Système de déduction automatique : application à la construction de programmes

Farid Ouabdesselam

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR INGENIEUR
«Informatique»

par

OUABDESSELAM Farid

SYSTEME DE DEDUCTION AUTOMATIQUE.

APPLICATION A LA CONSTRUCTION DE PROGRAMMES.

Thèse soutenue le 15 Décembre 1980 devant la commission d'examen.

L. BOLLIET	Président
A. COLMERAUER	
B. LEVRAT	Examinateurs
G. VEILLON	
G. RUGGIU	Rapporteur

Je remercie tout particulièrement

Monsieur Louis BOLLIET, Professeur à l'Institut Universitaire de Technologie de Grenoble, à qui je témoigne toute ma gratitude et ma profonde reconnaissance pour m'avoir accueilli dans son équipe, conseillé et vivement soutenu par ses encouragements constants ;

Monsieur Gérard VEILLON, Professeur à l'Institut National Polytechnique de Grenoble, qui a dirigé mes premiers travaux en Intelligence Artificielle et dont la bienveillance m'a permis d'achever ce travail ;

Monsieur Gilles RUGGIU, Chef du Groupe de Recherches en Informatique au Laboratoire Central de Recherches de Thomson-CSF, qui a bien voulu donner son avis de spécialiste ;

Monsieur Alain COLMERAUER, Professeur à l'Université d'Aix-Marseille, dont j'ai reçu les premiers conseils pour la réalisation d'un précédent système de déduction et qui a accepté de juger ce travail ;

Monsieur Bernard LEVRAT, Doyen de la Faculté des Sciences de l'Université de Genève, qui a consenti à examiner mon travail.

Mes remerciements s'adressent également aux deux autres membres de l'équipe Validation de Programmes, Michel CAPLAIN qui la dirige et Bernard AMY, pour leurs critiques à la fois constructives et amicales.

Je remercie enfin Madame CHALAND pour sa patience et la part active qu'elle a pris dans la réalisation matérielle de ce document.

RESUME

L'étude des moyens de déduction nécessaires pour entreprendre une construction assistée de programmes mène à la définition d'un système de déduction automatique capable
- de traiter l'égalité d'une façon générale ;
- d'entreprendre la preuve de formules qui ne sont pas des théorèmes, et d'étudier les causes d'échec ;
- d'effectuer un tri pour retirer d'un large ensemble d'informations celles utiles à la preuve.

Ces caractères ont pu être pris en compte dans le cadre d'un système général, de genre « déduction naturelle ».

Certaines classes d'expressions reçoivent toutefois un traitement particulier. Ainsi une méthode de codage permet de déterminer, sans simplification ni mise sous forme normale, l'équivalence d'expressions numériques.

SOMMAIRE pages

INTRODUCTION

Deux aspects de la production de logiciel fiable nous intéressent ici : la vérification de programmes et la construction de programmes.

• Vérification de programmes

L'emploi de la méthode de Floyd - Naur [F3] permet par l'introduction d'assertions dans un programme de vérifier que l'effet du programme est bien conforme à l'intention du programmeur (l'intention est précisément représentée par les assertions).

Instructions et assertions sont composés pour produire les conditions de vérification dont la preuve assure la vérification du programme. Ces conditions ont la forme suivante

$$\varphi \supset \psi$$

où φ et ψ sont des formules atomiques ou composées à partir de $\wedge$, $\vee$, $\equiv$ et $\supset$; elles sont exprimées en CP_1.

L'exemple suivant nous sert à préciser la nature des conditions. Le programme calcule a^b.

DEBUT

A1 | AFFIRMER $X = A \wedge Y = B \wedge B \geq 0$ |

$Z := 1;$

TANTQUE $Y \neq 0$ FAIRE

A2 | AFFIRMER $Y \geq 0 \wedge Z * (X \uparrow Y) = A \uparrow B$ |

DEBUT

SI $Y \text{ MOD } 2 = 1$ ALORS

DEBUT

$Z := Z * X;$

FIN

$Y := Y \text{ DIV } 2;$

$X := X * X;$

FIN

A3 | AFFIRMER $Z = A \uparrow B$ |

$X = A \wedge Y = B \wedge B \geq 0$

$\supset \quad Y \geq 0 \wedge 1 * (X \uparrow Y) = A \uparrow B$

Pour démontrer $1 * (X \uparrow B) = A \uparrow B$, le système doit posséder les propriétés de l'opérateur $*$ pour simplifier $1 * (X \uparrow Y)$ en $X \uparrow Y$. Il suffit alors de reconnaître l'identité $A \uparrow B = A \uparrow B$. Cette démonstration peut facilement être obtenue avec les deux modes de déduction et se présente de manière naturelle pour un utilisateur.

En supposant que le système est capable de réaliser cette preuve, il doit faire usage des propriétés de réflexivité et de remplacement de l'égalité, quelle que soit la manière dont il procède : qu'il substitue à X et à Y les valeurs fournies par les égalités en hypothèse $X = A$ et $Y = B$, puis qu'il reconnaisse l'identité $A \uparrow B = A \uparrow B$, ou bien qu'il déduise de la formule à prouver $X \uparrow Y = A \uparrow B$ qu'il lui faut établir $X = A$ et $Y = B$ en recherchant des hypothèses du type $X = \ldots$ et $Y = \ldots$

Pour la vérification de programmes de ce genre il faut donc disposer des propriétés de l'égalité et de certains opérateurs.

Un autre programme numérique, réalisant la division de A par B, nous sert à préciser d'autres besoins.

```
DEBUT
Q := 0 ;
R := A ;
Y := B ;
S := 1 ;
TANT QUE  Y < A  FAIRE
DEBUT
Y := Y * 2 ;
S := S * 2 ;
FIN ;
TANT QUE  R > B  FAIRE
A1 | AFFIRMER   A = B * Q + R ; |
SI  Y ≤ R  ALORS
    DEBUT
    R := R - Y ;
    Q := Q + S ;
    FIN ;
Y := Y MOD 2 ;
S := S MOD 2 ;
FIN ;
FIN.
```

La condition de vérification pour le chemin A1 - A1 s'écrit :

$$A = B * Q + R$$
$$\wedge\ Y \leqslant R$$
$$\wedge\ (R - Y) > B$$
$$\Rightarrow\ A = B * (Q + S) + (R - Y)$$

La preuve n'est pas possible sans la connaissance des propriétés de l'égalité et la propriété de distributivité de * par rapport à +. Mais en fait on s'aperçoit que la condition de substitution n'est pas un théorème. Le programmeur a en effet écrit une assertion trop faible. Il faut y ajouter $Y = B * S$, puisque l'assertion est un invariant de boucle et que toutes les variables de la boucle doivent y figurer.

Les cas d'oubli semblables sont en fait très fréquents car il est impensable que dès leur première écriture les assertions sont complètes et sans erreurs.

Les preuves entièrement automatiques sont ici encore concevables mais tout échec doit être accompagné de renseignements sur ses causes pour que l'utilisateur ait les moyens de déceler rapidement ses erreurs.

Si l'on envisage de pouvoir fournir à l'utilisateur des renseignements sur le déroulement de la preuve, il faut

alors éviter de réécrire systématiquement certains opérateurs comme MOD en fonction d'autres opérateurs comme +, ÷, etc.

Le système de déduction doit pouvoir se rendre compte par lui-même s'il lui faut recourir à la définition d'un opérateur ou bien s'il peut se contenter de ses propriétés.

• Construction de programmes

La définition d'un système qui ne se contenterait pas de vérifier des programmes déjà écrits mais permettrait plutôt à l'utilisateur de mêler écriture et vérification de programmes donne la possibilité de faire des preuves de plus haut niveau [F13] [F8].

Jusqu'à présent, l'expression des conditions de vérification a fait intervenir les opérateurs du langage de programmation, les opérateurs logiques et les quantificateurs. Le langage d'assertion dans ces circonstances peut donc être le calcul des prédicats du premier ordre avec des symboles prédéfinis ; il est suffisant alors d'intégrer les propriétés des opérateurs dans la démonstration pour être sûr d'avoir un système suffisamment performant.

Suzuki a montré [F16] que ce choix n'était pas

valable car certaines propriétés des programmes étaient inexprimables en calcul des prédicats du premier ordre avec un nombre de symboles fini.

Nous reprenons ses arguments :

Supposons qu'un programme S trie un tableau A (qui lui est fourni en entrée) et restitue le résultat dans un tableau B. L'assertion finale va mettre en relation les propriétés des éléments de B et de ceux de A : on exprime d'abord que B est trié (dans le sens croissant)

$$\forall i \left((1 \leqslant i \leqslant n-1) \supset B(i) \leqslant B(i+1) \right)$$

et ensuite que B est constitué par tous les éléments de A et eux seuls, c'est à dire qu'il y a bijection entre A et B :

$$\begin{aligned} \exists bij \quad & \forall i \ (1 \leqslant i \leqslant n) \supset (1 \leqslant bij(i) \leqslant n) \\ & \wedge \forall ij \ (1 \leqslant i \leqslant j \leqslant n) \supset (bij(i) \neq bij(j)) \\ & \wedge \forall i \ (1 \leqslant i \leqslant n) \supset A(i) = B(bij(j)) \end{aligned}$$

Or cette dernière formule n'est plus une formule du calcul des prédicats du premier ordre.

Pour éviter d'amener l'usage de démonstrateurs de théorèmes du deuxième ordre (domaine encore très peu exploré mais dont on connaît les difficultés [B4] [B12]) Suzuki propose d'étendre le langage d'assertions en permettant la définition de nouveaux symboles, et suggère de profiter au mieux de cette extension en introduisant des symboles de haut niveau traduisan

en un seul prédicat des propriétés qui demandent une phase complexe. Ainsi l'assertion finale précédente s'écrit simplement

$$\text{Ordonné}(B, 1, n) \wedge \text{Permutation}(B, A)$$

Ces nouveaux symboles doivent évidemment être définis, et leurs propriétés énoncées en spécifiant les types des objets auxquels elles sont applicables.

Dans cet exemple on a

$$\forall X \left(\forall x \left((i \leqslant x \leqslant j) \supset (X(x) \leqslant X(x+1))\right) \equiv \text{Ordonné}(X, i, j)\right)$$

X étant un tableau d'entiers, c'est à dire une application de $[1, n]$ dans $\mathbb{N}$.

La définition de Ordonné fait apparaître la nécessité de pouvoir prendre en compte les quantificateurs.

Elle soulève par ailleurs un problème particulier relatif à l'emploi de l'algorithme d'unification standard au coeur des démonstrateurs de type résolution. Supposons que l'on veuille montrer que $A(4) \leqslant A(5)$. La partie conclusion $X(x) \leqslant X(x+1)$ ne peut être retenue pour cette preuve car cet algorithme ne peut pas rendre $4+1$ et 5 syntaxiquement identiques. Il faudrait auparavant réécrire 5 en $4+1$.

