U SI NON DAN X U ALO 0 $
= DER IND + L M X U DER IND L X DER IND M X U $
 = DER IND 1 X + U V + DER IND 1 X U DER IND 1 X V $
= DER IND 1 X - U V - DER IND 1 X U DER IND 1 X V $
= DER IND 1 X * U V + * U DER IND 1 X V * V DER IND 1 X U $
= DER IND 1 X / U V / - * V DER IND 1 X U * U DER IND 1 X V PUI V 2 $
 = DER IND 1 X RAC U / DER IND 1 X U * 2 RAC U $
= DER IND 1 X PUI U V * * V PUI U - V 1 DER IND 1 X U $
 = DER IND 1 X SIN U * COS U DER IND 1 X U $
 = DER IND 1 X COS U * --- SIN U DER IND 1 X U $


 = DER IND 1 X LOG U / DER IND 1 X U U $
= DER IND 1 X EXP U * EXP U DER IND 1 X U $
 = DER IND + L M X U DER IND M X DER IND L X U $
= DER IND P X * U V SI NON DAN X U ALO * DER IND P X V U $
 = DER IND P X * U V SI NON DAN X V ALO * DER IND P X U V $
= DER IND P X + U V + DER IND P X U DER IND P X V $
 $
 REGLES A INVERSER
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 NIVEAU =    9        utilisé pour la définition des suites récurrentes
 $
 REGLES A INVERSER
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 NIVEAU =    0
```

On veut vérifier ici que $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

jeu de règles : celui qui est donné précédemment.
(avec au niveau 7 règle 13 : = / UV * U/ 1 V)

```
THEOREME A DEMONTRER
= SIG IND I I DEA I N / * N + N 1.2 3

IL FAUT PROUVER
= SIG IND I I DEA 1.0 1.0 1.0
DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
NIVEAU 6 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
1.0

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= SIG IND I I DEA 1.0 + 1.0 N / * + + 1.0 N 1.0 + 1.0 N 2.0
SYSTEME BLOQUE 13 NOEUDS

LE MEMBRE GAUCHE DONNE

MING 12 MIND 78 MINDIS -4
NIVEAU 6 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ SIG IND I I DEA 1.0 N + 1.0 N
NIVEAU (10) REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 3
+ / * + 1.0 N N 2.0 + 1.0 N
NIVEAU 2 REGLE 9 A L'ENDROIT 3 DONNE L'EXPRESSION 5
+ / + * 1.0 N * N N 2.0 + 1.0 N
NIVEAU 2 REGLE 11 A L'ENDROIT 1 DONNE L'EXPRESSION 9
+ + / + * 1.0 N * N N 2.0 1.0 N
NIVEAU 2 REGLE 15 A L'ENDROIT 1 DONNE L'EXPRESSION 12
+ + / + * 1.0 N * N N 2.0 N 1.0

LE MEMBRE DROIT DONNE

NIVEAU 7 REGLE 13 A L'ENDROIT 1 DONNE L'EXPRESSION 15
* * + + 1.0 N 1.0 + 1.0 N 0.5
NIVEAU 2 REGLE 9 A L'ENDROIT 2 DONNE L'EXPRESSION 19
* + * + 1.0 N + 1.0 N * + 1.0 N 1.0 0.5
NIVEAU 2 REGLE 2 A L'ENDROIT 10 DONNE L'EXPRESSION 33
* + * + 1.0 N + 1.0 N + 1.0 N 0.5
NIVEAU 2 REGLE 9 A L'ENDROIT 1 DONNE L'EXPRESSION 78
* * * + 1.0 N + 1.0 N 0.5 * + 1.0 N 0.5

IL RESTE A PROUVER
```

```
= + + / + * 1.0 N * N N 2.0 N 1.0 * * * + 1.0 N + 1.0 N 0.5 * + 1.
N 0.5
JE NORMALISE LE POLYNOME DU MEMBRE DROIT
+ + * PUI N 2.0 0.5 * PUI N 1.0 1.5 * PUI N 0.0 1.0

LE MEMBRE GAUCHE DONNE

MING 61 MIND 81 MINDIS 55
NIVEAU 7 REGLE 13 A L'ENDROIT 3 DONNE L'EXPRESSION 3
+ + * + * 1.0 N * N N 0.5 N 1.0
NIVEAU 2 REGLE 17 A L'ENDROIT 4 DONNE L'EXPRESSION 19
+ + * * + 1.0 N N 0.5 N 1.0
NIVEAU 2 REGLE 20 A L'ENDROIT 3 DONNE L'EXPRESSION 61
+ + * * 0.5 N + 1.0 N N 1.0

LE MEMBRE DROIT DONNE

NIVEAU 0 REGLE 0 A L'ENDROIT 0 DONNE L'EXPRESSION 81
+ + * PUI N 2.0 0.5 * PUI N 1.0 1.5 * PUI N 0.0 1.0

IL RESTE A PROUVER

= + + * * 0.5 N + 1.0 N N 1.0 + + * PUI N 2.0 0.5 * PUI N 1.0 1.5
* PUI N 0.0 1.0
JE NORMALISE LE POLYNOME DU MEMBRE GAUCHE
+ + * PUI N 2.0 0.5 * PUI N 1.0 1.5 * PUI N 0.0 1.0
DEVELOPPEMENTS:A GAUCHE 1 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
EGALITE EVIDENTE

FORMULE VRAIE POUR TOUT N
```

Au rang 1 : c'est immédiat

Au rang n : On a deux redémarrages

- au premier, on normalise le membre droit
- au second on normalise le membre gauche et l'on conclut

(temps total d'exécution (avec lecture des règles) : 23 s)

On vérifie ici que $\sum_{i=1}^{n} (i^2 - i) = \frac{n.(n^2-1)}{3}$

Jeu de règles : celui qui est donné précédemment

```
THEOREME A DEMONTRER
= SIG IND I - PUI I 2 I DEA 1 N / * N - PUI N 2 1 3 $

IL FAUT PROUVER
= SIG IND I - PUI I 2.0 I DEA 1.0 1.0 0.0
DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
NIVEAU 6 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
0.0

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= SIG IND I - PUI I 2.0 I DEA 1.0 + 1.0 N / * - PUI + 1.0 N 2.0 1.0
+ 1.0 N 3.0

LE MEMBRE GAUCHE DONNE

MING 51 MIND 99 MINDIS 16
NIVEAU 6 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ SIG IND I - PUI I 2.0 I DEA 1.0 N - PUI + 1.0 N 2.0 + 1.0 N
NIVEAU 10 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 5
+ / * - PUI N 2.0 1.0 N 3.0 - PUI + 1.0 N 2.0 + 1.0 N
NIVEAU 2 REGLE 6 A L'ENDROIT 11 DONNE L'EXPRESSION 17
+ + PUI + 1.0 N 2.0 * + 1.0 N -1.0 / * - PUI N 2.0 1.0 N 3.0
NIVEAU 2 REGLE 15 A L'ENDROIT 1 DONNE L'EXPRESSION 51
+ + / * - PUI N 2.0 1.0 N 3.0 PUI + 1.0 N 2.0 * + 1.0 N -1.0

LE MEMBRE DROIT DONNE

NIVEAU 2 REGLE 10 A L'ENDROIT 2 DONNE L'EXPRESSION 85
/ + * - PUI + 1.0 N 2.0 1.0 1.0 * - PUI + 1.0 N 2.0 1.0 N 3.0
NIVEAU 2 REGLE 6 A L'ENDROIT 13 DONNE L'EXPRESSION 99
/ + * + PUI + 1.0 N 2.0 -1.0 N * - PUI + 1.0 N 2.0 1.0 1.0 3.0

IL RESTE A PROUVER

= + + / * - PUI N 2.0 1.0 N 3.0 PUI + 1.0 N 2.0 * + 1.0 N -1.0 / +
* + PUI + 1.0 N 2.0 -1.0 N * - PUI + 1.0 N 2.0 1.0 1.0 3.0
```

```
LE MEMBRE GAUCHE DONNE

MING= 42 MIND= 97 MINDIS= 167
 NIVEAU 7 REGLE 3 A L'ENDROIT 2 DONNE L'EXPRESSION 2
+ / + * PUI + 1.0 N 2.0 3.0 * - PUI N 2.0 1.0 N 3.0 * + 1.0 N -1.0
 NIVEAU 7 REGLE 3 A L'ENDROIT 1 DONNE L'EXPRESSION 10
/ + + * PUI + 1.0 N 2.0 3.0 * - PUI N 2.0 1.0 N * * + 1.0 N -1.0 3
 3.0
 NIVEAU 3 REGLE 5 A L'ENDROIT 5 DONNE L'EXPRESSION 42
 / + + * * + 1.0 N + 1.0 N 3.0 * - PUI N 2.0 1.0 N * * + 1.0 N -1.0
 3.0 3.0

LE MEMBRE DROIT DONNE

 NIVEAU 3 REGLE 5 A L'ENDROIT 5 DONNE L'EXPRESSION 83
 / + * + * + 1.0 N + 1.0 N -1.0 N * - PUI + 1.0 N 2.0 1.0 1.0 3.0
 NIVEAU 2 REGLE 9 A L'ENDROIT 3 DONNE L'EXPRESSION 97
 / + + * * + 1.0 N + 1.0 N N * -1.0 N * - PUI + 1.0 N 2.0 1.0 1.0 3

 IL RESTE A PROUVER

 = / + + * * + 1.0 N + 1.0 N 3.0 * - PUI N 2.0 1.0 N * * + 1.0 N -1.
 3.0 3.0 / + + * * + 1.0 N + 1.0 N N * -1.0 N * - PUI + 1.0 N 2.0 1
 1.0 3.0
LA REGLE 1 NIVEAU 1 SIMPLIFIE LE PROBLEME

 IL SUFFIT DE PROUVER

 = + + * * + 1.0 N + 1.0 N 3.0 * - PUI N 2.0 1.0 N * * + 1.0 N -1.0
 3.0 + + * * + 1.0 N + 1.0 N N * -1.0 N * - PUI + 1.0 N 2.0 1.0 1.0
 JE NORMALISE LE POLYNOME DU MEMBRE GAUCHE
 + + + * PUI N 3.0 1.0 * PUI N 2.0 3.0 * PUI N 1.0 2.0 * PUI N 0.0
 0.0
 JE NORMALISE LE POLYNOME DU MEMBRE DROIT
 + + + * PUI N 3.0 1.0 * PUI N 2.0 3.0 * PUI N 1.0 2.0 * PUI N 0.0
 0.0
 DEVELOPPEMENTS:A GAUCHE 1 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
 EGALITE EVIDENTE

FORMULE VRAIE POUR TOUT N
```

Au rang 1 : c'est immédiat

Au rang n : deux redémarrages

Elle ne normalise pas le second membre tout de suite comme précédemment. Elle conserve le dénominateur 3 . Elle le simplifie au début du second redémarrage. Après normalisation des deux membres, elle peut alors conclure. (On peut remarquer la décroissance spectaculaire de MINDIS entre les deux redémarrages).

(temps total d'exécution : 51 s).

On vérifie ici que $\sum_{i=1}^{n} i\,(i+1)(i+2) = \frac{n(n+1)(n+2)(n+3)}{4}$

Jeu de règles : celui qui est donné précédemment.

