UNIVERSITE de ROUEN

INSTITUT de RECHERCHE
sur
l'ENSEIGNEMENT des MATHEMATIQUES

I R E M

tél : 35 14 61 41

ECRITURE AUTOMATIQUE DE DEMONSTRATIONS EN GEOMETRIE

Philippe GUILLOT
Luc JOLIVET
Jean Luc LEPERD
Claude MOULIN

Groupe du Havre

Mars 1990

IREM de ROUEN

BP 153, 76135 Mont Saint Aignan CEDEX

Introduction :

Un but fixé par le groupe IREM Informatique du Havre était d'améliorer les logiciels éducatifs de géométrie existant par l'adjonction d'un module d'aide à l'apprentissage des démonstrations. Pour se familiariser avec les algorithmes et les structures de données les plus adaptées, il fallait dans un premier temps concevoir un système capable d'écrire de telles démonstrations. Ces lignes présentent les premiers résultats dans ce sens.

Après une présentation de la maquette en Lisp que nous avons produite, nous développerons les différents problèmes posés par l'écriture automatique des démonstrations en géométrie ainsi que les solutions que nous y avons apportées. Enfin, nous détaillerons les structures du programme en accompagnant nos explications des portions du texte source qui nous paraissent significatives.

Présentation et utilisation de la maquette LISP :

Cette maquette a été concue et écrite sous Le_Lisp version 15. Elle est de ce fait interactive. La boucle d'interprétation est celle de l'évaluateur LISP sous-jacent.

Il permet de construire des figures et de demander des preuves concernant cette figure. Pour limiter les ambitions à ce qui semble raisonnable, le programme est supposé connaître tous les points géométriques nécessaires à la démonstration. Sont donc exclues pour l'instant les preuves qui commencent par l'adjonction d'un point judicieusement choisi sur la figure.

Les fonctions utiles sont des fonctions Lisp de type lambda. Il faut donc "quoter" tous les arguments. Ce logiciel ne prétendant être qu'une maquette, la robustesse par rapport aux entrées erronées n'a pas été étudiée. Il faut donc respecter la syntaxe présentée ici sous peine de conséquences incontrôlables.

Les constructions possibles sont :

1) Construction d'un point sur la figure sans propriété particulière. L'utilisateur doit fournir ses coordonnées à l'exclusion de toute autre information. On utilise cette construction pour définir les points A, B et C dans un énoncé du type : "soit ABC un triangle quelconque... ". Cette opération se fait par l'utilisation de la fonction soit :

? (soit 'a 3 1 7 3) => ok

Placera sur la figure un point de nom a et de coordonnées $\frac{3}{1}$ et $\frac{7}{3}$.

Le premier argument doit être un symbole et les autres des entiers.

2) Construction du milieu d'un segment: "Soit m le milieu du segment [ab]..." :

? (milieu 'm 'a 'b) => ok

Cette fonction ajoute à la figure un nouveau point de nom m, dont les coordonnées sont celles du milieu du segment [ab]. a et b doivent être des points déjà définis. Les hypothèses sont enrichies de cette propriété du point m.

3) Construction d'un symétrique par rapport à un point: "Soit b le symétrique de a par rapport à m... " :

? (symétrique 'b 'a 'm) => ok

Comme la précédente, cette fonction calcule les coordonnées de b et complète les hypothèses.

4) Construction du point d'intersection de deux droites: "Soit G le point d'intersection des droites (AB) et (CD)... " ;
? (intersection 'g 'a 'b 'c 'd) => ok

Si les droites sont parallèles, cette fonction retourne echec. Les coordonnées de g sont calculées et les énoncés ajoutés aux hypothèses sont l' alignement de g a et b d'une part et celui de g, c et d d'autre part.

5) Construction de l'image d'un point par une translation. C'est cette construction qui est utilisée pour construire des parallèlogrammes: Le parallèlogramme ABCD peut se construire par exemple en placant trois points quelconques A, B et C, puis en définissant D comme le translaté de A par le bipoint (B,C). Il y a bien sur d'autres possibilités.
"soit a l'image du point b par la translation définie par le vecteur $\overrightarrow{cd}$" :
? (trasnlate 'a 'b 'c 'd) => ok

Cette fonction calcule les coordonnées du point a et complète les hypothèses.