Ceci est sans doute une grande faiblesse des

systèmes de démonstration qu'il faudrait supprimer si l'on voulait obtenir des preuves plus naturelles.

L'expression de la règle Ordonné nous fait découvrir par la formule $i \leqslant x \leqslant j$ que beaucoup de preuves consisteront à montrer l'existence de certaines propriétés sur des domaines restreints, que nous appelons <u>domaines de validité</u>. Ces preuves sont assez spécifiques pour avoir reçu un traitement particulier nommé <u>preuves par cas</u> [D 1]. Nous allons essayer tout de même de les faire entrer dans un cadre très général.

La multiplication des règles du type de Ordonné amène les auteurs de systèmes de construction de programmes [F 17] à envisager des méthodes particulières pour que le système puisse lui-même retrouver les informations importantes pour une preuve. L'expérience des bases de données est dans ce cas intéressante [D 12].

La construction de programmes telle que nous venons de la voir est plutôt une construction assistée [F12] [F13] [F17] dans laquelle l'utilisateur intervient.

Mais il est également possible de créer automatiquement des programmes à partir de leurs spécifications [F6] [F15]. Ces spécifications s'écrivent en général

$$\forall x \quad I(x) \Rightarrow \exists z \quad O(x,z)$$

où z est le résultat désiré, I et O représentant les conditions d'entrée et de sortie du programme à dériver. En calcul des prédicats du premier ordre, dans le cas des systèmes de type résolution, il existe un moyen de construire le résultat : c'est le prédicat-réponse de Green [D7]. Dans des systèmes de déduction naturelle ce moyen reste à trouver.

A travers quelques exemples, nous venons de voir les caractéristiques indispensables pour un système de déduction que l'on voudrait utiliser à la construction de programmes. La liste de ces caractéristiques, en plus de celle liée à l'emploi du calcul des prédicats du premier ordre est la suivante :

prise en compte de l'égalité,

possibilité d'entreprendre la preuve de formules qui ne sont pas des théorèmes,

étude des causes de l'échec d'une preuve,

tri rapide des informations utiles à une preuve.

SYSTEME
DE DEDUCTION

La présentation du système de déduction est faite sans préjuger de la représentation interne des formules. Une hypothèse cependant est importante : tous les objets figurant dans les formules, et en particulier les termes, ont un type. La définition du type sera donnée plus loin. Il suffit de savoir que ce type traduit l'appartenance d'un objet à un certain domaine.

1.1 Mise en forme du théorème à prouver.

La formule représentant le théorème à prouver peut être présentée au système sous une forme quelconque avec des quantificateurs placés là où l'utilisateur l'a préféré.

Elle subit à l'entrée un petit traitement qui concerne les opérateurs logiques $\equiv$ et $\neg$, ainsi que les quantificateurs. (Nous verrons plus loin ce second point).

Les théorèmes sont donc récécrits selon les règles suivantes

$$A \equiv B \longrightarrow (A \supset B) \wedge (B \supset A)$$

$$\neg(A \vee B) \longrightarrow \neg A \wedge \neg B$$

$$\neg(A \wedge B) \longrightarrow \neg A \vee \neg B$$

$$\neg\neg A \longrightarrow A$$

La première règle est rendue nécissaire par le traitement des quantificateurs ne portant que sur A ou B qui prennent une signification différente selon que l'on utilise l'une ou l'autre des implications.

Une fois les transformations ci-dessus réalisées, les opérateurs logiques ne sont jamais modifiés, et pendant la preuve les formules ne sont pas interpretées différemment.

Ainsi $\neg A \vee \neg B$ n'est jamais vu comme $A \supset B$, de même que $A \supset B$ n'est jamais considéré comme étant $\neg B \supset \neg A$.

1.2 Recherche d'une preuve.

Toute preuve est effectué particulièrement dans chacun des deux modes dérivation et déduction, à partir d'une base de formules qui regroupe les hypothèses, c'est à dire la partie antécédent de la formule à prouver et les axiomes, définitions et théorèmes déjà prouvés qui représentent la « connaissance » du système.

Mode dérivation.

Les formules du genre $A \vee B$ occasionnent un fonc-tionnement particulier du système. En effet, une simulation du raisonnement humain est réalisée ; elle consiste à supposer l'une des formules de la disjonction et rechercher une preuve avec cette hypothèse, puis à effectuer le même travail avec l'autre formule.

Mode déduction.

Le système procède par décomposition des buts en sous-buts en appliquant les règles :

But	Sous-but
$\Gamma, A \supset A$	succès
$\Gamma \supset (A \supset B)$	$\Gamma, A \supset B$
$\Gamma \supset (A \wedge B)$	$\Gamma' \supset A$ et $\Gamma'' \supset B$ avec au départ $\Gamma' = \Gamma'' = \Gamma$
$\Gamma \supset A$	Echec
Γ ne contenant aucune occurrence de A en partie conclusion d'une implication	
$\Gamma \supset A$	$\Gamma, \neg A \supset A$
Il s'agit du but initial qui est tout entier un échec.	
$\Gamma \supset A \vee B$	$\Gamma \supset A$ ou $\Gamma \supset B$
$(\Gamma \supset A)$ ou Echec	$\Gamma \supset A$
$(\Gamma \supset A)$ et Echec	Echec

N.B. Aucune priorité n'existe entre les buts liés par « ou » et par « et ». Γ représente l'état de la base au moment où le but est pris en considération, ce but ne pouvant être qu'un littéral.

Au vu du traitement des disjonctions, dans la base et dans le graphe des buts, il apparaît que l'un et l'autre sont développés par contexte.

L'échec d'un but provoque toujours le retour au but précédent. En plus du cas prévu ci-dessus, l'échec est également prononcé lorsqu'un but est identique à l'un de ses prédécesseurs. On évite ainsi les cycles dans le graphe.

1.3 Méthode de déduction.

C'est elle qui assure la production des sous-buts et des nouvelles hypothèses. Elle regroupe plusieurs règles de déduction.

Mode réduction.

R1 $$\frac{\Gamma, A \supset B \vdash A \vee (\neg A \supset B)}{\Gamma, A \supset B \vdash B}$$

La règle R1 doit être lue de la manière suivante: Pour déduire B, sachant que $A \supset B$, il faut et il suffit de déduire $A \vee (\neg A \supset B)$.

Il convient de remarquer que cette règle exprime

une condition nécessaire et suffisante pour la preuve d'un but, contrairement à la plupart des autres méthodes conçues pour ce mode [B1] [B2] [C8] [C10] qui, n'employant que le modus ponens, n'expriment qu'une condition suffisante.

Mode dérivation.

R2 $$\frac{\Gamma \vdash A \text{ et } \Gamma \vdash A \supset B}{\Gamma \vdash B}$$

R3 $$\frac{\Gamma \vdash A \text{ et } \Gamma \vdash \neg A}{\Gamma \vdash B}$$

N.B. R2 est la règle de modus ponens bien connue.
R3 ne sert qu'aux preuves obtenues par contradiction.

L'utilisation des deux modes est absolument indispensable. En mode réduction, les formules de la base ne peuvent être sélectionnées pour une déduction que sur leur partie conclusion. Or, certaines preuves sont impossibles dans ce seul mode à moi d'inverser le sens d'une implication, ou bien parce que les formules de la base ont été exprimées avec la conjonction plutôt que l'implication. En voici deux exemples:

Exemple 1 :

Déduire B de A et $\neg B \supset \neg A$.

Dans aucun des deux modes, au départ, nous ne pouvons effectuer de déduction.
Le but initial B étant un échec provoque l'ajout à la base de sa négation $\neg B$ (voir 2.2). En mode dérivation on obtient alors $\neg A$, et la preuve définitive est acquise par contradiction.

Exemple 2 :

La négation dans les buts n'est l'objet d'aucun traitement particulier, l'information se conservant telle qu'elle parvient au système.

Si la conclusion du théorème à prouver est niée, on considère que la négation du même prédicat existe vraisemblablement parmi les hypothèses. Si ce n'est pas le cas, la preuve sera alors obtenue par contradiction sur la base. Nous illustrons ce fonctionnement en prouvant la règle d'introduction de $\neg$ des systèmes de déduction naturelle :

$$\big((A \supset B) \wedge (A \supset \neg B) \big) \supset \neg A .$$

Comme dans l'exemple précédent, aucune déduction n'est faite, quelque soit le mode dans lequel la preuve est commencée. En revanche l'addition de A (négation de $\neg A$)

à la base provoque en mode déclivation la déduction de B puis celle de $\neg B$, hypothèses contradictoires.

1.4 Prise en compte des termes.

Le principe de la méthode de déduction a été présenté en utilisant la notation du calcul propositionnel. Le passage au calcul des prédicats du premier ordre se fait par la création de conditions de substitution pour les arguments correspondants, dans les deux prédicats, sur la base du théorème de remplacement de l'égalité :

$$\text{Si } \Gamma \vdash r = s \text{ alors } \Gamma \vdash C_r \equiv C_s$$

r, s étant des termes, r partie constituante de la formule C_r, C_s résultat du remplacement de r par s.

Ainsi les règles R1, R2 et R3 s'écrivent maintenant:

$$\text{R1} \quad \frac{\Gamma, A \supset B(s) \vdash (s = t \wedge A) \vee (s = t \wedge (\neg A \supset B(t)))}{\Gamma, A \supset B(s) \vdash B(t)}$$

N.B. A et B sont des littéraux.

R_2 $$\frac{\Gamma \vdash A(r) \text{ et } \Gamma \vdash A(s) \supset B \text{ et } \Gamma \vdash r = s}{\Gamma \vdash B}$$

R_3 $$\frac{\Gamma \vdash A(s) \text{ et } \Gamma \vdash A(t) \text{ et } \Gamma \vdash r = s}{\Gamma \vdash B}$$

Leur application est soumise à la vérification préalable de l'identité des types des arguments.

Les égalités représentant les conditions de substitution doivent elles-même être démontrées. Elles peuvent être précédées de quantificateurs selon les variables qui apparaissent dans s et t.

Le théorème de remplacement ci-dessus n'étant applicable que sous réserve que les variables à substituer ne sont pas quantifiées, et les variables de la partie conclusion étant considérées comme constantes pendant la déduction, les égalités à démontrer consistent à mettre en évidence l'existence de substitutions pour les variables du terme s. On peut donc considérer que ces variables sont précédés de quantificateurs existentiels.

Soit l'hypothèse $A(x) \wedge C(y) \supset B(x+y)$ et le but $B(a)$; le sous-but suivant est

$$x + y = a \wedge A(x) \wedge C(y)$$

et doit être lu

$$\exists\, x\, y \quad x + y = a \wedge A(x) \wedge C(y) .$$

1.5 Traitement de l'égalité.

L'introduction de conditions de substitution construites à l'aide de l'égalité nous amène à considérer l'égalité comme un symbole de base connu du système.