```
THEOREME A DEMONTRER
= SIG IND 1 * * I + I 1 + I 2 DEA I N / * * * N + N 1 + N 2 + N 3 4 $

IL FAUT PROUVER
= SIG IND I * * + 1.0 I I + 2.0 I DEA 1.0 1.0 6.0
DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
NIVEAU 6 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
6.0

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= SIG IND I * * + 1.0 I I + 2.0 I DEA 1.0 + 1.0 N / * * * + + 1.0 N
1.0 + 1.0 N + * 1.0 N 2.0 + + 1.0 N 3.0 4.0

LE MEMBRE GAUCHE DONNE

MING 41 MIND 86 MINDIS 17
NIVEAU 6 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ * * + + 1.0 N 1.0 + 1.0 N + + 1.0 N 2.0 SIG IND I * * + 1.0 I I
+ 2.0 I DEA 1.0 N
NIVEAU 10 REGLE 1 A L'ENDROIT 17 DONNE L'EXPRESSION 6
+ / * * * + 1.0 N N + 2.0 N + 3.0 N 4.0 * * + + 1.0 N 1.0 + 1.0 N
+ + 1.0 N 2.0
NIVEAU 2 REGLE 20 A L'ENDROIT 17 DONNE L'EXPRESSION 41
+ / * * * + 1.0 N N + 2.0 N + 3.0 N 4.0 * * + + 1.0 N 2.0 + 1.0 N
* + 1.0 N 1.0

LE MEMBRE DROIT DONNE

NIVEAU 2 REGLE 10 A L'ENDROIT 4 DONNE L'EXPRESSION 86
/ * * + * + + 1.0 N 1.0 1.0 * + + 1.0 N 1.0 N + + 1.0 N 2.0 + + 1
N 3.0 4.0

IL RESTE A PROUVER

= + / * * * + 1.0 N N + 2.0 N + 3.0 N 4.0 * * + + 1.0 N 2.0 + 1.0
N + + 1.0 N 1.0 / * * + * + + 1.0 N 1.0 1.0 * + + 1.0 N 1.0 N + +
1.0 N 2.0 + + 1.0 N 3.0 4.0
```

```
LE MEMBRE GAUCHE DONNE

 MING  20  MIND  87  MINDIS -124
NIVEAU  7 REGLE  3  A L'ENDROIT  1 DONNE L'EXPRESSION  2
/ + * * * * + + 1.0 N 2.0 + 1.0 N + + 1.0 N 1.0 4.0 * * * + 1.0 N N
+ 2.0 N + 3.0 N 4.0
NIVEAU  2 REGLE 10  A L'ENDROIT  5 DONNE L'EXPRESSION  20
/ + * * * + * + + 1.0 N 2.0 1.0 * + + 1.0 N 2.0 N + + 1.0 N 1.0 4.0
* * * + 1.0 N N + 2.0 N + 3.0 N 4.0

LE MEMBRE DROIT DONNE

NIVEAU  2 REGLE 10  A L'ENDROIT  2 DONNE L'EXPRESSION  87
/ + * * + * + + 1.0 N 1.0 1.0 * + + 1.0 N 1.0 N + + 1.0 N 2.0 + 1
N * * + * + + 1.0 N 1.0 1.0 * + + 1.0 N 1.0 N + + 1.0 N 2.0 3.0
4.0

IL RESTE A PROUVER

= / * * * + * + + 1.0 N 2.0 1.0 * + + 1.0 N 2.0 N + + 1.0 N 1.0 4
* * * + 1.0 N N + 2.0 N + 3.0 N 4.0 / + * * * + * + + 1.0 N 1.0 1.0
* + + 1.0 N 1.0 N + + 1.0 N 2.0 + 1.0 N * * + * + + 1.0 N 1.0 1.0
* + + 1.0 N 1.0 N + + 1.0 N 2.0 3.0 4.0
LA REGLE  1 NIVEAU  1  SIMPLIFIE LE PROBLEME

IL SUFFIT DE PROUVER

= + * * + * + + 1.0 N 2.0 1.0 * + + 1.0 N 2.0 N + + 1.0 N 1.0 4.0
* * * + 1.0 N N + 2.0 N + 3.0 N + * * + * + + 1.0 N 1.0 1.0 * + +
1.0 N 1.0 N + + 1.0 N 2.0 + 1.0 N * * + * + + 1.0 N 1.0 1.0 * +
+ 1.0 N 1.0 N + + 1.0 N 2.0 3.0
JE NORMALISE LE POLYNOME DU MEMBRE GAUCHE
+ + + + * PUI N 4.0 1.0 * PUI N 3.0 10.0 * PUI N 2.0 35.0 * PUI N 1
50.0 * PUI N 0.0 24.0
JE NORMALISE LE POLYNOME DU MEMBRE DROIT
+ + + + * PUI N 4.0 1.0 * PUI N 3.0 10.0 * PUI N 2.0 35.0 * PUI N 1
50.0 * PUI N 0.0 24.0
DEVELOPPEMENTS:A GAUCHE  1 NOEUDS -A DROITE  0 NOEUDS

DEMONSTRATION
 EGALITE EVIDENTE

FORMULE VRAIE POUR TOUT N
```

Au rang n, on achève en normalisant les polynômes de chaque membre après avoir simplifié par 4.

Temps total d'exécution : 117 s

On vérifie ici que $\sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} \right) = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} \right)$

Jeu de règles : celui donné précédemment sans les règles (6-12),(6-13)

(La règle 6-12 ici est la règle 6-6 lue de droite à gauche)

```
THEOREME A DEMONTRER
= SIG IND I SIG IND J IND IND I J A DEA 1 M DEA 1 N
SIG IND J SIG IND I IND IND I J A DEA 1 N DEA 1 M $

IL FAUT PROUVER
= SIG IND I SIG IND J IND IND I J A DEA 1.0 M DEA 1.0 1.0 SIG IND J SIG
IND I IND IND I J A DEA 1.0 1.0 DEA 1.0 M
SYSTEME BLOQUE 2 NOEUDS
DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 2 NOEUDS

DEMONSTRATION
NIVEAU 6 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
SIG IND J IND IND 1.0 J A DEA 1.0 M

DEUXIEME PARTIE
NIVEAU 6 REGLE 1 A L'ENDROIT 4 DONNE L'EXPRESSION 4
SIG IND J IND IND 1.0 J A DEA 1.0 M

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= SIG IND I SIG IND J IND IND I J A DEA 1.0 M DEA 1.0 + 1.0 N SIG IND
J SIG IND I IND IND I J A DEA 1.0 + 1.0 N DEA 1.0 M
SYSTEME BLOQUE 4 NOEUDS
DEVELOPPEMENTS:A GAUCHE 4 NOEUDS -A DROITE 2 NOEUDS

DEMONSTRATION
NIVEAU 6 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ SIG IND I SIG IND J IND IND I J A DEA 1.0 M DEA 1.0 N SIG IND J IND
IND + 1.0 I J A DEA 1.0 M
NIVEAU 10 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 3
+ SIG IND J SIG IND I IND IND I J A DEA 1.0 N DEA 1.0 M SIG IND J IND
IND + 1.0 N J A DEA 1.0 M
NIVEAU 6 REGLE 12 A L'ENDROIT 1 DONNE L'EXPRESSION 4
SIG IND J + SIG IND I IND IND I J A DEA 1.0 N IND IND + 1.0 N J A DEA
1.0 M

DEUXIEME PARTIE
NIVEAU 6 REGLE 2 A L'ENDROIT 4 DONNE L'EXPRESSION 6
SIG IND J + SIG IND I IND IND I J A DEA 1.0 N IND IND + 1.0 N J A DEA
1.0 M

FORMULE VRAIE POUR TOUT N
```

Les arborescences sont très petites . Temps total de calcul : 3 s.

On vérifie ici que $\left(\sum_{i=1}^{n} a_i\right).B = \sum_{i=1}^{n} a_i.B$

Jeu des règles utilisées : celui donné précédemment avec au niveau 6 uniquement les règles (6-1), (6-2), (6-3), (6-4).

```
THEOREME A DEMONTRER
= * SIG IND I IND I A DEA 1 N B SIG IND I * IND I A B DEA 1 N $

IL FAUT PROUVER
= * SIG IND I IND I A DEA 1.0 1.0 B SIG IND I * IND I A B DEA 1.0 1.0
SYSTEME BLOQUE 2 NOEUDS
DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 2 NOEUDS

DEMONSTRATION
NIVEAU 6 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 2
* IND 1.0 A B

DEUXIEME PARTIE
NIVEAU 6 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 4
* IND 1.0 A B

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= * SIG IND I IND I A DEA 1.0 + 1.0 N B SIG IND I * IND I A B DEA 1.0
+ 1.0 N
SYSTEME BLOQUE 4 NOEUDS
DEVELOPPEMENTS:A GAUCHE 4 NOEUDS -A DROITE 2 NOEUDS

DEMONSTRATION
NIVEAU 6 REGLE 2 A L'ENDROIT 2 DONNE L'EXPRESSION 2
* + SIG IND I IND I A DEA 1.0 N IND + 1.0 N A B
NIVEAU 2 REGLE 9 A L'ENDROIT 1 DONNE L'EXPRESSION 3
+ * SIG IND I IND I A DEA 1.0 N B * IND + 1.0 N A B
NIVEAU 10 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 4
+ SIG IND I * IND I A B DEA 1.0 N * IND + 1.0 N A B

DEUXIEME PARTIE
NIVEAU 6 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 6
+ SIG IND I * IND I A B DEA 1.0 N * IND + 1.0 N A B

FORMULE VRAIE POUR TOUT N
```

On vérifie ici que $\sum_{j=1}^{n} (A_j + B_j) = \sum_{k=1}^{n} A_K + \sum_{L=1}^{n} B_L$

Jeu de règle précédent avec au niveau 6 seulement les règles (6 -1),(6-2), (6-3), (6 -4).

```
THEOREME A DEMONTRER
 = SIG IND J + IND J A IND J B DEA 1 N + SIG IND K IND K A DEA 1 N SIG IND
L B DEA 1 N $

IL FAUT PROUVER
 = SIG IND J + IND J A IND J B DEA 1.0 1.0 + SIG IND L IND L B DEA 1.0
 1.0 SIG IND K IND K A DEA 1.0 1.0
 SYSTEME BLOQUE 2 NOEUDS
 DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 4 NOEUDS

DEMONSTRATION
 NIVEAU 6 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ IND 1.0 A IND 1.0 B

DEUXIEME PARTIE
 NIVEAU 6 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 4
 + SIG IND K IND K A DEA 1.0 1.0 IND 1.0 B
 NIVEAU 6 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 6
 + IND 1.0 A IND 1.0 B

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
 = SIG IND J + IND J A IND J B DEA 1.0 + 1.0 N + SIG IND L IND L B DEA
 1.0 + 1.0 N SIG IND K IND K A DEA 1.0 + 1.0 N
SYSTEME BLOQUE 19 NOEUDS
 DEVELOPPEMENTS:A GAUCHE 19 NOEUDS -A DROITE 4 NOEUDS

DEMONSTRATION
 NIVEAU 6 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 2
 + SIG IND J + IND J A IND J B DEA 1.0 N + IND + 1.0 N A IND + 1.0 N B
 NIVEAU (10) REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 3
 + + SIG IND L IND L B DEA 1.0 N SIG IND K IND K A DEA 1.0 N + IND + 1.
N A IND + 1.0 N B
 NIVEAU 2 REGLE 11 A L'ENDROIT 1 DONNE L'EXPRESSION 5
+ + + SIG IND L IND L B DEA 1.0 N SIG IND K IND K A DEA 1.0 N IND + 1.
N A IND + 1.0 N B
 NIVEAU 2 REGLE 16 A L'ENDROIT 2 DONNE L'EXPRESSION 11
 + + + SIG IND K IND K A DEA 1.0 N IND + 1.0 N A SIG IND L IND L B DEA
 1.0 N IND + 1.0 N B
 NIVEAU 2 REGLE 16 A L'ENDROIT 1 DONNE L'EXPRESSION 19
+ + SIG IND L IND L B DEA 1.0 N IND + 1.0 N B + SIG IND K IND K A DEA
 1.0 N IND + 1.0 N A
```

```
DEUXIEME PARTIE
 NIVEAU  6 REGLE  2  A L'ENDROIT  2 DONNE L'EXPRESSION  21
+ + SIG IND L IND L B DEA 1.0 N IND + 1.0 N B SIG IND K IND K A DEA 1.
 + 1.0 N
 NIVEAU  6 REGLE  2  A L'ENDROIT 17 DONNE L'EXPRESSION  23


 + + SIG IND L IND L B DEA 1.0 N IND + 1.0 N B + SIG IND K IND K A DEA
1.0 N IND + 1.0 N A

FORMULE VRAIE POUR TOUT N
```

Temps total d'exécution : 8 s.

On vérifie ici que $\left(\sum_{i=1}^{n} A_i\right).\left(\sum_{j=1}^{m} B_j\right) = \sum_{i=1}^{n} \left(A_i \cdot \sum_{j=1}^{m} B_j\right)$

Jeu de règles : celui donné précédemment avec au niveau 6 uniquement les règles (6-1), (6-2), (6-3),(6-4).

```
THEOREME A DEMONTRER
 = * SIG IND I IND I A DEA I N SIG IND J IND J B DEA 1 M
SIG IND I * IND I A SIG IND J IND J B DEA 1 M DEA 1 N $

IL FAUT PROUVER
 = * SIG IND I IND I A DEA  1.0  1.0 SIG IND J IND J B DEA  1.0 M SIG IND
I * SIG IND J IND J B DEA 1.0 M IND I A DEA 1.0 1.0
 SYSTEME BLOQUE    2 NOEUDS
DEVELOPPEMENTS:A GAUCHE   2 NOEUDS -A DROITE   2 NOEUDS

DEMONSTRATION
  NIVEAU  6 REGLE  1  A L'ENDROIT  2 DONNE L'EXPRESSION   2
* SIG IND J IND J B DEA 1.0 M IND 1.0 A

 DEUXIEME PARTIE
  NIVEAU  6 REGLE  1  A L'ENDROIT  1 DONNE L'EXPRESSION   4
 * SIG IND J IND J B DEA  1.0 M IND  1.0 A

 FORMULE VRAIE AU RANG 1.