6) Construction du point d'intersection de droites perpendiculaires à des droites données :
? (inter_perp_perp g a b m c d n) => ok
Cette forme construira de manière analogue le point g, d'intersection de la droite perpendiculaire à la droite (ab) passant par m et de la droite perpendiculaire à la droite (cd) passant par n.

Le_Lisp ne disposant pas d'interface graphique, la construction de la figure est abstraite, le tracé géométrique de la figure reste à la charge de l'utilisateur. Cette lacune a été comblée dans le cadre d'une transcription en Turbo-Pascal de la présente maquette[1]. Le_Lisp Version 15 a été choisi pour sa disponibilité, sa structure dynamique et la facilité qu'il offre tant pour implémenter des algorithmes complexes que pour

---

[1] Cette transcription a été réalisée par Claude Moulin.

d'effectuer des adjonctions, des modifications etc...

Une fois la figure construite, l'utilisateur peut demander la démonstration d'un énoncé comme par exemple: "Prouver que les droites (AB) et (CD) sont paralleles.". En cas de succès et après un certain temps de calcul qui dépend de la démonstration demandée, un enchainement d'énoncés, élémentairement liés les uns anx autres, s'affiche. Le nouvel énoncé démontré ainsi que ceux qui l'ont été lors de la démonstration viennent enrichir ceux issus de la construction de la figure, ce qui permet de traiter efficacement le cas des questions enchainées.

? (prouver '(milieu a b c) 5) => ...

Le premier argument doit être un énoncé valide de littéral milieu, aligne, tri_rect ou equidistant suivi de trois noms de points ou bien de littéral equip, parallèle, isom ou perp suivi de quatre noms de points.

Le deuxième argument est le seuil d'élagage de l'arbre de recherche. Ce doit être un entier et il correspond au nombre maximal d'enchaînements de règles utilisés dans la démonstration.

Certains algorithmes de recherche ne nécessitent pas de seuil comme nous le verrons.

La démonstration s'affiche et la valeur retournée est ok ou echec selon le succès ou non de la recherche.

Les littéraux ci-dessus correspondent à des énoncés qu'on peut soumettre à démonstration. Ce sont des prédicats concernant les points de la figure. Ils sont des types suivants :

1) Prouver qu'un point M est le milieu d'un segment [AB] ;

2) Prouver que des droites (AB) et (CD) sont parallèles ;

3) Prouver que des points A, B et C sont alignés ;

4) Prouver que des bipoints (A,B) et (C,D) sont equipollents;

5) Prouver que les droites (AB) et (CD) sont perpendiculaires;

6) Prouver que le triangle ABC est rectangle en A ;

7) Prouver que le triangle ABC est isocèle de sommet principal A ;

8) Prouver que les segments [AB] et [CD] sont isométriques.

L'utilisateur peut alors ajouter d'autres points sur la figure, demander d'autres preuves etc ... selon un mode interactif.

Pour permettre de calculer ces preuves, certains théorèmes de géometrie sont implémentés. Sont entre autres disponibles les suivants:

Dans un parallèlogramme, les diagonales se coupent en leur milieu. Ceci pour prouver qu'un point est le milieu d'un segment ou bien que trois points sont alignés.

Dans un triangle, la droite qui passe par les milieux de deux côtés, est parallèle au troisième. Cette propriété permet de prouver que deux droites sont parallèles.

Un quadrilatère dont les diagonales ont le même milieu est un parallélogramme.

Dans un triangle, la droite passant par le milieu d'un côté, parallèle à un deuxième côté, coupe le troisième côté en son milieu.

Sont aussi implémentées les propriétés concernant la transitivité de l'équipollence et du parallélisme, les liens entre parallélisme et alignement, entre parallélisme et orthogonalité comme par exemple : "si A, B et C sont alignés, si les droites (AB) et (EF) sont parallèles, alors, les droites (AC) et (EF) sont parallèles."