Les propriétés de symétrie et de réflexivité sont directement construites dans la procédure de recherche des substitutions.

La déduction d'une égalité se fait à l'aide de la règle de base R1 (l'égalité est alors traitée comme tout autre prédicat), et d'une règle de déduction supplémentaire

R4
$$\frac{\Gamma \vdash x = y}{\Gamma \vdash f(x) = f(y)}$$

utilisable uniquement en mode réduction. Elle traduit la propriété de remplacement d'un terme par un autre dans un terme construit à l'aide de symboles fonctionnels. Comme les propriétés précédentes de l'égalité, cette propriété est prise

en compte par la procédure de recherche des substitutions.

1.6 Plans et stratégies.

1.6.1 Plans

Avant de commencer toute déduction le système procède à l'établissement d'un ou de plusieurs plans qui servent à diriger la déduction.

Ces plans sont fort différents, selon qu'il s'agit d'une égalité ou d'un autre prédicat.

En nous plaçant dans le cas le plus général, c'est à dire la preuve d'une formule ne portant pas sur une égalité, nous appelons plan global le plan dessiné pour trouver une preuve de cette formule, et plan local le plan développé pour prouver la condition de substitution lors de chaque utilisation d'une règle de déduction.

a) Plan global

Il consiste avant tout à déterminer les formules susceptibles d'être utiles parmi celles que le système connaît.

Il mène alors une simple recherche à partir des prédicats apparaissant dans la conclusion et les hypothèses, à l'aide des formules de type implicatif et sans engendrer aucune condition de substitution.

Dans chaque sens des chemins sont établis comme des suites de formules. Ne sont retenus à la fin que les chemins ayant une intersection. On est ainsi assuré que la preuve, si elle est possible, passera par l'un de ces chemins. Les longueurs des chemins en nombre de formules sont également conservées.

Exemple

Soit la formule à démontrer

$$\forall x \quad A(0) \wedge B(x) \Rightarrow C(x)$$

Dans la base sont présents les axiomes suivants.

(A1) $\forall x \quad U(x) \Rightarrow \exists y \; B(y) \wedge D(y, x)$

(A2) $\forall x \quad R(x) \Rightarrow A(x)$

(A3) $\forall x \; \exists y \; A(y) \wedge P(x, y) \Rightarrow Q(xy)$

(A4) $\forall x \; (\exists y \; Q(x, y) \Rightarrow C(x))$

Sont retenus pour le plan les axiomes A3 et A4 ; mais la preuve sera impossible.

b) Plan local

La preuve d'une égalité représentant une condition de substitution fait essentiellement appel à d'autres égalités.

Un plan semblable à celui qui vient d'être exposé n'a dans ce cas plus de sens.

Le plan local a pour rôle d'estimer la suite et la nature des transformations à réaliser afin de rendre les deux membres identiques. Les transformations se réduisent à la suppression S, l'insertion I et le remplacement R d'un opérateur, d'une constante ou d'une variable. Elles constituent un bon aperçu des caractéristiques expansion ou contraction des deux règles de réécriture sous lesquelles on peut lire l'égalité dans les deux sens droite-gauche, gauche-droite.

Chaque égalité-hypothèse est également étudiée de cette manière : on lui associe donc des suites de transformations.

Le plan est réalisé par décomposition de la suite des transformations initiales en plusieurs suites correspondant aux hypothèses. On conserve alors ces égalités hypothèses. Elles représentant le plan dont le nombre de pas est défini par le nombre de formules sélectionnées.

Exemple

Soit à prouver

$(x.e = x \wedge x.x = e) \Rightarrow a.b.b = a$

Pour chaque égalité sont construites deux suites de transfor.-mations sur la base de la représentation interne (préfixée).

H1 $x.e = x$ $\begin{cases} \overrightarrow{H1} : S(.)\ S(e) \\ \overleftarrow{H1} : I(.)\ I(e) \end{cases}$

H2 $x.x = e$ $\begin{cases} \overrightarrow{H2} : S(.)\ S(x)\ S(x)\ I(e) \\ \overleftarrow{H2} : I(.)\ I(x)\ I(x)\ S(e) \end{cases}$

B1 $a.b.b = a$ $\quad \overrightarrow{B1} : S(.)\ S(.)\ S(b)\ S(b)$

Le plan est établi en recherchant pour les trans.-formations du but des transformations équivalentes dans les hypothèses. Si l'emploi de certaines hypothèses entraîne des transformations non présentes dans la suite de celles du but il faut rechercher de nouvelles hypothèses permettant de les éliminer (transformations opposées).

Partant de $S(.)\ S(.)\ S(b)\ S(b)$

on sélectionne $\overrightarrow{H1}$ et la suite des transformations devient

$S(.)\ S(b)\ S(b)\ S(e)$

Par $\overrightarrow{H2}$ on réussit à supprimer S(e), et les transformations restantes sont identiques aux transformations introduites après instanciation de x par b. Le plan est achevé.
Le plan suggère donc d'employer H1 puis H2.

N.B. - Etant donné que l'on effectue les preuves par égalité, on travaille toujours au niveau le plus haut, et par conséquent on recherche une transformation opposée en bout de chaîne.

- La liste des substitutions est établie sans tenir compte des positions respectives du même symbole dans les deux membres. Ainsi

$$0 = x.0 \quad \text{ou} \quad 0 = 0.x$$

ont même suite de transformation

I (·) I (x) dans un sens, et S(·) S(x) dans l'autre.

Lorsque la preuve porte sur des égalités qui ne sont pas des conditions de substitution, il faut déterminer leur nature en fonction de leurs membres, afin d'établir s'il s'agit d'une égalité au sens identité ou d'une égalité sur un certain domaine pour laquelle il existe une définition.

Ainsi la preuve de ab = ba dans le cadre des

propriétés d'un groupe correspond à la preuve d'une égalité-identité. En revanche celle de A = B où A et B sont des ensembles devra éventuellement recourir à la définition de l'égalité entre deux ensembles.

Dans le premier cas le plan correspondra exactement au plan local décrit ci-dessus ; dans le second cas on développera plutôt un plan de type global.

1.6.2 Stratégies

Le système dispose de deux stratégies, une globale et une locale comme pour les plans; il convient de déterminer également le genre du théorème à prouver, selon qu'il porte sur une égalité ou non.

Les stratégies agissent sous le contrôle des plans Elles permettent de choisir un chemin parmi tous les chemins possibles à un certain stade de développement de la preuve. L'objectif consiste à sélectionner le chemin le plus court.

a) Stratégie globale

Le système de déduction opérant dans les deux

modes réduction et dérivation, deux espaces de recherche sont développés. Il faut éviter que chacun des processus d'expansion ne soit fait sans aucun lien avec son opposé. La situation la plus insolite serait le développement de deux preuves sur deux chemins différents.

La stratégie trouve là son premier rôle : conduire le développement des deux espaces de recherche en servant d'estimateur de distance entre les buts à prouver et l'ensemble des formules produites en mode dérivation.

Elle décide également de la fréquence avec laquelle alternent les deux modes.

• Choix du but à réduire

En mode réduction, l'objectif est de parvenir à un but identique à une formule atomique de la base. La stratégie sélectionne donc en priorité les buts les plus proches de ce genre de formules. Elle est fondée pour cela sur la différence entre les termes du but et de l'hypothèse, calculée par la stratégie locale.

Deux autres facteurs interviennent également : le travail de déduction encore nécessaire après sélection d'un but de type implication, et la position du but par rapport aux autres. Cette position est définie par l'appartenance du but à une

conjonction de buts, et surtout par leur nombre.

On préférera donc choisir une formule de la base n'entraînant pas une trop grande expansion de l'arbre des buts (ou plutôt une formule comprenant une courte partie antécédent), et un but le moins composé possible avec d'autres.

• Choix de la formule à développer

Remarquons d'abord qu'aucune règle en mode dérivation ne permet d'appliquer la transitivité de l'opération d'implication ; ce choix a été fait pour éviter d'avoir un surplus d'information difficile à gérer, au risque de ne pas pouvoir créer de lemmes.

Les formules à développer sont choisies parmi les hypothèses et parmi les formules résultant de l'emploi d'une hypothèse et d'une implication.

On donne la préférence aux formules les plus instanciées puis aux implications les plus proches des buts, et enfin à celles qui produisent le moins de nouvelles for-mules.

Les critères énumérés pour les choix sont évalués et composés par des fonctions d'évaluation, fd et fr

respectivement pour les modes dérivation et réduction.

$fd(b, f) = \alpha_1 d + \alpha_2 e + \alpha_3 c$

où b est un but, f une formule de la base

d = distance de b à f = Σ différences entre termes

e = nombre de formules en partie antécédent de f

c = nombre de buts liés par conjonction à b.

$fr(f, imp) = \beta_1 i - \beta_2 n + \beta_3 d$

où imp est une implication, f une formule atomique

i = nombre de constantes

n = nombre de formules en partie conclusion de f

d = distance de imp au but le plus proche.

Le prochain but à réduire est celui de plus petite valeur fd. En cas d'égalité on choisit celui de plus petit valeur e, puis d, puis c.

La prochaine formule à sélectionner est celle de plus grande valeur fr, la priorité en cas d'égalité étant ensuite donnée à celle de plus grande valeur i, puis b.

• <u>Fréquence de l'alternance entre les modes</u>.

Une preuve commence toujours par une

extension de la base en mode déduction sur deux niveaux. En entamant ensuite la déduction dans le mode opposé, on prend comme estimation de la longueur de la preuve la longueur du plus grand chemin défini par le plan (bien qu'on soit à la recherche du plus court). Sur la base de résultats connus [F2] on décide d'effectuer la moitié de la distance dans chacun des modes. La fréquence est de trois pas dans chaque mode.

b) Stratégie locale.

Son rôle consiste à calculer la distance séparant deux formules, à partir des caractéristiques des termes qu'il faut égaler. Pour une égalité, la stratégie locale emploie la différence entre les deux membres définie de la manière suivante: à chaque transformation est associé un coût qui ne dépend ni du caractère transformé ni de la nature de la transformation, mais de la position du caractère dans l'arbre représentant le terme. Plus le caractère est placé profondément dans l'arbre, plus son coût est élevé.
La fonction déterminant la distance est :

$$fl = \sum_{K \in E} diff$$

où k représente une égalité

E représente l'ensemble des égalités

$$diff = \sum_{t \in T} c_t$$

avec t désignant une transformation

T l'ensemble des transformations pour égaler les deux membres

c_t désignant le coût de la transformation.

1.7 Domaine d'incertitude. Aide à la correction.

En application de la règle R1, pour tout échec d'un but le système insère dans la base la négation de ce but. Nous appelons domaine d'incertitude l'ensemble des buts niés à un instant donné, puisqu'il représente les conditions pour lesquelle la preuve n'est pas assurée. Ce domaine évolue au cours de la preuve : chaque fois qu'un de ses éléments est utilisé pour prouver un but, il doit être supprimé.