 IL FAUT PROUVER
= * SIG IND I IND I A DEA  1.0  +  1.0 N SIG IND J IND J B DEA  1.0 M SIG
 IND I * SIG IND J IND J B DEA  1.0 M IND I A DEA  1.0  +  1.0 N
SYSTEME BLOQUE    4 NOEUDS
  DEVELOPPEMENTS:A GAUCHE   4 NOEUDS -A DROITE   2 NOEUDS

 DEMONSTRATION
  NIVEAU  6 REGLE  2  A L'ENDROIT  2 DONNE L'EXPRESSION   2
 * + SIG IND I IND I A DEA  1.0 N IND +  1.0 N A SIG IND J IND J B DEA  1.
M
  NIVEAU  2 REGLE  9  A L'ENDROIT  1 DONNE L'EXPRESSION   3
+ * SIG IND I IND I A DEA  1.0 N SIG IND J IND J B DEA  1.0 M * SIG IND
 J IND J B DEA  1.0 M IND +  1.0 N A
  NIVEAU (10) REGLE  1  A L'ENDROIT  2 DONNE L'EXPRESSION   4
 + SIG IND I * SIG IND J IND J B DEA  1.0 M IND I A DEA  1.0 N * SIG IND
J IND J B DEA 1.0 M IND + 1.0 N A

 DEUXIEME PARTIE
  NIVEAU  6 REGLE  2  A L'ENDROIT  1 DONNE L'EXPRESSION   6
 + SIG IND I * SIG IND J IND J B DEA  1.0 M IND I A DEA  1.0 N * SIG IND
J IND J B DEA 1.0 M IND + 1.0 N A

FORMULE VRAIE POUR TOUT N
```

Temps total d'éxécution 3 s.

On vérifie ici que $\frac{d^n}{dx^n}(K.A(x)) = K\frac{d^n}{dx^n}(A(x))$

Jeu de règles : celui donné précédemment en enlevant au niveau 8 les règle (8-15)(8-16),(8-17)

```
THEOREME A DEMONTREP
= DER IND N X * K IND X A * K DER IND N X IND X A $

IL FAUT PROUVER
= DER IND 1.0 X * IND X A K * DER IND 1.0 X IND X A K
DEVELOPPEMENTS:A GAUCHE   5 NOEUDS -A DROITE   0 NOEUDS

DEMONSTRATION
NIVEAU 8 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ * DER IND 1.0 X IND X A K * DER IND 1.0 X K IND X A
NIVEAU 8 REGLE 2 A L'ENDROIT 12 DONNE L'EXPRESSION 3
+ * DER IND 1.0 X IND X A K * IND X A 0.0
NIVEAU 2 REGLE 5 A L'ENDROIT 11 DONNE L'EXPRESSION 4
+ * DER IND 1.0 X IND X A K 0.0
NIVEAU 2 REGLE 3 A L'ENDROIT 1 DONNE L'EXPRESSION 5
* DER IND 1.0 X IND X A K

FORMULE VRAIE AU RANG 1.


IL FAUT PROUVER
= DER IND + 1.0 N X * IND X A K * DER IND + 1.0 N X IND X A K
SYSTEME BLOQUE   12 NOEUDS
DEVELOPPEMENTS:A GAUCHE   12 NOEUDS -A DROITE   2 NOEUDS

DEMONSTRATION
NIVEAU 8 REGLE 3 A L'ENDROIT 1 DONNE L'EXPRESSION 2
DER IND 1.0 X DER IND N X * IND X A K
NIVEAU 10 REGLE 1 A L'ENDROIT 5 DONNE L'EXPRESSION 4
DER IND 1.0 X * DER IND N X IND X A K
NIVEAU 8 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 6
+ * DER IND 1.0 X DER IND N X IND X A K * DER IND N X IND X A DER IND 1
X K
NIVEAU 8 REGLE 2 A L'ENDROIT 23 DONNE L'EXPRESSION 8
+ * DER IND 1.0 X DER IND N X IND X A K * DER IND N X IND X A 0.0
NIVEAU 2 REGLE 5 A L'ENDROIT 15 DONNE L'EXPRESSION 10
+ * DER IND 1.0 X DER IND N X IND X A K 0.0
NIVEAU 2 REGLE 3 A L'ENDROIT 1 DONNE L'EXPRESSION 12
* DER IND 1.0 X DER IND N X IND X A K

DEUXIEME PARTIE
NIVEAU 8 REGLE 3 A L'ENDROIT 2 DONNE L'EXPRESSION 14
* DER IND 1.0 X DER IND N X IND X A K

FORMULE VRAIE POUR TOUT N
```

Temps total d'exécution : 4 s.

On vérifie ici que $\frac{d^n}{dx^n}(A(x)+B(x))=\frac{d^n}{dx^n}(A(x))+\frac{d^n}{dx^n}(B(x))$

Jeu de règles : celui donné précédemment en enlevant les règles (8-15), (8-16), (8-17).

```
THEOREME A DEMONTRER
= DER IND N X + IND X A IND X B + DER IND N X IND X A DER IND N X IND X B

IL FAUT PROUVER
= DER IND 1.0 X + IND X A IND X B + DER IND 1.0 X IND X A DER IND 1.0
X IND X B
DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
 NIVEAU 8 REGLE 4 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ DER IND 1.0 X IND X A DER IND 1.0 X IND X B

FORMULE VRAIE AU RANG 1.


IL FAUT PROUVER
= DER IND + 1.0 N X + IND X A IND X B + DER IND + 1.0 N X IND X A DER
IND + 1.0 N X IND X B
SYSTEME BLOQUE 6 NOEUDS
DEVELOPPEMENTS:A GAUCHE 6 NOEUDS -A DROITE 6 NOEUDS

DEMONSTRATION
 NIVEAU 8 REGLE 3 A L'ENDROIT 1 DONNE L'EXPRESSION 2
DER IND 1.0 X DER IND N X + IND X A IND X B
 NIVEAU (10) REGLE 1 A L'ENDROIT 5 DONNE L'EXPRESSION 4
DER IND 1.0 X + DER IND N X IND X A DER IND N X IND X B
 NIVEAU 8 REGLE 4 A L'ENDROIT 1 DONNE L'EXPRESSION 6
+ DER IND 1.0 X DER IND N X IND X A DER IND 1.0 X DER IND N X IND X B


DEUXIEME PARTIE
 NIVEAU 8 REGLE 3 A L'ENDROIT 2 DONNE L'EXPRESSION 8
+ DER IND 1.0 X DER IND N X IND X A DER IND + 1.0 N X IND X B
 NIVEAU 8 REGLE 3 A L'ENDROIT 13 DONNE L'EXPRESSION 12
+ DER IND 1.0 X DER IND N X IND X A DER IND 1.0 X DER IND N X IND X B

FORMULE VRAIE POUR TOUT N
```

Temps total d'exécution : 3 s.

Exemple montrant la nécessité de développer le membre droit. L'arborescence G est bloquée à 6 noeuds. Il y a succès dès que l'on a développé 6 noeuds de D.

On vérifie ici que $\frac{d^n}{dx^n}(x\,e^x) = (x+n)\,e^x$

Jeu de règles : celui donné précédemment

```
 THEOREME A DEMONTRER
= DER IND N X * X EXP X * + X N EXP X $

IL FAUT PROUVER
 = DER IND 1.0 X * EXP X X * + 1.0 X EXP X
DEVELOPPEMENTS:A GAUCHE 24 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
 NIVEAU 8 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ * DER IND 1.0 X EXP X X * DER IND 1.0 X X EXP X
 NIVEAU 8 REGLE 1 A L'ENDROIT 11 DONNE L'EXPRESSION 3
+ * DER IND 1.0 X EXP X X * EXP X 1.0
 NIVEAU 8 REGLE 13 A L'ENDROIT 3 DONNE L'EXPRESSION 5
+ * * DER IND 1.0 X X EXP X X * EXP X 1.0
 NIVEAU 8 REGLE 1 A L'ENDROIT 4 DONNE L'EXPRESSION 9
+ * * EXP X 1.0 X * EXP X 1.0
 NIVEAU 2 REGLE 2 A L'ENDROIT 3 DONNE L'EXPRESSION 16
+ * EXP X 1.0 * EXP X X
 NIVEAU 2 REGLE 18 A L'ENDROIT 1 DONNE L'EXPRESSION 24
* + 1.0 X EXP X

FORMULE VRAIE AU RANG 1.


IL FAUT PROUVER
 = DER IND + 1.0 N X * EXP X X * + + 1.0 N X EXP X
SYSTEME BLOQUE 93 NOEUDS


LE MEMBRE GAUCHE DONNE

 MING 48 MIND 90 MINDIS 83
NIVEAU 8 REGLE 3 A L'ENDROIT 1 DONNE L'EXPRESSION 2
DER IND 1.0 X DER IND N X * EXP X X
NIVEAU 10 REGLE 1 A L'ENDROIT 5 DONNE L'EXPRESSION 4
DER IND 1.0 X * + N X EXP X
 NIVEAU 8 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 6
+ * DER IND 1.0 X + N X EXP X * DER IND 1.0 X EXP X + N X
NIVEAU 8 REGLE 4 A L'ENDROIT 3 DONNE L'EXPRESSION 11
+ * + DER IND 1.0 X N DER IND 1.0 X X EXP X * DER IND 1.0 X EXP X + N
X
 NIVEAU 8 REGLE 1 A L'ENDROIT 9 DONNE L'EXPRESSION 21
+ * + DER IND 1.0 X N 1.0 EXP X * DER IND 1.0 X EXP X + N X
 NIVEAU 8 REGLE 2 A L'ENDROIT 4 DONNE L'EXPRESSION 48
+ * DER IND 1.0 X EXP X + N X * EXP X 1.0
```

```
LE MEMBRE DROIT DONNE

 NIVEAU  2 REGLE  9  A L'ENDROIT  1 DONNE L'EXPRESSION  82
 + * + 1.0 N EXP X * EXP X X
 NIVEAU  2 REGLE  9  A L'ENDROIT  2 DONNE L'EXPRESSION  85
 + + * EXP X 1.0 * EXP X N * EXP X X
 NIVEAU  2 REGLE 16  A L'ENDROIT  1 DONNE L'EXPRESSION  90
 + + * EXP X N * EXP X X * EXP X 1.0

 IL RESTE A PROUVER

 = + * DER IND 1.0 X EXP X + N X * EXP X 1.0 + + * EXP X N * EXP X X *
EXP X 1.0
 LA REGLE  3 NIVEAU 1  SIMPLIFIE LE PROBLEME

 IL SUFFIT DE PROUVER

 = * DER IND 1.0 X EXP X + N X + * EXP X N * EXP X X
 DEVELOPPEMENTS:A GAUCHE  22 NOEUDS -A DROITE  0 NOEUDS

DEMONSTRATION
  NIVEAU  8 REGLE 13  A L'ENDROIT  2 DONNE L'EXPRESSION  2
* * DER IND 1.0 X X EXP X + N X
  NIVEAU  8 REGLE  1  A L'ENDROIT  3 DONNE L'EXPRESSION  4
* * EXP X 1.0 + N X
  NIVEAU  2 REGLE  2  A L'ENDROIT  2 DONNE L'EXPRESSION  9
* + N X EXP X
  NIVEAU  2 REGLE  9  A L'ENDROIT  1 DONNE L'EXPRESSION  22
+ * EXP X N * EXP X X

FORMULE VRAIE POUR TOUT N
```

Au rang 1 : La preuve est obtenue en ne développant que G (c'est très proche de ce que pourrait obtenir SIKLOSSY)

Au rang n : Le programme dérive beaucoup dans le membre gauche. L'arborescence atteint son maximum (50) . On développe alors le membre droit. Il se produit une petite simplification au redémarrage. La démonstration s'achève en ne travaillant que sur le membre gauche.

Temps total d'exécution : 25 s.

On vérifie ici que $\frac{d^x}{dx^n}((x+a)e^x) = ((x+a)+n)e^x$

Jeu de règles : celui donné précédemment

```
THEOREME A DEMONTRER
= DER IND N X * + X A EXP X * + + X A N EXP X $

IL FAUT PROUVER
= DER IND 1.0 X * + X A EXP X * + + X A 1.0 EXP X
SYSTEME BLOQUE 93 NOEUDS

LE MEMBRE GAUCHE DONNE

MING 75 MIND 91 MINDIS 133
NIVEAU 8 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ * DER IND 1.0 X + X A EXP X * DER IND 1.0 X EXP X + X A
NIVEAU 8 REGLE 4 A L'ENDROIT 3 DONNE L'EXPRESSION 4
+ * + DER IND 1.0 X X DER IND 1.0 X A EXP X * DER IND 1.0 X EXP X + X
A
NIVEAU 8 REGLE 1 A L'ENDROIT 4 DONNE L'EXPRESSION 8
+ * + DER IND 1.0 X A 1.0 EXP X * DER IND 1.0 X EXP X + X A
NIVEAU 8 REGLE 2 A L'ENDROIT 4 DONNE L'EXPRESSION 23
+ * DER IND 1.0 X EXP X + X A * EXP X 1.0
NIVEAU 8 REGLE 13 A L'ENDROIT 3 DONNE L'EXPRESSION 75
+ * * DER IND 1.0 X X EXP X + X A * EXP X 1.0