On dispose aussi des propriétés de la médiatrice d'un segment relativement aux triangles isocèles qui ont pour base ce segment.

Bien sûr, ces propriétés ne traitent qu'une partie très limitée de la géométrie. Certains problèmes liés à la métrique ou à la multiplication des vecteurs par des réels ne sont pas solubles.

D'autre part, pour élaguer l'arbre de recherche, et nous verrons plus loin que des mesures draconiennes ont été prises dans ce sens, l'utilisateur doit fixer un seuil au-delà duquel aucune recherche ne sera plus entreprise.

Malgré ses limites naturelles, ce programme a des performances qui permettent d'espérer qu'un travail en direction de l'efficacité (Le_Lisp est interprété !), du domaine d'application et de la présentation feront de ce logiciel un outil performant, complet et opérationnel.

Parmi les améliorations possibles, nous pouvons citer:

Compléter avec d'autres notions de géométrie ;

Ajouter une figure graphique ;

Concevoir une syntaxe plus facile d'accès et plus robuste ;

Se donner la possibilité de compiler des théorèmes de géométrie selon les besoins des problèmes posés.

Nous pouvons aussi citer quelques utilisations envisageables dans le contexte de la classe, pour faire de ce logiciel un outil pédagogique:

L'élève propose une démonstration, le programme vérifie la validité des enchaînements et, dans le cas où l'un d'entre eux n'est pas valide, il peut vérifier la déductibilité pour éventuellement proposer des étapes intermédiaires.

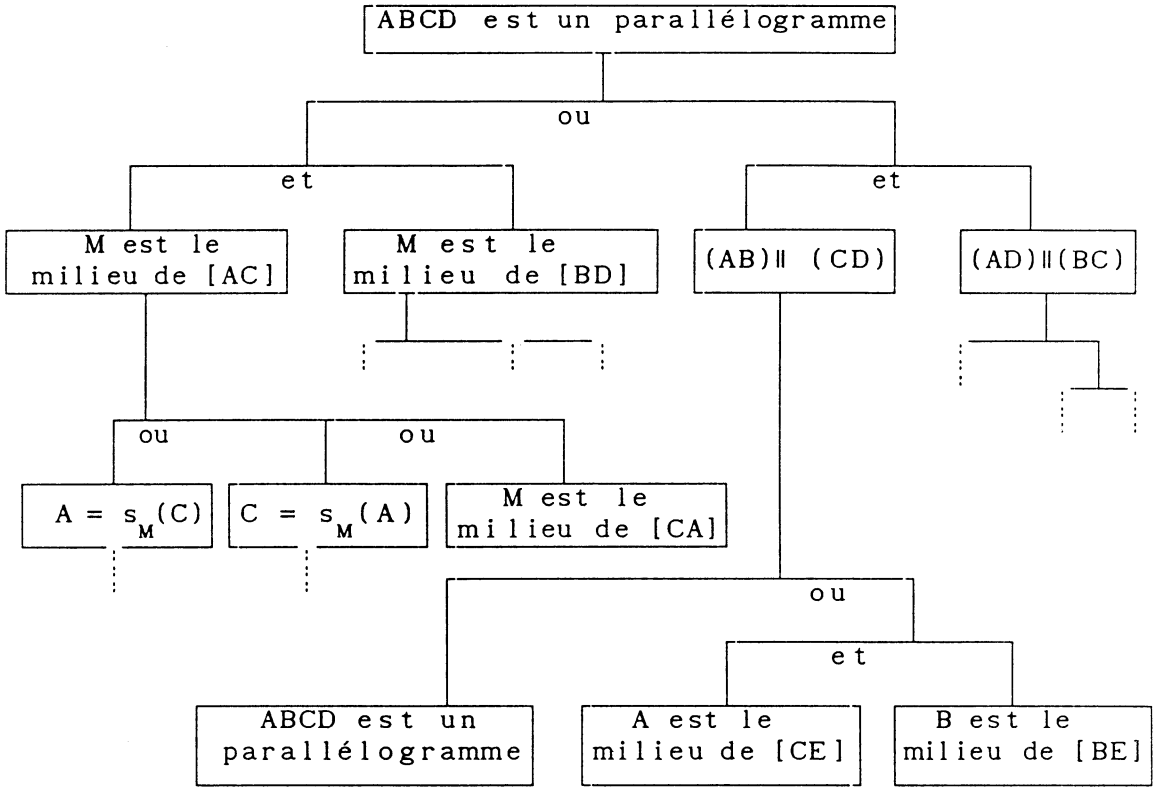
Le programme peut fournir une indication. Celle-ci peut être la dernière étape de la démonstration qu'il a calculé.