Si la preuve s'avère impossible, l'emploi de certaines des informations dans le domaine d'incertitude

fournit une aide pour corriger les hypothèses et ajouter celles qui manquent.

On retient en priorité les formules portant sur des variables du but initial, et parmi elles celles qui sont les plus proches de la base [D14].

1.8 Traitement des variables quantifiées.

Lors de la mise en forme du problème les quantificateurs sont utilisés pour produire une forme interne skolemisée, sans toutefois être supprimés.

A l'inverse de la partie antécédent où ce sont les variables existentielles qui deviennent des fonctions ou constantes de Skolem, dans la partie conclusion on skolemise les variables existentielles. L'effet pour les variables revient à mettre la formule sous forme normale conjonctive.

Le but de cette opération est de faire de toutes les variables pour lesquelles on recherche explicitement une instance (c'est à dire les variables existentielles), des variables libres dans le nouvel énoncé. Cette technique est très répandue et a été utilisée pour des démonstrateurs de type résolution mais orientés vers une décomposition du théorème en sous-but [B15][C6]

Ainsi $\forall x \exists y \, (\forall z \, P(z) \Rightarrow Q(x,y))$
s'écrit $P(f(y)) \Rightarrow Q(a,y)$

où a est une constante de Skolem, et f une fonction de Skolem.

Lors de la recherche des substitutions à partir de conditions de vérification, on considère comme impossible la résolution d'une égalité dont les deux membres sont ou bien des fonctions de Skolem différentes, ou bien des fonctions composées à partir d'une même fonction de Skolem mais avec des arguments différents.

On admet en revanche que soit tentée la preuve d'une égalité entre un terme, bâti à partir d'une forme de Skolem, et une constante. Ceci correspond à la preuve par reconstitution du domaine de validité.

Soit par exemple à montrer

$$P(1) \wedge P(2) \wedge P(3) \Rightarrow \forall x \, (1 \leqslant x \leqslant 3 \Rightarrow P(x))$$

x étant un entier.
Le théorème se réécrit :

$$(P(1) \wedge P(2) \wedge P(3) \wedge 1 \leqslant a \wedge a \leqslant 3) \Rightarrow P(a)$$

Avec l'algorithme d'unification classique, ni $P(1)$ ni $P(2)$ ne peuvent être employés pour prouver $P(a)$. En autorisant la production du but

$$a = 1$$

on initialise la preuve suivante

but 2 $\quad (a \geqslant 1) \wedge (a \leqslant 1)$

but 3 $\quad (a \leqslant 1)$

Nous n'arriverons jamais à prouver ce but ; il est donc nié $(a > 1)$ et inséré dans le domaine d'incertitude.

La preuve tentée à partir de $P(3)$ amène l'insertion de $a < 3$ dans le domaine d'incertitude.

Il reste à montrer

$$a = 2$$

et on y parvient en composant $a > 1$ et $a < 3$ pour donner

$$a \geqslant 2 \wedge a \leqslant 2$$

N.B. • La preuve telle qu'elle vient d'être réalisée peut être interprétée de la manière suivante : en présence de $P(1)$ et du but $P(a)$ on est assuré que P est vérifié au point 1 ; il suffit donc de montrer qu'il est encore vérifié ailleurs, et pour cela montrer

$$a \neq 1 \Rightarrow P(a).$$

Ce but par notre méthode est obtenu lorsque l'échec de la preuve de $a \leqslant 1$ se répercute sur le but $a = 1$, entraînant sa négation dans le domaine d'incertitude.

• La recherche de substitutions pour des fonctions de Skolem a déjà été effectuée [B 10] dans un démonstrateur de type résolution conçu pour résoudre des équations diophantines.

1.9 Evaluation. Comparaisons.

Nous venons de décrire un système de déduction général possédant toutes les caractéristiques pour faciliter le dialogue avec l'utilisateur.

Cependant rechercher l'interaction ne doit pas consister seulement en la définition de commandes par lesquelles l'utilisateur peut apporter de nouvelles hypothèses, en supprimer, fixer et ajuster en cours d'exécution la profondeur de la recherche, ou encore obtenir une trace des diverses tentatives et en repousser certaines, instancier des variables, etc. Il faut d'abord que l'utilisateur puisse influencer le système sans en connaître le fonctionnement interne. Il faut surtout que le système exprime la difficulté qu'il a à résoudre un problème autrement que par une trace illisible.

La construction de plans devrait garantir la satisfaction de ces deux exigences. Actuellement, les plans élaborés par

le système ont été expérimentés avec succès. Or, faire un plan, c'est s'assigner des étapes successives à atteindre d'un plus haut niveau que les buts développés par le système. De même que l'utilisateur, en fonction de ses aptitudes, est capable d'admettre que certains points de la démonstration soient passés sous silence, les plans doivent constituer le support du dialogue avec le système.

Les plans globaux que nous construisons s'inspirent de travaux effectués sur les moyens de déduction dans les bases de données [D9] [D12]. C'est également de ce domaine [E4] que nous vient la recherche des formules instanciées réalisée par la fonction fr ; quant au principe d'une stratégie prenant en compte les fronts des deux espaces de recherche, il a été lui aussi développé [E1] et comparé [E5]. La recherche multidirectionnelle est envisagée dans les systèmes de type résolution [B13].

Le traitement proposé pour l'égalité apparaît tout à fait inefficace puisqu'il exige que les propriétés de tous les opérateurs apparaissant dans les termes soient spécifiées par axiomes. Nous pensons toutefois que les plans proposés sont une amélioration par rapport aux stratégies utilisées par ailleurs [B18] [C2] qui obtiennent des indications de différences syntaxiques sous forme de bilan numérique (nombre de symboles communs, nombre de symboles différents). Car pour rendre les deux membres identiques, il faut réaliser des manipulations

symboliques, et ce caractère dynamique sont être traduit.
D'autre part le calcul de la distance diffère de ceux qu'effectuent les méthodes les plus proches (distance entre formes syntaxiques [G3]), car les transformations sont réalisées sur des symboles tous définis et le coût de chaque transformation est parfaitement défini.

Enfin la sélection des formules sur la base de l'égalité de leurs termes devrait s'avérer avantageuse pour des développements ultérieurs. La séparation des deux types de preuve (preuves de conditions de substitutions, et autres) devrait permettre de faire la distinction entre des preuves de « haut niveau » et de « bas niveau », avec la possibilité d'introduire des procédures spécialisées pour résoudre ces égalités ; cela devrait être d'autant plus facile qu'en termes de recherche heuristique ce problème est très bien défini, puisque la forme de départ et celle d'arrivée sont aisément comparables.
La génération des conditions de substitution présente par ailleurs l'avantage d'éviter les substitutions mal venues que peuvent effectuer les systèmes implémentant les propriétés de l'égalité sous la forme d'une règle d'inférence [B7] [B8] [C1]. L'emploi de l'égalité est ainsi restreint [B11].

REPRESENTATION INTERNE

RECHERCHE DES SUBSTITUTIONS

Le système de déduction a été exposé sans faire aucune hypothèse sur la représentation des formules.

Nous étudions maintenant une représentation particulière sous la forme d'arbres « typés », ainsi que la procédure de comparaison qui lui est adaptée.

2.1 Notion de type

La notion à laquelle nous faisons référence est celle proposée par M. Caplain dans sa représentation d'un langage de spécification [F2]. Nous considérons que dans toute « phrase » chaque objet désigné par un identifieur possède un type.

En guise d'introduction, observons tout d'abord que les types présents dans les langages de programmation servent essentiellement à garantir l'emploi à bon escient des opérateurs du langage ou des fonctions créées dynamiquement. Ces types ont donc surtout une valeur sémantique.

Or, cette première notion de type, si l'on fait abstraction des représentations internes des objets pour chaque type, peut être rapprochée de celle de référentiel employée dans tout le discours mathématique, notamment dans les théories élémentaires.

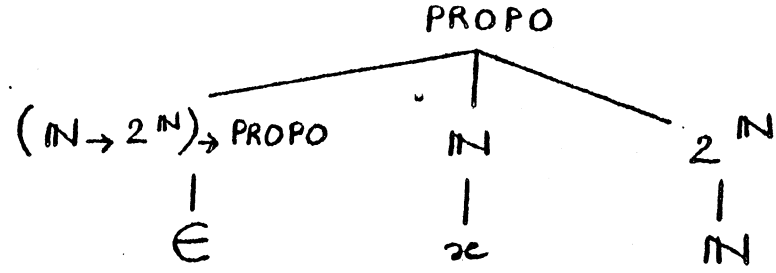
C'est cette association qui est retenue dans [F2] ; ainsi les types de base sont associés à certains domaines remarquables (Entier, Réel). A l'aide de constructeurs tels que l'application ($\rightarrow$), de nouveaux types sont composés pour représenter par exemple la classe des fonctions linéaires.

L'emploi des types par un système de déduction est essentiel lorsque ce système est conversationnel, car il faut pouvoir vérifier la cohérence des informations fournies. Il est impensable d'imposer à l'utilisateur une distinction syntaxique pour les objets qu'il manipule [C9], de même qu'il serait fatal pour le processus de déduction que toutes les informations sur les types soient fournies comme des hypothèses supplémentaires. Une fois vérifiée la cohérence des informations acquises par le système, les types ne sont plus nécessaires au processus de déduction si la notation interne permet de distinguer les opérateurs et les prédicats de même nom externe mais employés dans des domaines différents.

2.2 Représentation interne des formules.

On suppose qu'elles parviennent au système de déduction sous forme préfixée et représentées par des arbres, chaque nœud étant étiqueté par un type (à l'exclusion des feuilles).

Ainsi $x \in \mathbb{N}$ est représenté par :



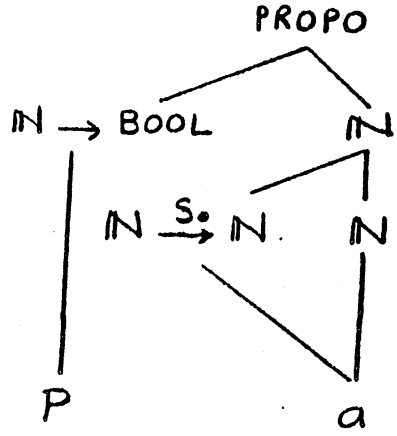
N.B. PROPOsition est le type à la racine de toute formule isolée.

L'opération de skolemisation affecte la représentation interne en créant pour toutes les variables existentielles un nœud intermédiaire portant le type des fonctions introduites. Ce type, du genre application, est composé à l'aide d'un constructeur spécial ($\underline{S}_{\rightarrow}$).