LE MEMBRE DROIT DONNE

NIVEAU 2 REGLE 9 A L'ENDROIT 1 DONNE L'EXPRESSION 82
+ * + X A EXP X * EXP X 1.0
NIVEAU 2 REGLE 9 A L'ENDROIT 2 DONNE L'EXPRESSION 86
+ + * EXP X X * EXP X A * EXP X 1.0
NIVEAU 2 REGLE 16 A L'ENDROIT 1 DONNE L'EXPRESSION 91
+ + * EXP X 1.0 * EXP X A * EXP X X

IL RESTE A PROUVER

= + * * DER IND 1.0 X X EXP X + X A * EXP X 1.0 + + * EXP X 1.0 * EXP
X A * EXP X X
DEVELOPPEMENTS:A GAUCHE 60 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
NIVEAU 8 REGLE 1 A L'ENDROIT 4 DONNE L'EXPRESSION 2
+ * * EXP X 1.0 + X A * EXP X 1.0
NIVEAU 2 REGLE 2 A L'ENDROIT 3 DONNE L'EXPRESSION 6
+ * + X A EXP X * EXP X 1.0
NIVEAU 2 REGLE 9 A L'ENDROIT 2 DONNE L'EXPRESSION 21
+ + * EXP X X * EXP X A * EXP X 1.0
NIVEAU 2 REGLE 16 A L'ENDROIT 1 DONNE L'EXPRESSION 60

+ + * EXP X 1.0 * EXP X A * EXP X X

FORMULE VRAIE AU RANG 1.
```

```
 IL FAUT PROUVER
= DER IND + 1.0 N X * + X A EXP X * + + 1.0 N + X A EXP X

 LE MEMBRE GAUCHE DONNE

  MING   69  MIND  106  MINDIS  186
 NIVEAU  8 REGLE  3  A L'ENDROIT  1 DONNE L'EXPRESSION  2
 DER IND  1.0 X DER IND N X * + X A EXP X
 NIVEAU (10) REGLE  1  A L'ENDROIT  5 DONNE L'EXPRESSION  5
 DER IND  1.0 X * + + X A N EXP X
 NIVEAU  8 REGLE  6  A L'ENDROIT  1 DONNE L'EXPRESSION  10
 + * DER IND  1.0 X + + X A N EXP X * DER IND  1.0 X EXP X + + X A N
 NIVEAU  8 REGLE  4  A L'ENDROIT  3 DONNE L'EXPRESSION  25
 + * + DER IND  1.0 X + X A DER IND  1.0 X N EXP X * DER IND  1.0 X EXP X
+ + X A N
 NIVEAU  8 REGLE  2  A L'ENDROIT  11 DONNE L'EXPRESSION  69
+ * + DER IND  1.0 X + X A  0.0 EXP X * DER IND  1.0 X EXP X + + X A N

 LE MEMBRE DROIT DONNE

 NIVEAU  2 REGLE  11  A L'ENDROIT  2 DONNE L'EXPRESSION  83
 * + + +  1.0 N X A EXP X
 NIVEAU  2 REGLE  15  A L'ENDROIT  2 DONNE L'EXPRESSION  89
 * + + +  1.0 N A X EXP X
 NIVEAU  2 REGLE  9  A L'ENDROIT  1 DONNE L'EXPRESSION  106
 + * + +  1.0 N A EXP X * EXP X X

 IL RESTE A PROUVER

 = + * + DER IND  1.0 X + X A  0.0 EXP X * DER IND  1.0 X EXP X + + X A N
+ * + +  1.0 N A EXP X * EXP X X

 LE MEMBRE GAUCHE DONNE

  MING   42  MIND  799  MINDIS  79
 NIVEAU  8 REGLE  4  A L'ENDROIT  4 DONNE L'EXPRESSION  2
 + * + + DER IND  1.0 X X DER IND  1.0 X A  0.0 EXP X * DER IND  1.0 X EXP
X + + X A N
 NIVEAU  8 REGLE  1  A L'ENDROIT  5 DONNE L'EXPRESSION  9
+ * + + DER IND  1.0 X A  1.0  0.0 EXP X * DER IND  1.0 X EXP X + + X A
N
 NIVEAU  8 REGLE  2  A L'ENDROIT  5 DONNE L'EXPRESSION  42
 + * DER IND  1.0 X EXP X + + X A N * EXP X  1.0
```

```
LE MEMBRE DROIT DONNE

 NIVEAU  2 REGLE 16  A L'ENDROIT  3 DONNE L'EXPRESSION  84
+ * + + N A 1.0 EXP X * EXP X X
 NIVEAU  2 REGLE  9  A L'ENDROIT  2 DONNE L'EXPRESSION  89
+ + * + N A EXP X * EXP X 1.0 * EXP X X
 NIVEAU  2 REGLE 15  A L'ENDROIT  1 DONNE L'EXPRESSION  99
+ + * + N A EXP X * EXP X X * EXP X 1.0


IL RESTE A PROUVER

= + * DER IND 1.0 X EXP X + + X A N * EXP X 1.0 + + * + N A EXP X * EXP
 X X * EXP X 1.0
LA REGLE 3 NIVEAU 1 SIMPLIFIE LE PROBLEME

IL SUFFIT DE PROUVER

= * DER IND 1.0 X EXP X + + X A N + * + N A EXP X * EXP X X
 DEVELOPPEMENTS:A GAUCHE  80 NOEUDS -A DROITE  6 NOEUDS

DEMONSTRATION
 NIVEAU  8 REGLE 13  A L'ENDROIT  2 DONNE L'EXPRESSION  2
* * DER IND 1.0 X X EXP X + + X A N
 NIVEAU  8 REGLE  1  A L'ENDROIT  3 DONNE L'EXPRESSION  6
* + + X A N * EXP X 1.0
 NIVEAU  2 REGLE  2  A L'ENDROIT  7 DONNE L'EXPRESSION  16
* + + X A N EXP X
 NIVEAU  2 REGLE  9  A L'ENDROIT  1 DONNE L'EXPRESSION  45
+ * + X A EXP X * EXP X N


DEUXIEME PARTIE
 NIVEAU  2 REGLE  9  A L'ENDROIT  2 DONNE L'EXPRESSION  82
+ + * EXP X N * EXP X A * EXP X X
 NIVEAU  2 REGLE 16  A L'ENDROIT  1 DONNE L'EXPRESSION  84
+ + * EXP X X * EXP X A * EXP X N
 NIVEAU  2 REGLE 18  A L'ENDROIT  2 DONNE L'EXPRESSION  86
+ * + X A EXP X * EXP X N

FORMULE VRAIE POUR TOUT N
```

Au rang 1 : la preuve n'est plus immédiate. Elle nécessite un redémarrage. Le programme conclut alors en ne redéveloppant que le membre gauche.

Au rang n : deux redémarrages. Au second une petite simplification par 1.

On peut remarquer que le programme n'a jamais pu ici simplifier par e^x.

(Il le fait si on lui donne la règle supplémentaire :

= + * U V * V W * + U W Y

pour faire la mise en facteur (importance de la normalisation des produits et influence sur les règles à fournir).

Temps total d'exécution : 80 s.

On vérifie ici que $\frac{d^n}{dx^n}(x^{n-1}) = 0$

Jeu de règles : celui donné précédemment

:

:

```
THEOREME A DEMONTRER
= DER IND N X PUI X - N 1 0 $

IL FAUT PROUVER
= DER IND 1.0 X PUI X 0.0 0.0
DEVELOPPEMENTS:A GAUCHE 7 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
NIVEAU 3 REGLE 6 A L'ENDROIT 5 DONNE L'EXPRESSION 3
DER IND 1.0 X 1.0
NIVEAU 8 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 7
0.0

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= DER IND * 1.0 N X PUI X - * 1.0 N 1.0 0.0
SYSTEME BLOQUE 81 NOEUDS

LE MEMBRE GAUCHE DONNE

MING 39 MIND 81 MINDIS 385
NIVEAU 8 REGLE 14 A L'ENDROIT 1 DONNE L'EXPRESSION 3
DER IND N X DER IND 1.0 X PUI X - * 1.0 N 1.0
NIVEAU 2 REGLE 6 A L'ENDROIT 11 DONNE L'EXPRESSION 12
DER IND N X DER IND 1.0 X PUI X * * 1.0 N -1.0
NIVEAU 2 REGLE 15 A L'ENDROIT 11 DONNE L'EXPRESSION 39
DER IND N X DER IND 1.0 X PUI X * 0.0 N

LE MEMBRE DROIT DONNE

NIVEAU 0 REGLE 0 A L'ENDROIT 0 DONNE L'EXPRESSION 81
0.0

IL RESTE A PROUVER

= DER IND N X DER IND 1.0 X PUI X * 0.0 N 0.0
SYSTEME BLOQUE 81 NOEUDS
```

```
LE MEMBRE GAUCHE DONNE

MING  48  MIND  81  MINDIS  331


NIVEAU  3 REGLE  4  A L'ENDROIT  9 DONNE L'EXPRESSION  3
DER IND N X DER IND  1.0 X * PUI X  0.0 PUI X N
NIVEAU  3 REGLE  6  A L'ENDROIT 10 DONNE L'EXPRESSION 11
DER IND N X DER IND  1.0 X * PUI X N  1.0
NIVEAU  2 REGLE  2  A L'ENDROIT  9 DONNE L'EXPRESSION 48
DER IND N X DER IND  1.0 X PUI X N

LE MEMBRE DROIT DONNE

NIVEAU  0 REGLE  0  A L'ENDROIT  0 DONNE L'EXPRESSION 81
0.0

IL RESTE A PROUVER

= DER IND N X DER IND  1.0 X PUI X N  0.0
  DEVELOPPEMENTS:A GAUCHE  54 NOEUDS -A DROITE  0 NOEUDS

DEMONSTRATION
NIVEAU  8 REGLE  9  A L'ENDROIT  5 DONNE L'EXPRESSION  2
DER IND N X * * PUI X - N  1.0 N DER IND  1.0 X X
NIVEAU  8 REGLE  1  A L'ENDROIT 13 DONNE L'EXPRESSION  3
DER IND N X * * PUI X - N  1.0 N  1.0
NIVEAU  8 REGLE 16  A L'ENDROIT  1 DONNE L'EXPRESSION  7
* DER IND N X * PUI X - N  1.0 N  1.0
NIVEAU  8 REGLE 16  A L'ENDROIT  2 DONNE L'EXPRESSION 16
* * DER IND N X PUI X - N  1.0 N  1.0
NIVEAU (10) REGLE  1  A L'ENDROIT  3 DONNE L'EXPRESSION 31
* *  0.0 N  1.0
NIVEAU  2 REGLE  4  A L'ENDROIT  2 DONNE L'EXPRESSION 54
0.0

FORMULE VRAIE POUR TOUT N
```

Au rang 1 : c'est presque immédiat

Au rang n : Il faut deux redémarrages. L'hypothèse de récurrence (niveau 10) s'applique tout à fait en fin de démonstration. On peut remarquer que la fonction d'évaluation a permis de faire un bon choix à chacun de ces redémarrages. Il faut en effet bien partir en disant que

$\frac{d^{n+1}}{dx^{n+1}}(f) = \frac{d^n}{dx^n}(\frac{df}{dx})$ dans ce cas là et non $\frac{d^{n+1}}{dx^{n+1}}(f) = \frac{d}{dx}(\frac{d^n}{dx^n} f)$ comme dans les exercices précédents (où l'hypothèse de recurrence s'applique au tout début du calcul).

Temps total d'exécution : 32 s.

On vérifie ici que $\frac{d^n}{dx^n}(x^n) = n!$ (en présentant n ! sans la forme $\prod_{i=1}^{n} i$). Jeu de règles : celui donné précédemment.