Le programme peut vérifier qu'une étape intermédiaire proposée par l'élève permet effectivement de trouver une démonstration.

D'autres utilisations sont certainement possibles et devront être mises au point dans le cadre d'une expérimentation sur le terrain.

Implémentation :

L'algorithme de résolution est très classique. C'est celui qu'on trouve dans les noyaux PROLOG et dans beaucoup de systèmes experts. Le "système de la géométrie" peut être considéré comme un graphe et/ou dont voici une partie très incomplète à titre d'exemple :



Cette représentation doit s'interpréter comme suit :

Pour prouver que ABCD est un parallélogramme, on peut:

Soit prouver que les diagonales ont le même milieu ;

Soit prouver que les côtés opposés sont parallèles.

Pour prouver que les droites (AB) et (CD) sont parallèles, on peut :

Soit prouver que ABCD est un parallélogramme ;

Soit prouver que (AB) est la droite des milieux d'un certain triangle (ici, le triangle CDE).

Pour prouver que M est le milieu du segment [AB], on peut :

Soit prouver que A est le symétrique de C par rapport à M ;

Soit prouver que C est le symétrique de A par rapport à M ;

Soit prouver que M est le milieu du segment [BA].

La recherche de la démonstration se présente alors comme la recherche d'un chemin, de préférence le plus court possible, mais ce n'est pas obligatoire, depuis un état initial qui est la question de l'énoncé, jusqu'à des états terminaux qui sont les hypothèses du problème, établies lors de la construction de la

figure.

L'écriture de la démonstration consistera donc à afficher ce chemin. Celle-ci se présente comme un graphe ordinaire, extrait du graphe "et/ou" de recherche, d'où sont éliminées les branches "ou" qui n'aboutissent pas.

L'exemple présenté met en évidence un certain nombre de difficultés sur lesquelles les systèmes existant actuellement échouaient, en particulier pour cause de bouclage ou d'explosion combinatoire. Voici présentées les réponses que nous avons apportées à ces difficultés.

Nécessité d'une écriture canonique des énoncés :

Certains énoncés sont équivalents bien qu'écrits différemment;

"M est le milieu de [AB]" et "M est le milieu de [BA]"

"ABCD est un parallélogramme" et "DCBA est un parallélogramme"

Le passage de l'un à l'autre ne doit pas se faire par l'application d'une règle, comme cela a été présenté plus haut, sous peine de voir le système se perdre dans d'interminables réécritures ou dans des démonstrations fleuves dans lesquelles l'essentiel est dilué.

Ce problème peut se résoudre par la convention d'une écriture canonique des énoncés basée sur l'ordre lexicographique des noms des points. C'est par la suite, cette écriture canonique seule qui est manipulée par le système.

Nécéssité d'une figure :

Pour appliquer une règle du type "ABCD est un parallélogramme si M est le milieu des segments [AB] et [CD]", un système PROLOG classique considère les noms A, B, C, D et M comme des variables qu'il va chercher à lier à tous les points qu'il connaît. Pourtant, il est clair qu'il est inutile de chercher à lier M à autre chose que le milieu effectif du segment [AB]. Or ce milieu se voit sur une figure. Les différents symboles manipulés n'ont pas à être des variables. On ne cherche pas à démontrer des

théorèmes généraux, mais seulement à résoudre des exercices dans des situations précises.