Soit par exemple le théorème à prouver

$$[\forall x\ 1 \leqslant x \leqslant 2 \Rightarrow P(x) \wedge P(3)] \Rightarrow [\forall x\ 1 \leqslant x \leqslant 3 \Rightarrow P(x)]$$

Après mise en forme du problème et skolemisation le but initial s'écrit $P(a)$, où a est une constante de Skolem. La représentation du but est



avec $\underline{S_0}_{\rightarrow}$ fonction constante.

Seules les variables du type ... $\xrightarrow{S}$ ... peuvent être individuellement substituées dans un arbre (voir 1.3). Les autres variables ne sont remplacées qu'avec les termes dans lesquels elles apparaissent, et à condition qu'il n'y ait pas d'occurrence individuelle de ces variables dans d'autres termes.

2.3 Recherche des conditions de substitution

Pour satisfaire le principe de sélection des formules lors de l'application des règles de déduction (application du théorème de remplacement des termes égaux), on ne doit retenir que les formules qui sont identiques (aux arguments près).

Si l'on appelle radical la partie d'un arbre ne contenant pas les arguments, la production des conditions de substitution consiste en la reconnaissance de deux radicaux identiques, puis en la construction des égalités entre termes.

2.4 Recherche des substitutions

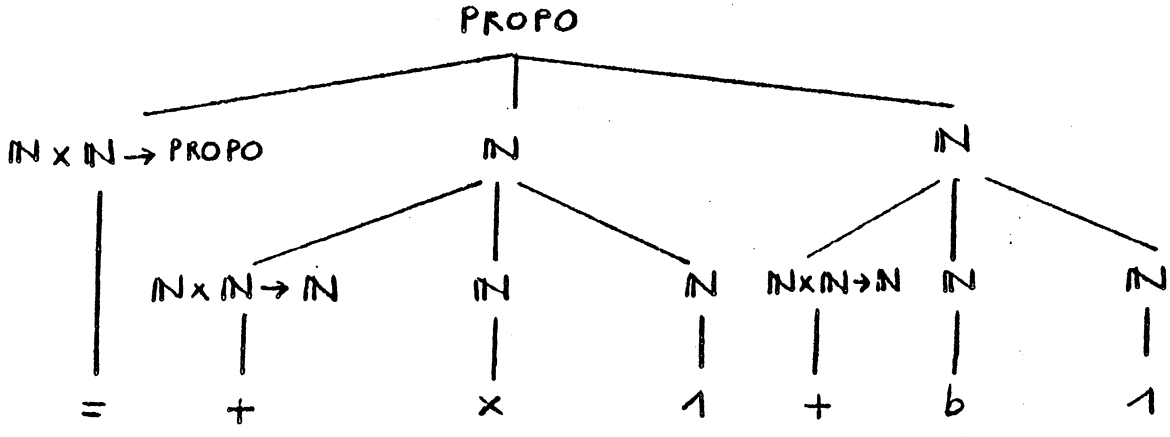
Cette recherche est effectuée pour la règle R4, sur des égalités uniquement et par comparaison des arbres repré-

-sentant les deux membres.

Dans le cas de la condition à démontrer

$$\exists x \quad x + 1 = b + 1$$

représentée par



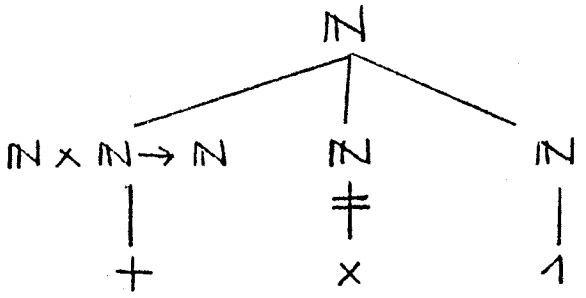
il serait intéressant de pouvoir facilement découvrir que les deux arbres représentant les arguments ne diffèrent que par les branches menant à x et b, le sous-arbre x étant le seul à pouvoir être remplacé par un autre.

• Notion de coupure.

Nous utilisons les coupures pour déterminer dans un arbre toutes les parties substituables. Ces coupures sont placées immédiatement après les nœuds indiquant les types de ces parties.

L'arbre de x + 1 devient alors (les coupures

sont séparées par le signe =)

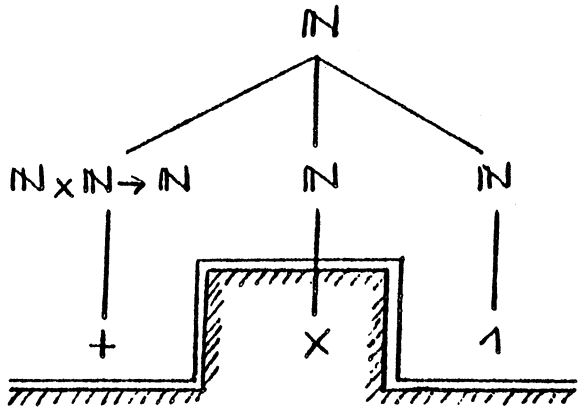


Par extension nous considérons que toutes les feuilles représentant des constantes et des opérateurs sont suivies d'une coupure, et que tous les nœuds – racines des arguments le sont aussi.

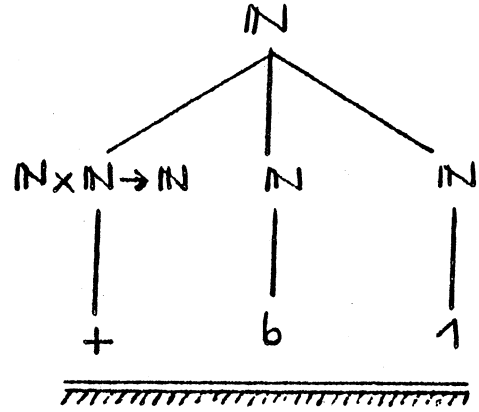
On généralise alors la définition d'un radical en le présentant comme la partie d'un arbre allant de la racine jusqu'à la frontière jalonnée par les coupures.

Ainsi deux arbres représentant deux formules ou expressions identiques (aux substitutions près) lorsqu'ils ont même radical.

En revanche un arbre A1 est une instance d'un autre A2 lorsque la frontière de son radical est plus basse (par rapport à la racine) que celle de A2.



A2



A1

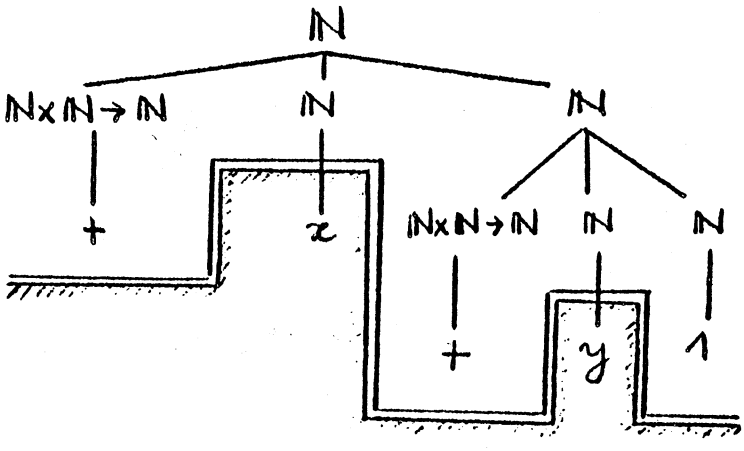
Le radical de l'arbre contenant la partie substituable se superpose parfaitement au radical de l'arbre instance.

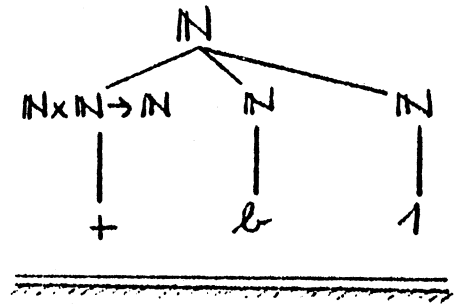
Cette définition n'est pourtant pas suffisante dans le cas ci-dessous :

$$\exists\, x\, y \quad x + (y+1) = b + 1$$

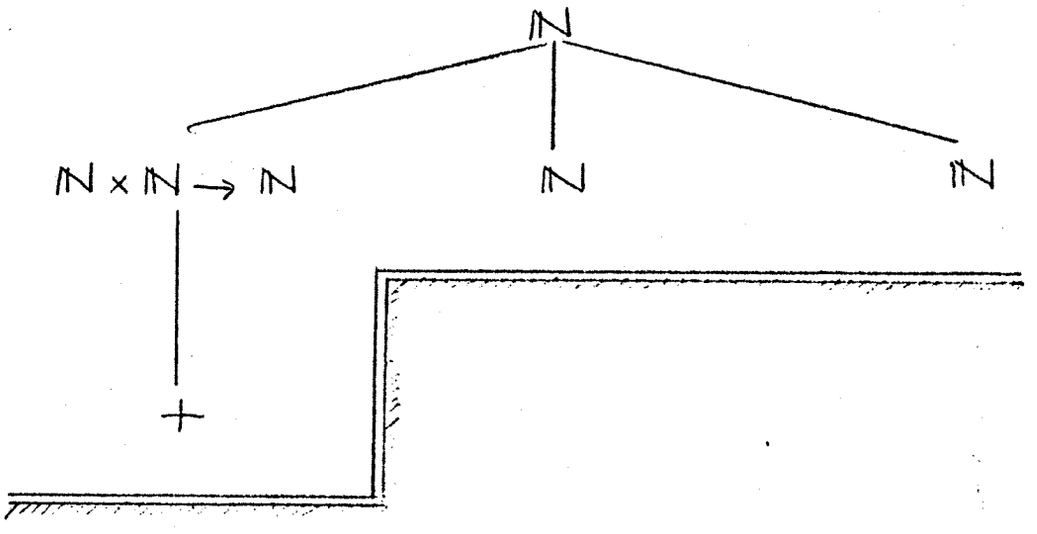
Le système devrait être capable de fournir la substitution $x \leftarrow b$ et la condition $y + 1 = 1$.

Observons les arbres :





On voit que leurs radicaux ne se superposent pas. Pour cela on introduit de nouvelles coupures intermédiaires servant à isoler dans les termes les sous-termes construits à partir de variables.
Le principe de création de ces nouvelles coupures est : tout nœud dont un des successeurs porte une coupure en porte une à son tour. On a alors plusieurs frontières sur un même arbre, ces frontières étant organisées en niveaux.
L'arbre A1 devient alors :



et les parties substituables sont b et $y+1$.
On peut alors engendrer la condition de substitution $y+1=1$.

2.5 Codage spectral des arbres

Pour accélérer la comparaison des radicaux, on propose [D14] de composer un ensemble de codes caractères par chaque arbre, grâce auquel nous ramenons la comparaison de deux arbres à celle de deux ensembles réduits de codes.
Il faut au préalable remarquer qu'un arbre est toujours composé à partir de sous-arbres correspondant à des opérateurs. L'arbre de a + b + c est ainsi composé de deux sous-arbres correspondant à l'opérateur + .