```
THEOREME A DEMONTRER
= DER IND N X PUI X N PRD IND I I DEA I N S

IL FAUT PROUVER
= DER IND 1.0 X PUI X 1.0 PRD IND I I DEA 1.0 1.0
SYSTEME BLOQUE 11 NOEUDS
DEVELOPPEMENTS:A GAUCHE 11 NOEUDS -A DROITE 2 NOEUDS

DEMONSTRATION
NIVEAU 3 REGLE 7 A L'ENDROIT 5 DONNE L'EXPRESSION 3
DER IND 1.0 X X
NIVEAU 8 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 8
1.0

DEUXIEME PARTIE
NIVEAU 6 REGLE 3 A L'ENDROIT 1 DONNE L'EXPRESSION 13
1.0

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= DER IND * 1.0 N X PUI X * 1.0 N PRD IND I I DEA 1.0 * 1.0 N
SYSTEME BLOQUE 84 NOEUDS

LE MEMBRE GAUCHE DONNE

MING 70 MIND 84 MINDIS 274
NIVEAU 8 REGLE 14 A L'ENDROIT 1 DONNE L'EXPRESSION 3
DER IND N X DER IND 1.0 X PUI X * 1.0 N
NIVEAU 8 REGLE 9 A L'ENDROIT 5 DONNE L'EXPRESSION 6
DER IND N X * * PUI X - * 1.0 N 1.0 * 1.0 N DER IND 1.0 X X
NIVEAU 8 REGLE 1 A L'ENDROIT 17 DONNE L'EXPRESSION 10
DER IND N X * * PUI X - * 1.0 N 1.0 * 1.0 N 1.0
NIVEAU 2 REGLE 2 A L'ENDROIT 5 DONNE L'EXPRESSION 21
DER IND N X * PUI X - * 1.0 N 1.0 * 1.0 N
NIVEAU 8 REGLE 16 A L'ENDROIT 1 DONNE L'EXPRESSION 70
* DER IND N X PUI X - * 1.0 N 1.0 * 1.0 N
```

```
LE MEMBRE DROIT DONNE

  NIVEAU  6 REGLE  4  A L'ENDROIT  1 DONNE L'EXPRESSION  82
* PRD IND I I DEA 1.0 N + 1.0 N
  NIVEAU  2 REGLE 10  A L'ENDROIT  1 DONNE L'EXPRESSION  83
+ * PRD IND I I DEA 1.0 N 1.0 * PRD IND I I DEA 1.0 N N
  NIVEAU  2 REGLE  2  A L'ENDROIT  2 DONNE L'EXPRESSION  84
+ * PRD IND I I DEA 1.0 N N PRD IND I I DEA 1.0 N

IL RESTE A PROUVER

= * DER IND N X PUI X - + 1.0 N 1.0 + 1.0 N + * PRD IND I I DEA 1.0
  N N PRD IND I I DEA 1.0 N
SYSTEME BLOQUE  81 NOEUDS

LE MEMBRE GAUCHE DONNE

  MING  26  MIND  81  MINDIS  174
  NIVEAU  2 REGLE 18  A L'ENDROIT  8 DONNE L'EXPRESSION  6
  * DER IND N X PUI X + 0.0 N + 1.0 N
  NIVEAU  2 REGLE 17  A L'ENDROIT  8 DONNE L'EXPRESSION  26
  * DER IND N X PUI X N + 1.0 N

LE MEMBRE DROIT DONNE

  NIVEAU  0 REGLE  0  A L'ENDROIT  0 DONNE L'EXPRESSION  81
+ * PRD IND I I DEA 1.0 N N PRD IND I I DEA 1.0 N

IL RESTE A PROUVER

= * DER IND N X PUI X N + 1.0 N + * PRD IND I I DEA 1.0 N N PRD IND I
  I DEA 1.0 N
DEVELOPPEMENTS:A GAUCHE  8 NOEUDS -A DROITE  0 NOEUDS

DEMONSTRATION
  NIVEAU (10) REGLE  1  A L'ENDROIT  2 DONNE L'EXPRESSION  2
* PRD IND I I DEA 1.0 N + 1.0 N
  NIVEAU  2 REGLE 10  A L'ENDROIT  1 DONNE L'EXPRESSION  4
+ * PRD IND I I DEA 1.0 N 1.0 * PRD IND I I DEA 1.0 N N
  NIVEAU  2 REGLE  2  A L'ENDROIT  2 DONNE L'EXPRESSION  8
+ * PRD IND I I DEA 1.0 N N PRD IND I I DEA 1.0 N

FORMULE VRAIE POUR TOUT N
```

On peut faire les mêmes remarques que sur l'exercice précédent. L'hypothèse de récurrence s'applique très tard (avec les risques que cela comporte de ne plus avoir en mémoire d'expressions sur lesquelles on peut l'utiliser). On peut encore remarquer la décroissance de MINDIS entre les redémarrages.

On vérifie ici que $\begin{cases} u_1 = 3a + 2 \\ u_{n+1} = 3u_n + 2 \end{cases} \Rightarrow u_n = 3^n(1 + A) - 1$

Jeu de règles : celui donné précédemment avec au niveau 9

9 - 1 : = IND U 1 + * 3 A 2

9 - 2 : = IND U + 1 N + * 3 IND U N 2

```
THEOREME A DEMONTRER
= IND U N - * PUI 3 N * I A 1 S

IL FAUT PROUVER
= IND U 1.0 - * * 1.0 A 3.0 1.0
JE NORMALISE LE POLYNOME DU MEMBRE DROIT
+ * PUI A 1.0 3.0 * PUI A 0.0 2.0
SYSTEME BLOQUE 2 NOEUDS

LE MEMBRE GAUCHE DONNE

MING 2 MIND 3 MINDIS 29
NIVEAU 9 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
+ * 3.0 A 2.0

LE MEMBRE DROIT DONNE

NIVEAU 0 REGLE 0 A L'ENDROIT 0 DONNE L'EXPRESSION 3
+ * PUI A 1.0 3.0 * PUI A 0.0 2.0

IL RESTE A PROUVER

= + * 3.0 A 2.0 * * PUI A 1.0 3.0 * PUI A 0.0 2.0
JE NORMALISE LE POLYNOME DU MEMBRE GAUCHE
+ * PUI A 1.0 3.0 * PUI A 0.0 2.0
DEVELOPPEMENTS:A GAUCHE 1 NOEUDS -A DROITE 0 NOEUDS

DEMONSTRATION
EGALITE EVIDENTE

FORMULE VRAIE AU RANG 1.
```

```
 IL FAUT PROUVER
= IND U + 1.0 N - * PUI 3.0 + -1.0 N + 1.0 A 1.0
 DEVELOPPEMENTS:A GAUCHE 80 NOEUDS -A DROITE 5 NOEUDS

 DEMONSTRATION
 NIVEAU 9 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 2
 + * IND U N 3.0 2.0
  NIVEAU 10 REGLE 1 A L'ENDROIT 3 DONNE L'EXPRESSION 3
```

```
 + * - * PUI 3.0 N + 1.0 A 1.0 3.0 2.0
 NIVEAU 2 REGLE 6 A L'ENDROIT 3 DONNE L'EXPRESSION 4
 + * + * PUI 3.0 N + 1.0 A -1.0 3.0 2.0
 NIVEAU 2 REGLE 9 A L'ENDROIT 2 DONNE L'EXPRESSION 6
 + + * * PUI 3.0 N + 1.0 A 3.0 -3.0 2.0
 NIVEAU 2 REGLE 16 A L'ENDROIT 1 DONNE L'EXPRESSION 12
 + * * PUI 3.0 N + 1.0 A 3.0 -1.0
 NIVEAU 2 REGLE 20 A L'ENDROIT 2 DONNE L'EXPRESSION 25
 + * * + 1.0 A 3.0 PUI 3.0 N -1.0
 NIVEAU 2 REGLE 20 A L'ENDROIT 2 DONNE L'EXPRESSION 47
 + * * PUI 3.0 N 3.0 + 1.0 A -1.0
```

```
DEUXIEME PARTIE
 NIVEAU 3 REGLE 4 A L'ENDROIT 3 DONNE L'EXPRESSION 82
- * * PUI 3.0 N 3.0 + 1.0 A 1.0
 NIVEAU 2 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 85
+ * * PUI 3.0 N 3.0 + 1.0 A -1.0

FORMULE VRAIE POUR TOUT N
```

A u rang 1 : Il faut attendre le redémarrage pour normaliser le membre gauche et conclure.

Au rang n : Pas de redémarrage . Il a suffit de développer 5 noeuds du membre droit pour retrouver un noeud de l'arborescence de G. L'arborescence de G a été bien assez développée. Il était inutile d'y passer plus de temps. La recherche "aveugle" à partir de G n'aurait sans doute pas permis de conclure.

Temps total d'exécution : 14 s.

On vérifie ici $u_m = 3^m (1 + A) - 1 \Rightarrow u_n = 3 u_n - 1 + 2$. Jeu de règles : celui donné précédemment avec au niveau 9 : (9-1) = IND U M - * PUI 3 M + 1 A 1

```
THEOREME A DEMONTRER
= IND U N + * 3 IND U - N 1 2 $

IL FAUT PROUVER
= IND U 1.0 + * IND U 0.0 3.0 2.0
SYSTEME BLOQUE 7 NOEUDS
DEVELOPPEMENTS:A GAUCHE 7 NOEUDS -A DROITE 23 NOEUDS

DEMONSTRATION
NIVEAU 9 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
- * + 1.0 A 3.0 1.0
NIVEAU 2 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 3
+ * + 1.0 A 3.0 -1.0

DEUXIEME PARTIE
NIVEAU 9 REGLE 1 A L'ENDROIT 3 DONNE L'EXPRESSION 9
+ * - * + 1.0 A 1.0 1.0 3.0 2.0
NIVEAU 2 REGLE 2 A L'ENDROIT 4 DONNE L'EXPRESSION 10
+ * - + 1.0 A 1.0 3.0 2.0
NIVEAU 2 REGLE 6 A L'ENDROIT 3 DONNE L'EXPRESSION 13
+ * + + 1.0 A -1.0 3.0 2.0
NIVEAU 2 REGLE 9 A L'ENDROIT 2 DONNE L'EXPRESSION 18
+ + * + 1.0 A 3.0 -3.0 2.0
NIVEAU 2 REGLE 16 A L'ENDROIT 1 DONNE L'EXPRESSION 30
+ * + 1.0 A 3.0 -1.0

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= IND U + 1.0 N + * IND U - + 1.0 N 1.0 3.0 2.0
SYSTEME BLOQUE 14 NOEUDS

LE MEMBRE GAUCHE DONNE

MING 13 MIND 86 MINDIS -54
NIVEAU 9 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
- * PUI 3.0 + 1.0 N + 1.0 A 1.0
NIVEAU 2 REGLE 6 A L'ENDROIT 1 DONNE L'EXPRESSION 3
+ * PUI 3.0 + 1.0 N + 1.0 A -1.0
NIVEAU 2 REGLE 10 A L'ENDROIT 2 DONNE L'EXPRESSION 5
+ + * PUI 3.0 + 1.0 N 1.0 * PUI 3.0 + 1.0 N A -1.0

NIVEAU 2 REGLE 2 A L'ENDROIT 3 DONNE L'EXPRESSION 8
+ + * PUI 3.0 + 1.0 N A PUI 3.0 + 1.0 N -1.0
NIVEAU 2 REGLE 15 A L'ENDROIT 1 DONNE L'EXPRESSION 13
+ + * PUI 3.0 + 1.0 N A -1.0 PUI 3.0 + 1.0 N

LE MEMBRE DROIT DONNE

NIVEAU 9 REGLE 1 A L'ENDROIT 3 DONNE L'EXPRESSION 16
+ * - * PUI 3.0 - + 1.0 N 1.0 + 1.0 A 1.0 3.0 2.0
NIVEAU 2 REGLE 6 A L'ENDROIT 3 DONNE L'EXPRESSION 17
+ * + * PUI 3.0 - + 1.0 N 1.0 + 1.0 A -1.0 3.0 2.0
NIVEAU 2 REGLE 6 A L'ENDROIT 7 DONNE L'EXPRESSION 21
```

```
NIVEAU  2 REGLE  6  A L'ENDROIT  7 DONNE L'EXPRESSION  21
 + * + * PUI  3.0 + +  1.0 N -1.0 +  1.0 A -1.0  3.0  2.0
NIVEAU  2 REGLE  9  A L'ENDROIT  2 DONNE L'EXPRESSION  35
 + + * * PUI  3.0 + +  1.0 N -1.0 +  1.0 A  3.0 -3.0  2.0
NIVEAU  2 REGLE 20  A L'ENDROIT  3 DONNE L'EXPRESSION  86
 + + * PUI  3.0 + +  1.0 N -1.0 * +  1.0 A  3.0 -3.0  2.0

IL RESTE A PROUVER

 = + + * PUI  3.0 +  1.0 N A -1.0 PUI  3.0 +  1.0 N + + * PUI  3.0 + +  1.
N -1.0 * +  1.0 A  3.0 -3.0  2.0
SYSTEME BLOQUE  19 NOEUDS

LE MEMBRE GAUCHE DONNE

MING  19 MIND  90 MINDIS  -77
NIVEAU  3 REGLE  4  A L'ENDROIT  4 DONNE L'EXPRESSION  2
+ + * * PUI  3.0 N  3.0 A -1.0 PUI  3.0 +  1.0 N
NIVEAU  3 REGLE  4  A L'ENDROIT 11 DONNE L'EXPRESSION  6
+ + * * PUI  3.0 N  3.0 A -1.0 * PUI  3.0 N  3.0
NIVEAU  2 REGLE 15  A L'ENDROIT  1 DONNE L'EXPRESSION  12
+ + * * PUI  3.0 N  3.0 A * PUI  3.0 N  3.0 -1.0
NIVEAU  2 REGLE 20  A L'ENDROIT  3 DONNE L'EXPRESSION  17
+ + * PUI  3.0 N *  3.0 A * PUI  3.0 N  3.0 -1.0
NIVEAU  2 REGLE 18  A L'ENDROIT  2 DONNE L'EXPRESSION  19
+ * + *  3.0 A  3.0 PUI  3.0 N -1.0