Pour chercher à appliquer la règle ci-dessus, on cherchera d'abord sur la figure quel est le milieu du segment [AB]. Si ce milieu n'existe pas, cette règle ne sera pas appliquée. S'il existe et qu'il s'appelle M, alors seulement on appliquera la règle. On dispose donc d'une part d'une figure qui est ici une liste de points munis de coordonnées, sur lesquels tous les calculs sont permis pour guider l'intuition, et des hypothèses, qui sont seules utilisées pour établir les preuves.

Plusieurs siècles de pratique de la Géométrie nous garantissent la compatibilité entre ces deux systèmes : ne sont démontrables que des propriétés vraies. Cette remarque fondamentale est notamment utilisée lorsqu'on soumet un énoncé faux à la démonstration.

Ce système permet en outre de choisir la règle qu'on va appliquer :

Pour prouver que les droites (AB) et (CD) sont parallèles, le calcul des coordonnées des vecteurs $\overrightarrow{AB}$ et $\overrightarrow{CD}$ va guider la recherche de la démonstration. Si ces vecteurs sont égaux, on pourra chercher à prouver que ABCD est un parallélogramme. Si par contre $\overrightarrow{AB} = 2.\overrightarrow{CD}$, on pourra chercher à démontrer que la droite (AB) est la droite des milieux d'un certain triangle. En tout état de cause, ces deux étapes sont exclusives, elles ne sont donc jamais explorées simultanément lors d'une même démonstration.

Cette méthode effectue donc, on le voit, un élagage féroce de l'arbre de démonstration et a contribué à accélérer l'établissement des preuves de façon impressionnante.

Détecter les bouclages :

Dans l'exemple de graphe de démonstration présenté ci-dessus, on voit apparaitre le cercle vicieux suivant :

ABCD est un parallélogramme si (entre autres) les droites (AB) et (CD) sont parallèles et ...

Les droites (AB) et (CD) sont parallèles si ABCD est un parallélogramme. etc...

Ce genre de boucle se rencontre aussi lorsqu'on veut

appliquer des règles de transitivité.

La solution adoptée pour éviter de se perdre dans ces boucles est de mémoriser dynamiquement la liste des énoncés intermédiaires entre la racine de l'arbre de démonstration et l'énoncé courant qu'on cherche à prouver. Si une règle fait apparaître un énoncé faisant partie de cette liste, c'est qu'on est dans une situation de bouclage, la branche correspondante ne sera pas explorée.

Nous verrons d'ailleurs que certains algorithmes de parcours de l'arbre de démonstration nécessitent aussi la mémorisation de ces énoncés intermédiaires.

Les principales structures des données :

Il existe trois variables globales: figure, faits et sol, contenant respectivement les points de la figure et leurs coordonnées, les hypothèses et les énoncés démontrés. Les coordonnées sont les coordonnées cartésiennes écrites sous forme de fraction pour éviter les problèmes d'arrondis posés en virgule flottante comme l'égalité à zéro, les approximations etc... Un point est donc repéré par quatre entiers. Il a été nécessaire d'écrire une famille de fonctions spécialisée dans les calculs fractionnaires.

Une fraction est un doublet de deux entiers. De plus, afin de limiter les constructions de doublets, ces fonctions prennent en premier argument le doublet où sera physiquement écrit le résultat. Les résultats sont simplifiés avec dénominateur positif. La fonction de simplification opère sur place, toujours dans le but de limiter les constructions de doublets.

```
(de simplifier (fx)
   (if (zerop (car fx))
       (rplacd fx 1)
       (let ((p (pgcd (abs (car fx)) (cdr fx))))
            (rplac fx
                   (div (car fx) p)
                   (div (cdr fx) p)))))
```

La figure est une liste d'association de la forme :
((a (5 . 1) 3 . 1) ...)

La clé est le symbole donnant le nom du point considéré ;

La valeur est un doublet (abcsisse . ordonnée), chacune de ses composantes étant elle même un doublet (numérateur . dénominateur).

Deux fonctions d'accès sont nécessaires concernant la figure:

(coord 'a) => ((5 . 1) 3 . 1) ; rendant les coordonnées du point donné en argument.

(point '((5 . 1) 3 . 1)) => a ; rendant, s'il existe le symbole de la figure dont les coordonnées sont données en argument.