Le spectre est un ensemble redondant de codes comprenant lui-même

- un ensemble ordonné de codes placés sur les nœuds de la frontière et caractérisant le radical ;
- un ensemble ordonné de codes placés sur tous les nœuds de l'arbre.

Chaque code traduit à un nœud la totalité du chemin jusqu'à ce nœud, c'est à dire la suite composé des sous-arbres et les types portés par les nœuds.

Ce code est calculé de la manière suivante :

- le nœud-racine reçoit un code standard ;
- chaque autre nœud reçoit un code dépendant du code du nœud

-père, de la branche du sous-arbre menant à ce nœud et du type porté par ce nœud.

Pour cela, à chaque type est attribué un code au hasard, et à chaque branche d'un sous-arbre est associée une fonction caractéristique F_i (correspondant à l'application d'un opérateur particulier) telle que :

- $\forall i j \quad i \neq j$ F_i et F_j sont linéairement indépendantes, c'est à dire $Prob\,(F_i(a,b) = F_j(a,b)) = \frac{1}{N}$
- les variables aléatoires

Y_n : nombre calculé et affecté au nœud n

Y_p : nombre calculé et affecté au nœud p (père du nœud n)

X_n : nombre aléatoire affecté au type porté par le nœud Y_n

reliées par la formule

$Y_n = F_i(Y_p, X_n)$ sont indépendantes.

En choisissant F_i fonction

- à valeurs dans un intervalle $[0, N-1]$
- à distribution plate $Prob\,(F_i(a,b) = \frac{1}{N})$,

on est assuré que la suite des variables aléatoires Y_n est une suite de variables indépendantes, et indépendantes des X_n si $F_i(k,.)$ est bijective.

Dans ces conditions, on peut garantir que la longueur de collision

(deux noeuds différents recevant un même code) est

$$\frac{N^n - N(N-1)\dots(N-n+1)}{N^n},$$

soit, pour $n \ll n^2$ et N^2 très grand de l'ordre de $\frac{n^2}{2N}$.

La recherche des substitutions entre deux arbres A et B fait alors intervenir les ensembles de codes des spectres notés respectivement C^*_A et C^*_B pour les codes aux coupures, et C_A et C_B pour les autres :

$$A \text{ est une instance de } B \text{ si } C^*_B \subseteq C^*_A$$

Il existe une substitution possible entre A et B si

$$C^*_B / C^*_{A,B} \subseteq C^*_A$$

où $C^*_{A,B} = \{ c \in C^*_A \,/\, c \in C^*_B \}$

$C^*_B / C^*_{A,B} = \{ c \in C^*_B \,/\, c$ n'est pas le code d'un noeud dans un sous-arbre dont la racine appartient à $C^*_{A,B} \}$.

2.6 Codes homomorphes.

En choisissant convenablement les fonctions F_i dans les arbres, on peut traduire dans les codes les propriétés des opérateurs [G5].

Pour l'opérateur + par exemple, la commutativité est prise en compte en attribuant la même fonction F_i aux deux branches - opérandes.

L'algorithme de recherche de substitution peut alors trouver plusieurs correspondances entre les divers ensembles de codes.

TRAITEMENT DES FORMULES ALGÉBRIQUES

Nous avons constaté dans l'introduction que les conditions de vérification de programmes numériques constituent des théorèmes démontrables sans l'emploi d'une méthode de déduction générale. Il est peut-être même souhaitable d'éviter cet emploi.

Ces formules sont en effet composées d'expressions algébriques dont les opérateurs ont des propriétés très riches (commutativité, associativité, distributivité), qui multiplient les formes pour une même expression. La prise en compte de ces propriétés va se faire la plupart du temps par des méthodes ou techniques adaptées et non pas à partir d'axiomes.

3.1. Simplification dans les systèmes de vérification

Dans la plupart des systèmes de vérification [F3], [F7], [F10], on trouve un programme qui officie avant le démonstrateur. Son but est en général double : simplifier la formule, mais aussi la mettre sous une forme normale bien définie, telle que deux expressions *équivalentes* (ayant même valeur pour toute affectation de valeurs à leurs variables libres), soient *identiques* (représentées par la même suite de caractères). Si le symbole = traduit l'équivalence et $\doteq$ l'identité, une *forme canonique* pour une classe d'expressions C est une fonction $f : C \rightarrow C$ telle que

$$\forall e \in C \quad f(e) = e$$

$$\forall e_1, e_2 \in C \quad e_1 = e_2 \supset f(e_1) \doteq f(e_2)$$

Une expression est dite mise sous forme canonique quand $f(e) \doteq e$. Une forme *normale* est un concept plus faible, applicable aux expressions albégriques, qui fait correspondre O à toutes les formules équivalentes à O.

La *simplification* enfin est une opération encore plus faible, qui transforme une expression en une autre, équivalente et syntaxiquement plus simple.

L'emploi des formes canoniques est important à la fois pour la manipulation algébrique et la démonstration automatique, car l'existence d'une telle forme garantit que toutes les expressions qui y sont converties ne sont équivalentes que si et seulement si elles sont identiques.

Pour la première version de SVP, on avait intégré un simplificateur écrit en langage PROLOG [D15] [ ]. Lorsque les conditions de vérification parvenaient ensuite au démonstrateur, elles étaient très proches de tautologies. Ce traitement pourtant suffisant pour des programmes numériques simples, a été abandonné au profit d'un autre qui évite la simplification et qui est plus général.

3.2. Traitements généraux

Lorsque les propriétés des opérateurs sont exprimées par axiomes, les expressions algébriques constituent un domaine difficile à traiter, car l'égalité y est plus largement employée et aucun moyen efficace n'existe pour la prendre en compte [B7].

A cela s'ajoute l'expression des propriétés des relations d'ordre ($>$, $\geq$, $<$, $\leq$), en particulier la transitivité. Ainsi, en présence d'une formule à déduire, presque tous les axiomes peuvent être utilisés à chaque fois. Cela est un facteur agravant pour la taille de l'espace de recherche.

Ces constatations ont amené les auteurs de démonstrateurs à abandonner l'expression des propriétés sous forme d'axiomes. Bledsoe [B22] a choisi de définir un ensemble de procédures heuristiques ; Slagle et Norton [B18] ont incorporé les propriétés des relations dans de nouvelles règles d'inférence qu'ils ont définies suivant un principe présenté dans l'exemple suivant :

L'axiome $(x < y) \vee (y < z) \vee (x < z)$ qui exprime sous forme de clause la transitivité de la relation $<$, est remplacé par une règle que l'on peut résumer ainsi :

> si on a les clauses $(s < t_1) \vee C_1$ et $(t_2 < r) \vee C_2$ et si t_1 et t_2 sont égaux à une substitution près, on en déduit, modulo des substitutions, la clause $(s < r) \vee C_1 \vee C_2$; de cette manière, la propriété ne sera utilisée qu si l'on a effectivement les deux formules $s < t$ et $t < r$ (modulo des substitutions).

Les propriétés des opérateurs, au lieu d'être spécifiées par égalités, ont pu être prises en compte dans le développement d'algorithmes d'unification spécialisés [B3], [B19] ; des études sont en cours pour trouver le moyen général de dériver une procédure d'unification dans le cadre d'une théorie équationnelle [B16],[G2]. On supprime ainsi une grande partie des clauses inutiles développées par les démonstrateurs utilisant la paramodulation comme règle d'inférence [B7] et l'algorithme standard d'unification du premier ordre [B6]. En l'absence de stratégies performantes, ces démonstrateurs passent en effet beaucoup de temps à écrire et réécrire les mêmes termes. Cependant, à l'inverse de l'algorithme standard, ces nouvelles procédures d'unification, pour être complètes, doivent trouver la substitution la plus générale pour un couple de formules atomiques.

3.3. Traitement spécifique

Nous nous intéressons à la classe des problèmes qui consistent à essayer de déduire une égalité ou inégalité à partir d'un ensemble d'autres égalités ou inégalités, aucune n'étant précédée de quantificateur.

Les égalités et inégalités (toutes appelées formules) sont construites à partir de relations (symbolisées par $\mathcal{R}$) et qui sont respectivement notées :
$=$ et $>$, $<$, $\geq$, $\leq$, $\neq$.

Les deux membres des formules sont des expressions composées à l'aide de variables libres (à valeurs constantes dans toute la déduction), de constantes entières et des opérateurs +, x, ↑ (puissance), ÷ (division entière), - (unaire). Toutes ces expressions sont des termes.

La procédure de preuve décrite ci-après admet les théorèmes sous la forme

$$H_1 \wedge \ldots \wedge H_n \supset C$$

Tous les signes de négation, excepté pour ≠, sont supprimés en passant à la relation opposée lors d'un prétraitement.

La procédure ne s'exécute que dans un seul mode : le mode réduction. Si un but est la conjonction de plusieurs formules, chacune des formules est analysée séparément.

a) Méthode de déduction

Après recherche de toutes les hypothèses partageant une expression avec le but (B) et sélection d'une d'entre elles (H), H est résolue en s (sous expression commune) et devient

$$s \mathfrak{R} e$$

Des décisions différentes sont alors prises en fonction de la nature de $\mathfrak{R}$. Elles mènent soit à la création d'un nouveau sous-but, soit à l'abandon de cette voie de recherche.

Pour chaque but à atteindre, la procédure commence par un test de validité. Si le but n'est pas satisfait et non identiquement faux, un nouveau sous-but est engendré.

Dans le cas où s = e, le but créé résulte simplement de la substitution de e à s dans le but actuel. Dans les autres cas, B est résolue à son tour pour se réécrire :

$$s \mathfrak{R}' e'$$

Elle est conservée telle quelle.

Le nouveau but est alors défini automatiquement par la méthode dans laquelle ont été intégrés toutes les propriétés de transitivité des relations et les propriétés les liant. Ces propriétés sont par exemple :

$$(e_1 > e_3) \wedge (e_3 \geq e_2) \supset e_1 > e_2$$

$$(e_1 \geq e_2) \wedge (e_2 \geq e_1) \supset e_1 = e_2$$

$$(e_1 > e_2) \supset (e_1 \neq e_2)$$

Trois types d'échec peuvent apparaître à l'analyse d'un but :

- il est identiquement faux,
- il est équivalent à un but précédent ou à un but sur une autre branche (ce test tient compte du fait que ces buts peuvent ne pas être sous la même forme et vise à faire du graphe de recherche un arbre) ;
- il n'a plus de successeur possible.

L'échec définitif est prononcé quand le but initial lui-même signale un échec. Le succès d'une preuve coïncide par contre avec la découverte d'un but identiquement vrai.
Un cas d'indécision existe et apparaît quand le système n'est pas capable de résoudre les deux formules (but et hypothèse) en s.

b) Stratégie

C'est elle qui décide de l'ordre dans lequel vont être produits les nouveaux buts en déterminant deux classements : un pour les sous-expressions du but retrouvées dans les hypothèses et un autre entre les hypothèses elles-mêmes.

d) Identification des expressions équivalentes

Toutes les formules composant le problème à examiner sont représentées en arbre sous forme préfixée, *sans simplification*.