LE MEMBRE DROIT DONNE

NIVEAU  2 REGLE  9  A L'ENDROIT 11 DONNE L'EXPRESSION  22
 + + * PUI  3.0 + +  1.0 N -1.0 + *  3.0 A  3.0 -3.0  2.0
NIVEAU  2 REGLE 15  A L'ENDROIT  6 DONNE L'EXPRESSION  37
 + + * PUI  3.0 +  0.0 N + *  3.0 A  3.0 -3.0  2.0
NIVEAU  2 REGLE 16  A L'ENDROIT  1 DONNE L'EXPRESSION  90
 + * PUI  3.0 +  0.0 N + *  3.0 A  3.0 -1.0

IL RESTE A PROUVER

 = + * + *  3.0 A  3.0 PUI  3.0 N -1.0 + * PUI  3.0 +  0.0 N + *  3.0 A  3
-1.0
LA REGLE  3 NIVEAU  1  SIMPLIFIE LE PROBLEME

IL SUFFIT DE PROUVER

 = * + *  3.0 A  3.0 PUI  3.0 N * PUI  3.0 +  0.0 N + *  3.0 A  3.0
SYSTEME BLOQUE  3 NOEUDS
 DEVELOPPEMENTS:A GAUCHE  3 NOEUDS -A DROITE  4 NOEUDS

IL RESTE A PROUVER

 = + * PUI  3.0 N  1.0 + *  3.0 A  3.0
SYSTEME BLOQUE  2 NOEUDS
 DEVELOPPEMENTS:A GAUCHE  2 NOEUDS -A DROITE  2 NOEUDS

IL SUFFIT DE PROUVER

 = * + *  3.0 A  3.0 PUI  3.0 N
SYSTEME BLOQUE  3 NOEUDS
 DEVELOPPEMENTS:A GAUCHE  3 NOEUDS -A DROITE  4 NOEUDS

DEMONSTRATION
NIVEAU  3 REGLE  4  A L'ENDROIT  2 DONNE L'EXPRESSION  5
 * * PUI  3.0 N  1.0 + *  3.0 A  3.0
NIVEAU  2 REGLE  2  A L'ENDROIT  2 DONNE L'EXPRESSION  7
 * + *  3.0 A  3.0 PUI  3.0 N

FORMULE VRAIE POUR TOUT N
```

Cet exercice est le dual du précédent.

Au rang 1 : La preuve n'est pas tout à fait immédiate (Il n'y a pas de redémarrage mais il faut bien développer le membre droit pour retrouver une feuille de l'arborescence du membre gauche (dont tous les noeuds sont morts)).

Au rang n : On peut remarquer que le programme n'utilise pas l'hypothèse de récurrence. C'est à dire qu'il donne une preuve directe du résultat. La preuve obtenue est longue et pas très élégante. Remarquer la décroissance de MINDIS aux redémarrages.

Temps total d'exécution : 40 s.

On vérifie ici que $\begin{cases} u_1 = A \\ u_{n+1} = B.\, u_n \end{cases} \Rightarrow u_n = A.B^{n-1}$

Jeu de règles : celui donné précédemment avec au niveau 9

9 - 1 = IND U 1 A

9 - 2 = IND U + 1 N * B IND U N

```
 THEOREME A DEMONTRER
= IND U N * A PUI B - N 1 $

 IL FAUT PROUVER
 = IND U 1.0 * PUI B 0.0 A
SYSTEME BLOQUE 2 NOEUDS
 DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 3 NOEUDS

 DEMONSTRATION
 NIVEAU 9 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
 A

 DEUXIEME PARTIE
 NIVEAU 3 REGLE 6 A L'ENDROIT 2 DONNE L'EXPRESSION 4
* 1.0 A
 NIVEAU 2 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 5
A

FORMULE VRAIE AU RANG 1.

 IL FAUT PROUVER
 = IND U + 1.0 N * PUI B - + 1.0 N 1.0 A
SYSTEME BLOQUE 6 NOEUDS
 DEVELOPPEMENTS:A GAUCHE 6 NOEUDS -A DROITE 31 NOEUDS

 DEMONSTRATION
 NIVEAU 9 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 2
 * IND U N B
 NIVEAU 10 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 3
 * * PUI B - N 1.0 A B
 NIVEAU 2 REGLE 20 A L'ENDROIT 1 DONNE L'EXPRESSION 5
 * PUI B - N 1.0 * A B

 DEUXIEME PARTIE
 NIVEAU 2 REGLE 14 A L'ENDROIT 4 DONNE L'EXPRESSION 10
* PUI B * - N 1.0 1.0 A
 NIVEAU 3 REGLE 4 A L'ENDROIT 2 DONNE L'EXPRESSION 16
* * PUI B - N 1.0 PUI B 1.0 A
 NIVEAU 3 REGLE 7 A L'ENDROIT 8 DONNE L'EXPRESSION 24
 * * PUI B - N 1.0 B A

 NIVEAU 2 REGLE 20 A L'ENDROIT 1 DONNE L'EXPRESSION 37
 * PUI B - N 1.0 * A B

FORMULE VRAIE POUR TOUT N
```

Temps total d'exécution : 6 s.

On vérifie ici que $u_m = A.B^{m-1} \Rightarrow u_{n+1} = B.u_n$

Jeu de règles : celui donné précédemment avec au niveau 9

9 - 1 = IND U M * A PUI B - M 1

```
THEOREME A DEMONTRER
= IND U + 1 N * B IND U N $

IL FAUT PROUVER
= IND U 2.0 * IND U 1.0 B
SYSTEME BLOQUE 2 NOEUDS
SYSTEME BLOQUE 5 NOEUDS

LE MEMBRE GAUCHE DONNE

MING 2 MIND 5 MINDIS -24
NIVEAU 9 REGLE 1 A L'ENDROIT 1 DONNE L'EXPRESSION 2
* PUI B 1.0 A

LE MEMBRE DROIT DONNE

NIVEAU 9 REGLE 1 A L'ENDROIT 2 DONNE L'EXPRESSION 4
* * PUI B 0.0 A B
NIVEAU 2 REGLE 20 A L'ENDROIT 1 DONNE L'EXPRESSION 5
* PUI B 0.0 * A B

IL RESTE A PROUVER

= * PUI B 1.0 A * PUI B 0.0 * A B
SYSTEME BLOQUE 2 NOEUDS
DEVELOPPEMENTS:A GAUCHE 2 NOEUDS -A DROITE 4 NOEUDS

DEMONSTRATION
NIVEAU 3 REGLE 7 A L'ENDROIT 2 DONNE L'EXPRESSION 2
* A B

DEUXIEME PARTIE
NIVEAU 3 REGLE 6 A L'ENDROIT 2 DONNE L'EXPRESSION 4
* * A B 1.0
NIVEAU 2 REGLE 2 A L'ENDROIT 1 DONNE L'EXPRESSION 6
* A B

FORMULE VRAIE AU RANG 1.

IL FAUT PROUVER
= IND U * * 1.0 N 1.0 * IND U * 1.0 N B

SYSTEME BLOQUE 12 NOEUDS
SYSTEME BLOQUE 42 NOEUDS
```

```
LE MEMBRE GAUCHE DONNE

 MING  11  MIND  42  MINDIS  -8
NIVEAU  9 REGLE  1  A L'ENDROIT  1 DONNE L'EXPRESSION  2
 * PUI B - + + 1.0 N 1.0 1.0 A
NIVEAU  2 REGLE  6  A L'ENDROIT  4 DONNE L'EXPRESSION  3
 * PUI B + + + 1.0 N 1.0 -1.0 A
NIVEAU  2 REGLE 15  A L'ENDROIT  4 DONNE L'EXPRESSION  6
 * PUI B + + + 1.0 N -1.0 1.0 A
NIVEAU  2 REGLE 16  A L'ENDROIT  5 DONNE L'EXPRESSION 11
 * PUI B + + + -1.0 N 1.0 1.0 A

LE MEMBRE DROIT DONNE

 NIVEAU 10 REGLE  1  A L'ENDROIT  2 DONNE L'EXPRESSION 14
* * IND U N B B
 NIVEAU  9 REGLE  1  A L'ENDROIT  3 DONNE L'EXPRESSION 16
* * * PUI B - N 1.0 A B B
 NIVEAU  2 REGLE  6  A L'ENDROIT  6 DONNE L'EXPRESSION 20
* * * PUI B + -1.0 N A B B
 NIVEAU  2 REGLE 20  A L'ENDROIT  1 DONNE L'EXPRESSION 27
* * PUI B + -1.0 N A * B B
 NIVEAU  2 REGLE 20  A L'ENDROIT  1 DONNE L'EXPRESSION 33
* PUI B + -1.0 N * * B B A
 NIVEAU  2 REGLE 12  A L'ENDROIT  1 DONNE L'EXPRESSION 36
* * PUI B + -1.0 N * B B A
 NIVEAU  2 REGLE 12  A L'ENDROIT  2 DONNE L'EXPRESSION 38
* * * PUI B + -1.0 N B B A
 NIVEAU  2 REGLE 20  A L'ENDROIT  1 DONNE L'EXPRESSION 40
* * PUI B + -1.0 N B * A B
 NIVEAU  2 REGLE 12  A L'ENDROIT  1 DONNE L'EXPRESSION 42
* * * PUI B + -1.0 N B A B

IL RESTE A PROUVER

= * PUI B * * + -1.0 N 1.0 1.0 A * * * PUI B + -1.0 N B A B
 DEVELOPPEMENTS:A GAUCHE  80 NOEUDS -A DROITE  16 NOEUDS

DEMONSTRATION
NIVEAU  2 REGLE 16  A L'ENDROIT  4 DONNE L'EXPRESSION  4
 * PUI B * * -1.0 N 2.0 A
NIVEAU  3 REGLE  4  A L'ENDROIT  2 DONNE L'EXPRESSION 13
 * * PUI B * -1.0 N PUI B 2.0 A
NIVEAU  3 REGLE  5  A L'ENDROIT  8 DONNE L'EXPRESSION 31
 * * PUI B + -1.0 N * B B A
NIVEAU  2 REGLE 20  A L'ENDROIT  1 DONNE L'EXPRESSION 65
 * PUI B + -1.0 N * * B B A

DEUXIEME PARTIE
 NIVEAU  2 REGLE 20  A L'ENDROIT  1 DONNE L'EXPRESSION 83
* * PUI B + -1.0 N B * A B
 NIVEAU  2 REGLE 20  A L'ENDROIT  1 DONNE L'EXPRESSION 88
* PUI B + -1.0 N * * A B B
 NIVEAU  2 REGLE 20  A L'ENDROIT  7 DONNE L'EXPRESSION 96
 * PUI B + -1.0 N * * B B A