Ces fonctions utilisent naturellement les fonctions de Le_Lisp spécialisées pour les A-listes: cassq et rassoc.

```
(de coord (a) (cassq a figure))
(de point (ca)
    (car (rassoc ca figure)))
```

Les hypothèses sont aussi rassemblées dans une liste d'association. Chaque hypothèse est un doublet dont le car est l'énoncé du fait lui-même et le cdr un éventuel commentaire. En effet, lorsqu'on déclare que B est le symétrique de A par rapport à M, il a été choisi d'écrire comme hypothèse que M est le milieu du segment [AB] suivi du commentaire : "car B est le symétrique de A par rapport à M" plutôt que d'écrire une règle spécifique qui aurait alourdi le fonctionnement de la résolution.

Une fonction fait est chargée de tester si un énoncé fait partie des hypothèses.

Les énoncés sont écrits (Lisp oblige...) en notation préfixée, comme une liste dont le premier élément est un littéral et le reste la liste des arguments. Ils sont pour l'instant de huit types :

(milieu a b c) : a est le milieu du segment [bc] ;

(equip a b c d) : les bipoints (a,b) et (c,d) sont équipollents ;

(parallele a b c d) : les droites (ab) et (cd) sont parallèles ;

(aligne a b c) : les points a, b et c sont alignés ;

(perp a b c d) : les droites (ab) et (cd) sont perpendiculaires ;

(isom a b c d) : les segments [ab] et [cd] sont isométriques ;

(tri_rect a b c) : le triangle abc est rectangle en a ;

(equidistant a b c) : le point a est à égale distance des points b et c.

Ils sont tous écrits sous une forme canonique :

Pour le milieu, les extrémités du segment sont dans l'ordre alphabétique.

Pour l'équipollence, on impose au premier argument d'être le premier dans l'ordre alphabétique, l'ordre des autres points s'en déduit immédiatement.

Pour le parallélisme, l'isométrie des segments et l'orthogonalité, on impose aussi au premier argument d'être le premier dans l'ordre alphabétique et en plus, les droites sont écrites dans l'ordre alphabétique.

Pour l'alignement, les trois arguments sont dans l'ordre alphabétique.

Pour l'équidistance et le triangle rectangle, les deux derniers arguments sont dans l'ordre alphabétique.

Des fonctions (milcan, equican, paracan, alican etc ...) sont chargées de rendre l'écriture canonique d'un énoncé. Elles utilisent des fonctions de tri très sommaires écrites pour la circonstance. Ce tri n'opérant au maximum que sur trois termes, il est inutile voire nuisible de faire appel à des algorithmes élaborés.

```
  Citons pour exemple:
(de milcan (a b c)
    (if (alphalessp b c)
        (list 'milieu a b c)
        (list 'milieu a c b)))

(de trier2 (a b)
    (if (alphalessp a b)
        (list a b)
        (list b a)))

(de inserer (a l)
   (cond ((alphalessp a (car l)) (cons a l))
         ((alphalessp a (cadr l)) (cons (car l) (rplaca l a)
         (t (newr l a))))

(de alican (a b c)
    (mcons 'aligne (inserer a (trier2 b c))))
```

Ici, la fonction inserer n'est chargée que d'insérer à sa place l'élément a dans la liste à deux éléments l ;

La fonction trier2 ne trie que des listes de deux éléments.

De plus, avant d'essayer de prouver un énoncé, une vérification a lieu sur les coordonnées des points concernés. Notamment, la figure est constamment balayée pour chercher quels sont les points intermédiaires qui serviront à la démonstration. Afin d'effectuer ces calculs en temps minimum, on a introduit un coefficient normal d'une droite : il s'agit du couple des plus petits entiers (a,b) d'une équation de la droite de la forme ax+by+c=0, avec b positif ou bien b=0 et a=1.

Le graphe de la démonstration :

Comme on l'a vu, un état de ce graphe est une liste d'énoncés. Le premier d'entre eux est celui dont on est en train de chercher la preuve et les suivants sont ceux dont il hérite. La nécessité de mémoriser ces derniers est rendue nécessaire pour éviter les cercles vicieux.