La détection de deux expressions équivalentes (dans deux arbres différents ou dans un même arbre) doit donc tenir compte du fait que les deux arbres peuvent être de formes différentes.

Au lieu d'une analyse noeud par noeud, la procédure de comparaison garde une vue globale des expressions et de leurs sous-expressions en attribuant des codes de [0, N-1] aux noeuds et en ne considérant en définitive qu'un seul code : celui de sa racine.

Les codes sont attribués à l'aide de la fonction décrite au paragraphe suivant. Ils sont tels que deux expressions équivalentes ont toujours le même code.

Bien sûr deux expressions non équivalentes peuvent avoir été affectées du même code. Ceci justifie le fonctionnement en deux étapes de la procédure de localisation des expressions équivalentes. La détection proprement dite doit être suivie d'une vérification qui peut prendre diverses formes : vérification rapide mais non garantie par nouveau codage, ou plus lente mais sûre par simplification. Nous avons choisi la première.

e) Fonction de codage

La fonction de codage H doit prendre en compte les propriétés des opérateurs et rester invariante sous les différentes transformations qui laissent les expressions équivalentes.

La fonction H est définie à partir de l'homomorphisme de $\mathbb{N}$ dans $\mathbb{N}/p$ (pour p premier) de la façon suivante :

- $H(x + y) \rightarrow (H(x) + H(y)) \bmod p$
- $H(x * y) \rightarrow (H(x) * H(y)) \bmod p$; pour prévenir le risque de dépassement de capacité, p sera choisi inférieur à M, racine du plus grand entier machine ;
- $H(x \uparrow y) \rightarrow (H(x) \uparrow H(y)) \bmod p$; en vertu du théorème de Fermat assurant que $(a^p = a) \bmod p$ (assertion équivalente à $(a^{p-1} = 1) \bmod p$), on a $(H(x) \uparrow H(y)) \bmod p = H(x) \uparrow (H(y) \bmod p-1)) \bmod p$, pour p premier quelconque. On évite le dépassement de capacité lors de l'exponentiation en réduisant cette opération à une suite de multiplications modulo p, fondée sur la décomposition binaire de l'exposant (l'algorithme est simple et a même donné lieu à un programme de test pour les systèmes de vérification). L'exponentiation se déroule alors en $\log_2(p-1)$ pas au maximum.

Des définitions de la somme et de la multiplication on obtient les définitions suivantes des opérations inverses :

. $H(-x) \rightarrow p - H(x)$ puisqu'on doit avoir $(H(-x) + H(x) = 0) \bmod p$

. $H(\frac{1}{x}) \rightarrow (H(x) \uparrow (p-2)) \bmod p$

Pour chaque variable x, H(x) correspond au choix au hasard d'une valeur dans l'intervalle [0, p-1].

f) Utilisation des codes

Ils servent en premier lieu, nous l'avons vu, à repérer les arbres équivalents. Mais ils sont également utilisés pour évaluer des relations.

En définissant H_1 comme

$$H_1(a \mathbin{\text{℞}} b) = H(a - b)$$

. toute identité a pour valeur 0

. des égalités équivalentes comme

$$e_1 = e_2,\ -e_1 = -e_2,\ e_2 = e_1,\ -e_2 = -e_1$$

ont même valeur ou des valeurs opposées modulo p,

. des inégalités équivalentes (par exemple $A \geq E$ et $-E \geq -A$) ont même valeur ;

. des inégalités opposées comme $A \geq E$ et $E \geq A$ ont des valeurs opposées modulo p.

Cette fonction est également utilisée pour découvrir certaines tautologies ($a < a + 1$) que recherche la méthode de déduction. Lorsque deux applications successives (pour des valeurs différentes des variables) de la fonction H_1 produisent la même valeur, on peut considérer qu'il y a deux occurrences d'un même sous-arbre, soit dans chacun des membres de la relation, soit dans le même membre. On recherche ce sous-arbre pour factoriser les expressions et simplifier la relation selon la règle $a = b \supset f(a) = f(b)$.

3.4 Conclusion

Contrairement aux méthodes générales de déduction, la méthode qui vient d'être présentée n'est pas complète. Certains théorèmes dans lesquels les hypothèses sont contradictoires (en particulier si ces hypothèses ne partagent pas de sous-expressions avec les autres hypothèses), ne pourront jamais être démontrés.

De plus, dans le cas de l'égalité, Shostak [B25] fait remarquer que si cette méthode de transformation des formules à travers une suite de substitutions est logiquement cohérente, il n'y a aucun moyen de déterminer avec certitude quelle est la bonne substitution ni combien de fois une même substitution doit dans certains cas, être appliquée.

En utilisant en revanche correctement la notion d'échec et l'heuristique de différences, nous sommes en mesure de fournir à l'utilisateur l'information manquante la plus probable.

CONCLUSION - PROLONGEMENTS

Les dix exemples de démonstrations réalisées par le système nous ont servi à nous assurer du bien-fondé de nos choix.

Cependant la recherche de la généralité est un handicap, car elle implique un manque d'efficacité sur de nombreux domaines que l'utilisateur du système peut considérer comme élémentaires.

On peut dans une certaine mesure compenser cette faiblesse par l'inclusion de procédures spécialisées et fortement heuristiques.

Le moyen d'introduire ce nouveau genre d'information est facile à créer, et nous est offert par les règles, appelées ACTION, dont la syntaxe pourrait être

ACTION : < forme >, < nom procédure > .

Lorsque la formule à déduire est de même forme que <forme> la procédure mentionnée peut être exécutée.
Cette technique s'inspire des appels de procédures dits « pattern-directed invocation » proposés à l'origine par les langages QA4 et Planner. Elle pourrait assez facilement être mise en œuvre, l'analyseur du langage étant capable de fournir au système de déduction l'arbre décrivant la forme

et le comparateur acceptant des formes qui comportent des variables de prédicats et des variables de fonctions. Evidemment l'écriture des procédures ne pourrait être réalisée que par des utilisateurs connaissant bien le fonctionnement interne du système.

Un premier essai d'emploi d'une règle ACTION a déjà été effectué [D 14]. Il portait sur la règle suivante

ACTION : $x = y$, SUBSTITUTION

et sur la procédure SUBSTITUTION, dont le rôle était d'opérer la substitution de x à y en mode descendant et dans des cas bien particuliers.

B I B L I O G R A P H I E

Abréviations :

CACM	Communications de l'ACM (Association for Computing Machinery)
ICRS	International Conference on Reliable Software
IJCAI	International Joint Conference on Artificial Intelligence
JACM	Journal de l'ACM

A. LOGIQUE. RAISONNEMENT MATHÉMATIQUE.

[1] FITCH F.B.
Symbolic logic : an introduction.
Ronald Press, New York, 1952.

[2] KLEENE S.C.
Logique mathématique.
Armand Colin, Collection U, Paris, 1971.

[3] KNEEBONE G.
Mathematical logic and the foundations of mathematics.

[4] LENAT D.B.
Automated theory formation in Mathematics.
IJCAI 5, Cambridge, Août 1977.

[5] MARINOV V.
Computer understanding of mathematical proofs.
IJCAI 5, Cambridge, Août 1977.

[6] PASTRE D.
Observation du mathématicien : aide à l'enseignement et à la démonstration automatique de théorèmes.
DD n° 8, UER de Didactique des Disciplines, Université Paris VII, 1978.

[7] RESCHER N.
Many-valued logic.
Mac Graw Hill, 1969.

B. MÉTHODES DE DÉDUCTION AUTOMATIQUE

. Méthodes générales de base :

[1] ERNST G.W.
A definition driven theorem prover.
IEEE Transactions on Computer, vol. 25, n° 4, 1976.

[2] ERNST G.W.
The utility of independent subgoals in theorem proving.
Information and Control, vol 18, 1971.

[3] NEVINS A.J.
A human oriented logic for automatic theorem proving.
JACM, vol 21, n° 4, 1974.

[4] PIETRZYKOWSKI T.
A complete mechanization of second order type theory.
JACM, vol 20, n° 2, 1973.

[5] PRAWITZ D.
A proof procedure with matrix reduction.
Symposium on Automatic Deduction, 1970. Lecture notes in Mathematics 125, Springer Verlag.

[6] ROBINSON J.A.
A machine oriented logic based on the resolution principle.
JACM, vol 12, n° 1, 1965.

[7] ROBINSON J.A. & WOS L.
Paramodulation and theorem proving in first order theories with equality.
Machine Intelligence 4, 1969.

[8] SIBERT E.E.
A machine oriented logic incorporating the equality relation.
Machine Intelligence 4, 1969.

[9] WANG H.
Formalization and automatic theorem proving.
Congrès IFIP, New York, 1965.

. Méthodes générales élaborées :

[10] BUNDY A.
Evaluation and resolution.
Memo n° 55, Department of Computational Logic, University of Edimburg, 1972.

[11] HARISSON M.C. & RIBIN N.
Another generalization of resolution.
JACM, vol 25, n° 3, 1978.

[12] HUET G.
Constrainted resolution : a complete method for type theory.
Thèse de Ph.D., Case Western Reserve University, 1972.

[13] KOWALSKI R.
A proof procedure using connection graphs.
JACM, vol 22, n° 4, 1975.

[14] KOWALSKI R. & KUEHNER D.
Linear resolution with selection fonction.
Artificial Intelligence, vol 2, 1971.

[15] LOVELAND D.W. & STICKEL M.E.
A hole in goal trees : some guidance from resolution theory.
3rd IJCAI, Stanford, 1973.

[16] PLOTKIN G.D.
Building in equational theories.
Machine Intelligence 7, 1972.

[17] SLAGLE J.R.
Automatic theorem proving with built-in theories including equality partial ordering and sets.
JACM, vol 19, n° 1, 1972.

[18] SLAGLE J.R. & NORTON L.M.
Automated theorem proving for the theories of partial and total ordering.
CACM, vol 16, n° 11, 1973.

[19] SLAGLE J.R.
Automated theorem proving for theories with simplifiers commutativity and associativity.
JACM, vol 21, n° 4, 1974.

[20] WOS L., ROBINSON G.A., CARSON D.F.
Efficiency and completness of the set of support strategy in theorem proving.
JACM, vol 12, n° 3, 1965.

[21] WOS L., ROBINSON G.A., CARSON D., SHALLA L.
The concept of demodulation in theorem proving.
JACM, vol 14, n° 4, 1967.

. Méthodes spécialisées :

[22] BLEDSOE W.W.
The Sup.-Inf. method in Presburger arithmetic.
Memo ATP-18, The University of Texas at Austin, Decembre 1974.