FORMULE VRAIE POUR TOUT N
```

Cet exercice est le dual du précédent.

Au rang 1 : La preuve n'est pas immédiate. Les arborescences G et D sont vite mortes. En effet, le sélecteur de régles ne donne accès qu'aux niveaux 9 et 2 ici. (d'où l'impossibilité de remplacer PUI B 1 par B avant le premier redémarrage).
Au redémarrage, le sélecteur donne accès au niveau 3 (manipulation de PUI) et on peut conclure rapidement.

Au rang n : Le même phénomène se reproduit pour la sélection des règles. Cette fois-ci, le programme utilise l'hypothèse de récurrence et ne donne pas une preuve directe comme il l'avait fait dans l'exercice de même type précédent .

Temps total d'exécution 20 s.

Améliorations souhaitables

1 - Il serait intéressant de doter le programme de la possibilité de manipuler la technique du changement de variables. Cela permettrait de remplacer dans certains cas un terme par une variable et d'alléger un certain nombre d'expressions assez lourdes. L'expression définissant le changement de variable deviendrait règle privilégiée disponible à chaque instant . Son statut pourrait être comparable aux formules définissant les suites numériques dans les problèmes de suites.

2 - Les possibilités de relance vers RECUR d'un certain nombre de problèmes qui n'ont pu être résolus par PROUV mériteraient d'être examinées. Cela nécessiterait de dégager un certain nombre d'heuristiques permettant d'apprécier la complexité d'un problème afin de ne pas faire boucler le programme sur des problèmes de plus en plus compliqués.

3 - Du point de vue normalisation d'expressions, nous avons juste utilisé des formes normales pour les polynômes. Il serait intéressant d'améliorer notre outillage en programmant des algorithmes de normalisation pour des expressions plus riches (contenant des exponentielles, logarithmes et lignes trigonométriques par exemple).

4 - Il serait sans doute possible de perdre moins de temps au moment de la recherche du couple de noeuds les plus proches pour lancer un redémarrage. Pour cela, on pourrait utiliser deux fonctions d'évaluation de distance ; une très simple et rapide permettant d'éliminer les couples les plus mauvais. Ceci permettrait de faire une présélection et c'est seulement sur les couples retenus que serait appliquée l'évaluation sophistiquée, efficace mais lente.

5 - Dans un système de démonstration automatique, nous souhaitons toujours voir le programme se comporter comme un bon mathématicien. Il est en particulier gênant de voir le programme sortir un certain nombre de pas de démonstration que le mathématicien saute parce qu'ils lui paraissent évidents. Dans le programme que nous avons écrit, il serait aisé de ne faire sortir que les étapes importantes du calcul. Il suffirait, pour cela, au moment de la preuve, de ne pas sortir ce qui a été obtenu en utilisant des règles désignées à priori. Par exemple, on pourrait décider de ne pas sortir les pas correspondants à l'application de l'associativité de + ou de * , de $\frac{du}{dx} = 0$ si x ne figure pas dans u etc... Nous aurions ainsi, sur le papier

...

des démonstrations plus courtes, plus proches de celles données par le mathématicien. Cela ne changerait en rien l'établissement du résultat lui-même. Toutes les étapes seraient faites. Mais après tout, le mathématicien aussi les fait ces étapes intermédiaires même s'il ne les écrit pas ! Nous n'avons pas utilisé cette possibilité (simple à mettre en oeuvre), car en cas d'erreur, les étapes effectivement suivies apparaissent moins bien et il est plus difficile de retrouver où le programme a fait fausse route.

CONCLUSION

Le présent travail doit être considéré comme une étape dans un ensemble de recherches encore loin de leur terme. Un certain nombre d'insuffisances du programme ont été signalées. La plupart d'entre elles pourraient aisémment être surmontées lors de la réalisation d'une nouvelle version de ce programme.

Les résultats obtenus par cette première version sont cependant loin d'être négligeables.

- Ils permettent de mettre en évidence la puissance du prouveur qui a été écrit. En particulier , cela souligne l'intérêt et l'efficacité du développement du second membre et de la technique de redémarrage. Cela à permis de montrer expérimentalement qu'il est possible d'allier dans un système de démonstration automatique des techniques combinatoires et des techniques heuristiques. De ce côté là, la méthode peut être considérée comme un acquis.

- Ils permettent d'envisager la réalisation effective de calculs formels dans des domaines complexes mais réels. En ce sens les enseignements tirés de cette réalisation sont riches et peuvent servir de base pour la poursuite des recherches dans ce domaine. Il semble à ce sujet qu'un effort soit encore nécessaire afin de trouver des représentations canoniques pour des classes d'expressions suffisamment larges. L'écriture d'algorithmes rapides de conversions d'expressions sous ces formes canoniques permettraient d'aborder le traitement heuristique avec moins de difficultés.

- Il était également intéressant de dégager des méthodes permettant de travailler dans un espace aussi réduit que possible avec des volumes d'informations aussi large que possible. La technique du redémarrage peut être observée sous cet angle. Elle montre comment à un instant donné, à l'aide d'heuristiques, il est possible de décider de récupérer tout l'espace occupé par une masse d'informations , le problème continuant à se développer sur la même zone que celle qu'il occupait précédemment. Il semble intéressant de continuer à dégager de telles méthodes dans la mesure où nous

...

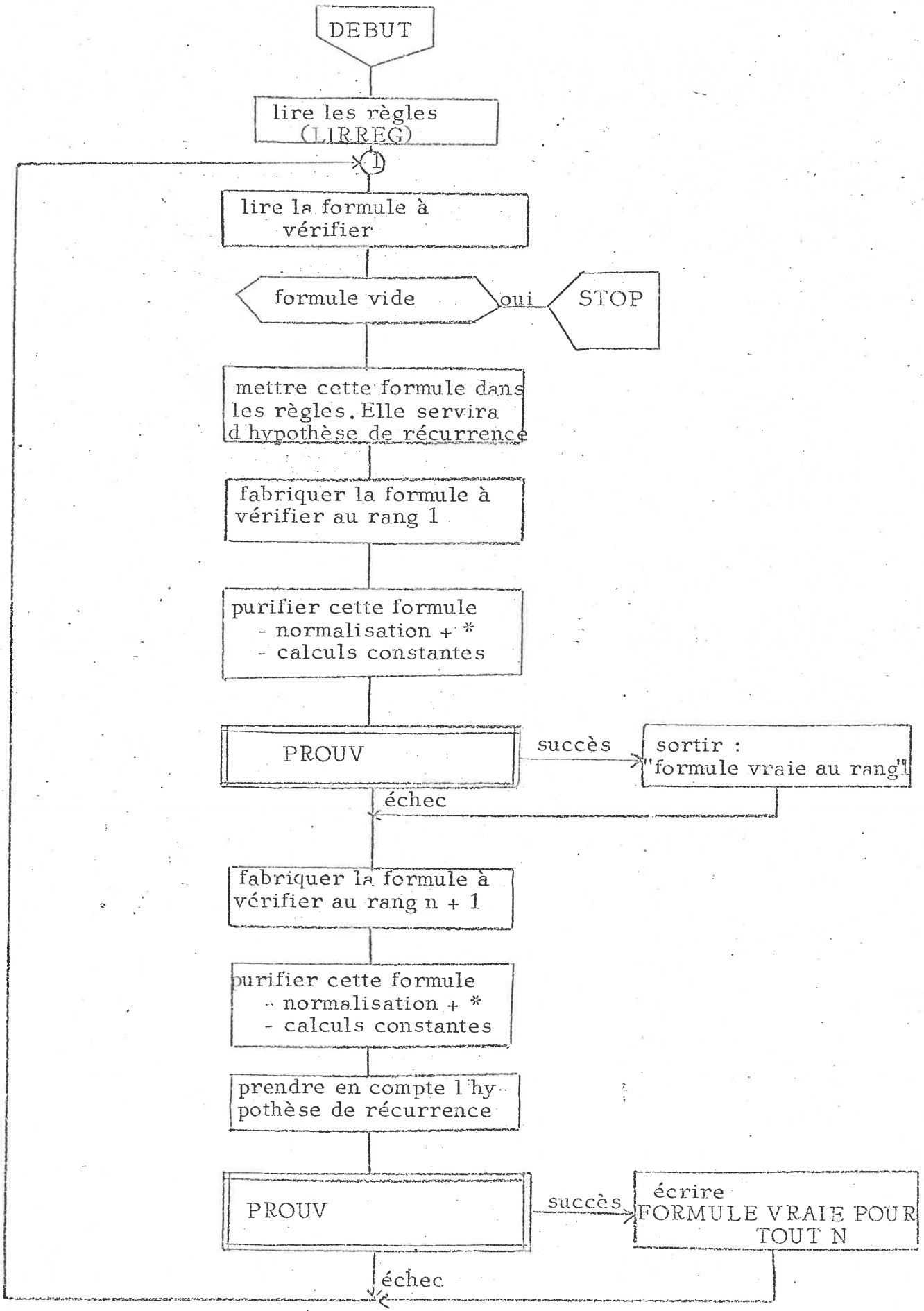
savons que nous serons pour longtemps encore limité par l'espace mémoire.

Enfin, il semble que ce travail fondé sur des expérimentations apporte une contribution à la réflexion sur certains aspects des mathématiques en particulier sur les mécanismes qui nous permettent de conduire un calcul et c'est là qu'il est possible de s'étonner devant cette merveilleuse machine qu'est le cerveau humain.

ANNEXE A

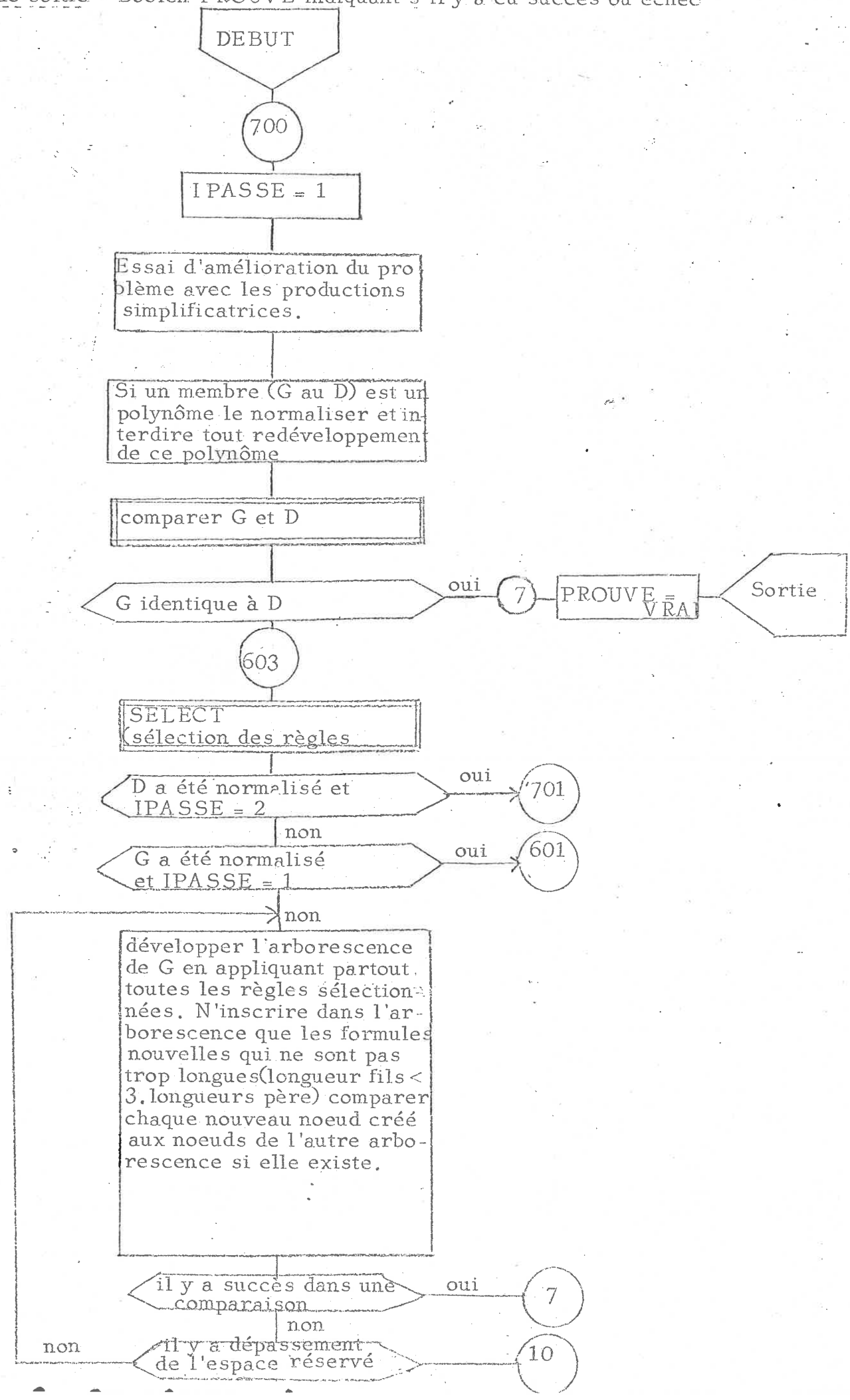
Principaux organigrammes

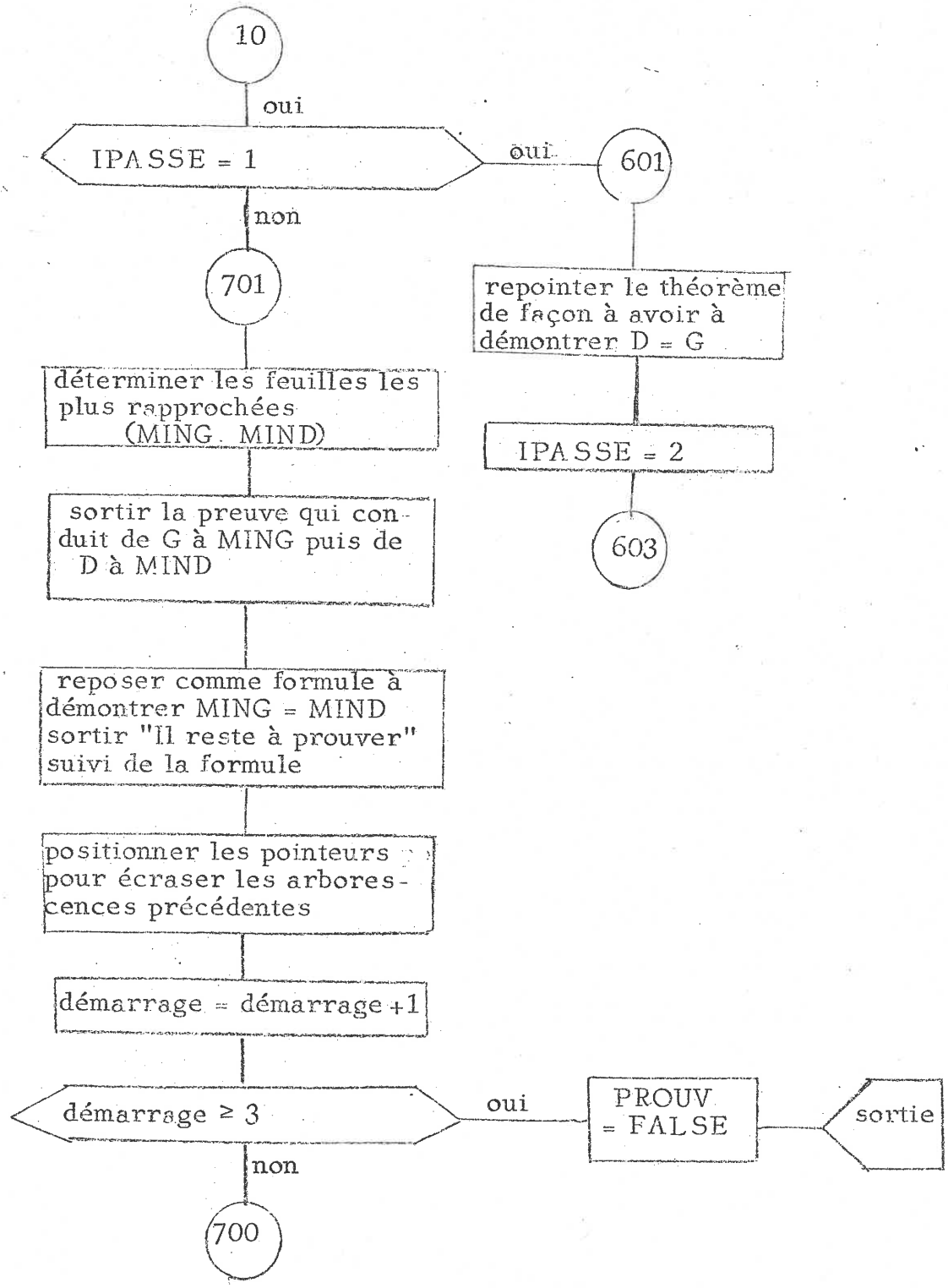
Organigramme général de RECUR



ORGANIGRAMME GENERAL DE PROUV

argument d'entrée - tête du théorème dans la liste
argument de sortie - Boolen PROUVE indiquant s'il y a eu succès ou échec




10
oui
IPASSE = 1
oui
601
non
701
repointer le théorème de façon à avoir à démontrer D = G
déterminer les feuilles les plus rapprochées (MING. MIND)
IPASSE = 2
sortir la preuve qui conduit de G à MING puis de D à MIND
603
reposer comme formule à démontrer MING = MIND sortir "Il reste à prouver" suivi de la formule
positionner les pointeurs pour écraser les arborescences précédentes
démarrage = démarrage +1
démarrage ≥ 3
oui
PROUV = FALSE
sortie
non
700

ANNEXE B

Table des connectives utilisées

UNAIRES

RAC	racines
LOG	logarithme
EXP	exponentielle
SIN	sinus
COS	cosinus
TG	tangente
COT	cotangente
NON	négation
---	- unaire
ALO	ALORS en liaison avec le SI pour le SI ... ALORS des règles conditionnelles

BINAIRES

+ - * /	opérations arithmétiques habituelles
=	égalité (permet de relier les deux membres d'une égalité : règle ou théorème)
PUI	puissance (PUI X Y représentant X^Y)
ET	conjonction
OU	disjonction
IMP	implication logique (IMP P Q pour $p \Rightarrow q$)
EQU	équivalence logique (EQU p q pour $p \Leftrightarrow q$)
SIG	représente Σ
	- le premier terme commence par IND et précise l'indice et le terme sur lequel il agit
	- le second terme commence par DEA et précise le champ de variation de l'indice.
PRD	représente π . idem Σ
DEA	permet d'exprimer le champ de variation d'un indice.
IND	permet de relier un indice au terme indicé (a_i s'écrit IND I A)
	(a été utilisé comme symbole de fonction pour des exercices sur la

...

dérivation. ex : K. $A_{(x)}$ s'écrit * K IND X A)