Le graphe est défini par une fonction succ. Celle-ci rend une disjonction de conjonctions d'énoncés correspondant respectivement aux arcs "ou" et "et" du graphe présenté plus haut.

Nous aurons par exemple :

(succ '((aligne a b c))) =>

((((parallele a b a c) (aligne a b c))) ; 1ère conjonction

```
(((aligne a b d) (aligne a b c))
 ((aligne a c d) (aligne a b c)))        ; 2ème conjonction
)
```

On peut comprendre ce résultat ainsi :

Pour prouver que a b et c son alignés, on peut soit prouver que les droites (ab) et (ac) sont parallèles, soit trouver un point d tel que a b d d'une part et a c d d'autre part sont alignés.

Dans les deux cas, on mémorise le fait que ces nouveaux énoncés sont issus de la preuve de l'alignement des points a, b et c, pour ne pas y revenir.

Il est aussi important de noter que le point d est choisi parmi les points de la figure dont les coordonnées permettent d'affirmer qu'il est effectivement aligné avec a, b et c. D'ailleurs, la fonction successeur n'engendre que des énoncés vrais sur les coordonnées. Un premier travail, lorsqu'on soumet un énoncé à la preuve, consiste à vérifier qu'il est plausible. Pour cela, une batterie de précicats effectuant des calculs sur les coordonnées est disponible.

La fonction succ opère donc selon le littéral de l'énoncé en argument et retourne d'autres énoncés qui correspondent à des propriétés ou à des définitions géométriques. Ceux-ci sont directement rendus avec des points de la figure et non pas des variables qui seraient instanciées par ceux-ci comme ce serait le cas dans un modèle de type "PROLOG". Ceci a pour conséquence d'éviter les problèmes d'environnement posés par l'unification.

La puissance du système dépend donc en grande partie de la quantité de théorèmes disponibles et du choix fait quant à leur implantation. Ici, la difficulté est de rendre de façon simple, efficace et surtout exhaustive, les propriétés utilisées dans une démonstration.

Propriétés de particularisation :

```
(parallele a b c d) <---        si a = b   (aligne a c d)
                                si a = c   (aligne a b d)
```

si a = d (aligne a b c)

etc...

Propriétés de transitivité :

(parallèle a b c d) <-- (parallele a b u v) et (parallele u v c d)

(equip a b c d) <--- (equip a b u v) et (equip u v c d)

Ici, il faut balayer la figure pour trouver les points u et v convenables.

Propriété de généralisation :

(aligne a b c) <--- (parallele a b a c)

Propriétés liées à des théorèmes ou à des définitions usuelles:

(equip a b c d) <--- (milieu u a d) et (milieu u b c)
<--- (parallele a b c d) et (parallele a d b c)
<--- (equip a d b c)

(milieu a b c) <--- (equip b a a c)

Théorèmes classiques :

(milieu a b c) <--- (milieu u b v)
(parallele u a v c) et
(aligne a b c)

Pour le théorème "Dans un triangle, la droite passant par le milieu d'un côté, parallèle à un deuxième côté, coupe le troisième en son milieu".

(parallele a b c d) <--- (milieu a c u) et (milieu b d u)

Pour une réciproque du théorème précédent.

Nous renvoyons le lecteur au texte du programme pour le détail de l'implémentation de ces propriétés. A ce niveau, un

important travail d'optimisation est nécessaire pour obtenir un système efficace. Il faut notamment limiter les balayages au minimum nécessaire, éviter les calculs redondants etc...

Le moteur d'inférence :

C'est la partie la plus classique du système. Dans un premier temps, nous avons utilisé un algorithme qui opérait par chainage arrière en profondeur d'abord, d'où la nécessité d'un seuil pour éviter de se perdre dans la première très longue branche.

Sa fonction principale est etablir qui prend un état u en argument et qui rend une valeur booléenne : nil si la démonstration a échoué et non nil sinon. A la fin de son travail, la variable globale sol contient sous forme d'arbre n-aire la démonstration retenue qu'il reste alors à afficher.