[23] BUNDY A.
Doing arithmetic with diagrams.
IJCAI 3, Stanford, 1973.

[24] ELSPAS B., GREEN N.W., LEVITT K.N., WALDINGER R.J.
Research in interactive program proving techniques.
SRI, Menlo Park, Mai 1972.

[25] SHOSTAK R.E.
An algorithm for reasoning about equality.
5th IJCAI, Cambridge, Août 1977.

C. PROGRAMMES DE DÉMONSTRATION AUTOMATIQUE

. Par résolution :

[1] HERTZ A.
Programme de démonstration de théorèmes formulables en logique de prédicats du premier ordre avec égalité.
Thèse de Doctorat de 3ème Cycle, Université de Paris VI, 1975.

[2] NORTON L.M.
Experiments with a heuristic theorem proving program for predicate calculus with equality.
AI. vol 2, 1971.

[3] ROUSSEL P.
Définition et traitement de l'égalité formelle en démonstration automatiqu
Thèse de Doctorat de Spécialité, Université d'Aix Marseille, UER de Luminy, Mars 1972.

[4] SAYA H.
Définition et utilisation des S-L graphes en démonstration automatique.
Thèse de Docteur Ingénieur, Université Scientifique et Médicale de Grenoble, Mars 1975.

[5] SAYA H. & CAFERRA R.
Représentation compacte des matrices et traitement de l'égalité formelle dans la méthode Prawitz en démonstration automatique.
Congrès AFCET, Gif sur Yvette, Novembre 1978.

[6] STICKEL M.E.
The programmable strategy theorem-prover : an implementation of the linear MESON procedure.
Rapport, Department of Computer Science, Carnegie Mellon University, Juin 1974.

[7] WINKER S.K.
An evaluation of an implementation of qualified hyper resolution.
IEEE Transactions on Computer, vol 25, n° 8, 1976.

. Par déduction naturelle :

[8] BALLANTYNE M. & BLEDSOE W.W.
Automatic proofs of theorems in analysis using non standard techniques.
JACM, vol 24, n° 3, 1977.

[9] BLEDSOE W.W.
Splitting and reduction heuristics in automatic theorem proving.
AIJ, vol 2, 1971.

[10] BLEDSOE W.W. & BRUELL P.
A man machine theorem proving system.
AIJ, vol 5, 1974.

[11] DURAND A.
Un programme de démonstration d'exercices d'algèbre.
Thèse de Doctorat de 3ème Cycle, Université Paris VI, 1975.

[12] OUABDESSELAM F.
Proving theorems by pattern difference.
IInd International Joint Conference on Pattern Recognition,
Copenhague, Août 1974.

[13] PASTRE D.
Démonstration automatique de théorèmes en théorie des ensembles.
Thèse de Doctorat de 3ème Cycle, Université Paris VI, Octobre 1976.

. Etudes comparatives :

[14] LAURENCE J.D. & STARKEY J.D..
Experimental tests of resolution based theorem proving strategies.
Computer Science Department, Washington State University, 1974.

[15] MAC CHAREN J.D., OVERBEEK R.A., WOS L.A.
Problems and experiments for and with automated theorem proving programs.
IEEE Transactions on Computer, vol 25, n° 8, 1976.

[16] REBOH R., RAPHAEL B., YATES R.A.
Study of automatic theorem proving programs.
SRI, Menlo Park, Novembre 1972.

[17] SIKLOSSY L., RICH A., MARINOV V.
Breadth-first search : some surprising results.
AIJ, vol 4, 1973.

[18] WILSON G.A. & MINKER J.
Resolution, refinements and search strategies : a comparative study.
IEEE Transactions on Computer, vol 25, n° 8, 1976.

D. APPLICATIONS DE LA DÉMONSTRATION AUTOMATIQUE

[1] BLEDSOE W.W. & MABRY T.
Typing and proofs by cases in program verification.
Machine Intelligence 8, 1977.

[2] COLMERAUER A.
Un système de communication homme-machine en français.
Rapport préliminaire, Université d'Aix-Marseille, UER de Luminy, Octobre 1972.

[3] DAHL V. & SAMBUC R.
Un système de banque de données en logique du premier ordre, en vue de sa consultation en langue naturelle.
Université d'Aix Marseille, UER de Luminy, 1976.

[4] DARLINGTON J.L.
Theorem proving and information retrieval.
Machine Intelligence 4, 1969.

[5] FIKES R. & NILSSON N.J.
STRIPS : a new approach to the application of theorem proving to problem solving.
Artificial Intelligence, vol 2, 1971.

[6] FURUKAWA K.
A deductive question-answering system on relational data bases.
5th IJCAI, Cambridge, Août 1977.

[7] GREEN C.C.
Theorem proving by resolution as a basis for question answering system.
Machine Intelligence 4, 1969.

[8] HUET G. & LANG B.
Proving and applying program transformations expressed with second order patterns.
5th IJCAI, Cambridge, Août 1977.

[9] KELLOGG C., KLAHR P., TRAVIS L.
Deductive method for large data bases.
5th IJCAI, Cambridge, Août 1977.

[10] KOWALSKI R.
Logic and data bases.
Department of Computing and Control, Imperial College, 1976.

[11] KOWALSKI R.
Logic as programming language.
Department of Computing and Control, Imperial College, Avril 1977.

[12] MAC SKININ J.R. & MINKER J.
The use of semantic network in a deductive question - answering system.
5th IJCAI, Cambridge, Août 1977.

[13] MANNA Z. & WALDINGER R.J.
Towards automatic program synthesis.
CACM, vol 14, n° 3, 1971.

[14] OUABDESSELAM F.

Un système de déduction interactif pour la vérification de programmes.

2ème Colloque International sur la Programmation, Paris, Avril 1976.

[15] ROUSSEL P.

PROLOG : manuel de référence et d'utilisation.

Université d'Aix Marseille, UER de Luminy, 1975.

[16] WARREN D.H.D.

Generating conditional plans and programs.

Artificial Intelligence and Simulation of Behaviour Conference, Edimburg, 1976.

E. STRATÉGIES

[1] DE CHAMPEAUX D. & SINT L.
An improved bidirectional heuristic search algorithm.
JACM, vol 24, n° 2, 1977.

[2] GIANNESINI J.
Générateur de plans utilisant une hiérarchie sur un ensemble de buts.
Université d'Aix-Marseille, UER de Luminy, 1978.

[3] KOWALSKI R.
And-or graphs, theorem proving graphs and bi-directional search.
Machine Intelligence 7, Edimburg University Press, 1972.

[4] MINKER J., FISCHMAN D., MAC SKIMIN J.
The Q algorithm. A search strategy for a deductive question - answering system.*
Artificial Intelligence, vol 4, 1973.

[5] POHL I.
Bi-directional search.
Machine Intelligence 6, 1971.

[6] SACERDOTI E.D.
Planing in a hierarchy of abstraction spaces.
3th IJCAI, Stanford, 1973.

F. VÉRIFICATION ET CONSTRUCTION DE PROGRAMMES

. Fondements :

[1] ARSAC J.
La construction de programmes structurés.
Ed. Dunod, Paris, 1977.

[2] CAPLAIN M.
Langage de spécification.
Thèse de Doctorat d'Etat, INPG, Décembre 1978.

[3] FLOYD R.W.
Assigning meaning to programs.
Symposium Applied Mathematics, vol 19, Ed. American Mathematical Society.

[4] FLOYD R.W.
Toward interactive design of correct programs.
Congrès IFIP, Amsterdam, 1971.

[5] HOARE C.A.R.
An axiomatic basis of computer programming.
CACM, vol 12, n° 7, 1969.

[6] MANNA Z. & WALDINGER R.
Synthesis : Dreams => Programs
IEEE Transactions on Software Engineering, vol 5, n° 4, 1979.

[7] WIRTH N.
Program development by stepwise refinement.
CACM, vol 14, n° 4, 1971.

. Réalisations :

[8] AMY B. & OUABDESSELAM F.
SVP : un système interactif pour la validation et la préparation de programmes
1er Colloque International sur la Programmation, Paris, Avril 1974.

[9] AMY B., CAPLAIN M., OUABDESSELAM F.
Système conversationnel de préparation et de validation de programmes.
Rapport de Synthèse Final, Contrat DRME 74.341, Avril 1976.

[10] DEUTSCH L.P.
An interactive program verifier.
Thèse de Ph.D., Université de Californie, Berkeley, 1973.

[11] ERNST G.W. & HOOKWAY R.J.
The use of higher order logic in program verification.
IEEE Transactions on Computers, vol 25, 1976.

[12] GOOD D.I., LONDON R.L., BLEDSOE W.W.
An interactive program verification system.
ICRS, Los Angeles, 1975.

[13] IGARASHI S., LONDON R.L., LUCKHAM D.C.
Automatic program verification I : logical basis and its implementation.
Acta Informatica, vol 4, 1975.

[14] KING J.C.
A program verifier.
Thèse de Ph.D., Université Carnegie Mellon, Pittsburgh, Septembre 1969.

[15] MANNA Z. & WALDINGER R.
The automatic synthesis of systems of recursive programs.
5th IJCAI, Cambridge, Août 1977.

[16] SUZUKI N.
Verifying programs by algebraic and logical reduction.
ICRS, Los Angeles, 1975.

[17] TOPOR R.W.
Interactive program verification using virtual programs.
Thèse de Ph.D., Université d'Edinburgh, 1975.

[18] VON HENKE F.W., LUCKHAM D.C.
A methodology for verifying programs.
ICRS, Los Angeles, 1975.

[19] WALDINGER R.J. & LEVITT R.N.
Reasoning about programs.
Artificial Intelligence, vol 5, 1974.

G. DIVERS

[1] COLES L.S.

Techniques for information retreival and inferential question-answering system with natural language input.

Technical Note 74, SRI, Menlo Park, Novembre 1972.

[2] FAY M.J.

First order unification in an equational theory.

3rd Workshop on Automated Deduction, Austin, Janvier 1979.

[3] FU K.S. & LU S.Y.

A clustering procedure for syntactic patterns.

IEEE Transactions on Systems, Man and Cybernetics, vol 7, n° 10, 1977.

[4] MARTIN W.A.

Determining the equivalence of algebraic expressions by hash-coding.

JACM, vol 18, n° 4, 1971.

[5] OUABDESSELAM F.

A syntax-tree like pattern matcher using the pattern difference principal.

IEEE Conference on Pattern Recognition and Image Processing, Chicago, Mai 1978.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU les rapports de présentation de Messieurs :

- L. BOLLIET, Professeur à l'Université des Sciences Sociales de GRENOBLE

- G. RUGGIU, Chef de Service - LCR THOMSON CSF - ORSAY -

Monsieur Farid OUABDESSELAM

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Informatique".

Grenoble, le 1er Décembre 1980

Ph. TRAYNARD
Président
de l'Institut National Polytechnique