SI annonce une condition, le premier terme est une condition, le second commence par ALO.

DER symbole indiquant une dérivation. Le premier terme commence par IND et précise les paramètres utiles (ordre de la dérivation et variables de dérivation). Le 2ème terme précise la fonction à dériver.

DAN manipulation d'un terme dans une polonaise. Précise une substitution à faire (ex : DAN T= I U indique que dans le terme T, il faut remplacer la variable I par le terme U). Cette connective a été utilisée également dans des conditions de base pour examiner la présence d'un terme dans un autre (ex : NON DAN X + * 3 X 2 à la valeur faux, DAN X + * 3 X 2 ayant la valeur VRAI , X apparaissant dans + * 3 X 2).

BIBLIOGRAPHIE

I - Généralités

1 - A. ARNOLD

Les mathématiques à la portée de l'ordinateur (Dunod, 1970)

2 - R. BELLMAN et P. BROCK

On the concepts of a problem and problem solving (American Math. Monthly, vol 67, n° 2, fév. 1960)

3 - D.C. COOPER

Theorem proving in computers, Advances in programming and non numerical pomputation (Ed. Fox)

4 - HUNT MARIN-STONE

Experiments in induction (Academic Press, 1966)

5 - E. ROUCHE et Ch. de COMBEROUSSE

Géométrographie dans le Traité de géométrie (constructions de figure de géométrie à l'aide d'opérations élémentaires - mesure de la simplicité d'une construction)

6 -

Le dossier de la cybernétique, utopie ou science de demain dans le monde d'aujourd'hui (Marabout Université)

II - Intelligence Artificielle - manipulation d'heuristiques

1 - W.W. BLEDSOE

Splitting and reduction heuristics in automatic theorem proving (Artificial Intelligence, vol 2, n° 1, 1971)

2 - L.I. HODGSON

An approach to analytic integration using ordered algebraic expressions (Machine Intelligence, tome 2, p. 47)
(D. MICHIE, Ed., Edimbourg - 1968)

3 - R. LECLERCQ

Le raisonnement scientifique et sa mécanisation
(Dunod ed. PARIS 1969)

4 - B. MELTZER

A new look at mathematics and its mechanization (Machine Intelligence 3, p. 63)
(D. MICHIE Ed., Edimbourg - 1968)

5 - B. MELTZER

Artificial Intelligence and heuristic programming
(Findler Ed., University Press. 1970)

6 - MINSKY

. Some methods of artificial intelligence and heuristic programming
(Mechanization of thought processes, vol. 1, p. 3)
. Heuristic aspects of the artificial intelligence problem
(MIT Group Report 34-55, Déc. 56)
. Steps toward artificial intelligence (Computers and thought
Edited b Fergenbaum and Feldman (Mac GRAW HILL, 1963, p. 406)

7 - J. PITRAT

Intelligence artificielle et méthodes heuristiques, Revue Française de recherche opérationnelle n° 39 (Dunod Ed. PARIS, 1966)

8 - G. POLYA

. Comment poser et résoudre un problème (Dunod, 1957)
. Les mathématiques et le raisonnement "plausible" (Gauthier-Villars, 1958)

9 - E. SANDEWALL

Heuristic search : concepts and methods

10 - J.R. SLAGLE

Artificial Intelligence : the heuristic programming approach (Mac GRAW HILL Ed. 1971)

11 - WALDINGER

Toward Automatic Program Synthesis (C. ACM, vol 14, nb 3, march 1971)

12 - D.A. WATERMAN

Generalization learning techniques for automating the learning of heuristics (Artificial Intelligence 1, 1970, p. 121-170)

III - Généralités sur la notion de démonstration

1 - HADAMARD

Essai sur la psychologie de l'invention dans le domaine mathématique (Librairie scientifique A. Blanchard 1959)

2 - LECLERQ

La création scientifique - Complément au guide théorique et pratique de la recherche expérimentale (Gauthier-Villars 1959)

3 - L. MULLER

Recherches sur la compréhension des règles algébriques chez l'enfant. (Ed. Delachaux Niestlé S.A. Neuchatel PARIS 1956)

4 - POLYA

Comment poser et résoudre un problème (Dunod, 1957)

5 - POLYA

Les mathématiques et le raisonnement "plausible" (Gauthier Villars, 1958)

6 - VAN DER WAERDEN

La démonstration dans les sciences exactes de l'antiquité (Bul. de la Soc. Math. de Belgique, Tome VIII, fasc. 1, 1956)

IV - Programme de démonstration de théorèmes ou de résolution de problèmes

1 - BLEDSOE-BOYER-HENNEMANN

Computer proof of limit theorems (Revue Artificial Intelligence vol. 3 n° 1, 1972)

2 - CHANG

The unit proof and the input proof in theorem proving (J. ACM, 17-4-1970, p. 698-707)

3 - DELHAYE J.L.,

DATAL, un programme de démonstration automatique de théorèmes (en algol) Thèse 3ème cycle (PARIS 1970)

4 - FIKES

REF-ARF. A system for solving problems stated as procedures (Artificial Intelligence 1 - 1970 p. 27-120)

5 - GELERNTER

Intelligent behavior in problem solving machines

6 - GELERNTER

Realization of a geometry-theorem proving machine (Computers and thought p. 63 - Mac GRAW HILL Ed., 1963)

7 - GILMORE P.C.

A examination of the geometry-theorem machine (Revue Artificial Intelligence, vol. 1, n° 3, 1970)

8 - LAURENT J.P.

DALI, un programme qui calcule des limites en levant les indéterminations par des procédés heuristiques. Thèse 3ème cycle (PARIS 1972)

9 - A. NEWELL, H.A. SIMON

GPS, a program that simulates human thought (Computers and thought p. 279, Mac GRAW HILL Ed., 1963)

10 - PITRAT J.

Réalisation de programmes de démonstration de théorèmes utilisant des méthodes heuristiques. Thèse (PARIS, mai 1966)

11 - PITRAT J

Un programme de démonstration de théorèmes (monographie d'informatique AFCET - Dunod Ed. 1970)

12 - QUINLAN-HUNT

A formal deductive problem solving system (J. ACM, vol. 15, n° 4, oct. 1968, p. 625-646)

13 - J.R. SLAGLE, P. BURSKY

Experiments with a multi purpose theorem proving heuristic program (J. ACM. vol. 15, n° 1, 1968)

14 - J.R. SLAGLE

Automatic theorem proving with built in theories including aquality, partial ordering and sets (J. ACM, vol. 19, n° 1, 1972)

15 - J.R. SLAGLE

A heuristic program that solves symbolic integration problems in freshman calculus (Computers and thought, p. 191, Mac GRAW HILL Ed., 1963)

16 - L. SIKLOSSY, V. MARINOV

Heuristic search versus. Exhanstive search (2e IJCAI, Advance Papers, 1971)

17 - L. SIKLOSSY, A. RIEH, MARINOV

Breadth first search. Some surprising results (A.I., vol. 4 n° 1, 1973)

V - Problèmes de représentation - manipulation algébrique

1 - AMAREL S.

On representations of problems of reasoning about actions (Machine intelligence 3, p. 131-171, 1968)

2 - W.S. BROWN

Rational exponential expressions and a conjecture concerning π and e

3 - CAVINESS

On canonical forms and simplification (J. ACM, vol. 17, n° 2, 1970)

4 - COLLINS

A system for polynomial manipulatior (C. ACM, vol. 9, n° 8, 1966)

5 - ERNST, NEWELL

Some issues of representation in a general problem solver Proceeding of the spring joint computer conference (1967)

6 - GROSS-LENTIN

Notions sur les grammaires formelles (Gauthier-Villars)

7 - W.A. MARTIN

Determining the equivalence of algebraic expressions by hash coding

8 - MOSES

Algebraic simplification. A guide for the perplexe' (C. ACM, vol. 14, n° 8, 1971)

9 - POHL

Heuristic search viewed as path findingin a grap' (Artificial Intelligence 1, 3, p. 193-204, 1970)

10 - ROSENBLOOM

The elements of mathematical logic (Dover Publications)

11 - G. SALTON

Manipulation of trees in information retrieval (C. ACM, vol. 5, n° 2, p. 103-114, 1962)

VI - Récurrence

1 - R.M. BURSTALL

Proving properties of programs by structural induction (Computing Journal 12, 1, 1969, p. 41-48)

2 - J.L. DARLINGTON

Automatic theorem proving with equality, substitution and mathematical induction (Machine Intelligence, 3, Edimbourg University Press, p. 113-123, 1968)

3 - L. HENKIN

On mathematical induction (American Math. Monthly, vol. 67, n° 4, avril 1960)

4 - KING and FLOYD

Interpretation oriented theorem prover over integers (Second. ann. ACM symp. on theory of computing Northampton Mass., 1970)

5 - MANNA-WALDINGER

Toward automatic program synthesis (C. ACM, vol. 14, n° 3, March 1971)

6 - PARK

Fix point induction and proof of program properties (MI 5, p. 59-78)