Le fonctionnement de la fonction etablir est le suivant :

Si un état est terminal, c'est le succès immédiat, sinon, il faut établir tous les énoncés d'au moins une conjonction des successeurs de u. Ce dernier travail est confié à la fonction etablir_dij qui effectue un appel récursif à la fonction etablir.

```
(de etablir (u)
  (prin '|.|) (flush)
  (or (terminal u)
      (when (<= (length u) seuil)
            (etablir_dij (succ u)))))

(de etablir_dij (i)
  (any (lambda (x)
         (when (every 'etablir x)
               (newl sol (cons (car u)
                               (mapcar 'car x)))))
       i))
```

On peut remarquer au passage la concision et la clarté d'une telle écriture en Lisp et la puissance du style applicatif.

Différentes autres stratégies ont été testées :

En profondeur d'abord sauf si une conjonction est immédiatement prouvée,

En triant les disjonctions selon le nombre décroissant d'énoncés résolus qu'elles contenaient, ou selon le nombre croissant d'énoncés non résolus.

Mais il ne semble pas qu'elles apportent une amélioration sensible au fonctionnement général du système, le temps passé au tri ne compensant pas l'économie des états développés qui en résulte.

Par ailleurs, nous avons expérimenté deux stratégies de recherche en largeur. Elles permettent d'éviter à l'utilisateur d'avoir à fournir un seuil pour la recherche de la démonstraton et ont l'avantage de fournir la démonstration la moins profonde quant au nombre d'étapes. Il a été intéressant de constater que les démonstrations fournies n'étaient pas nécessairement les plus courtes.

La premiére est inspirée d'une méthode générale de transformation d'un graphe et/ou en graphe ordinaire. Elle utilise la variable globale sol dans laquelle tous les résultats déjà démontrés figurent ainsi que leur démonstration. Elle utilise comme structure principale de données une file d'attente dans laquelle figurent des piles de problèmes posés. Si une de ces piles est vide, le problème initial est résolu et sa démonstration figure dans la variable sol. Si c'est la file d'attente qui est vide, le probleme posé n'a pas de solution.

L'algorithme consiste à explorer dans l'ordre ces piles de problèmes et à en extraire le premier. Si celui-ci est déjà résolu, ou bien si l'application d'une règle permet de le résoudre, il est effacé et la règle utilisée enrichit la variable sol. Sinon, on enfile les nouvelles piles de problèmes constituées de la pile courante auxquelles est adjoint tout nouvel énoncé qui contribue à résoudre le problème examiné. Dans ce contexte, il est plus facile de détecter les bouclages de règles. En effet, il suffit, avant d'empiler un nouveau problème, de vérifier que celui-ci ne figure pas déjà dans la pile. On peut simplifier alors l'écriture de la fonction qui rend les successeurs d'un état du graphe de démonstration. Il devient inutile de mémoriser les antécédents d'un énoncé, la détection de bouclage intervenant au

niveau de l'algorithme de résolution lui-même.

La fonction etablir a été redéfinie. Elle utilise la fonction un_pas qui traite la première pile de problèmes de la file d'attente :

```
(de etablir (le_pb)
  (setq file (list (list le_pb)))
  (setq queue file)
  (tag sol (while file (un_pas (car file)) (nextl file))
           'echec))

(de un_pas (ppb)
  (prin '|.|) (flush)
  (if (resolu (car ppb))
      (if (consp (cdr ppb))
          (un_pas (cdr ppb))
          (exit sol (dem le_pb)))
      (let ((s (succ (car ppb))))
        (when s
           (if (any (lambda (x)
                      (when (every 'resolu x)
                        (newl sol (cons (car ppb x)))))
                    s)
               (if (consp (cdr ppb))
                   (un_pas (cdr ppb))
                   (exit sol (dem le_pb)))
               (mapc 'ajouter (rectifier (apply 'nconc s)()
  )))))))))
```

Un défaut de cet algorithme est qu'il recalcule plusieurs fois les successeurs d'un état donné pour l'application d'une même règle. Par contre, la variable sol étant globale, tout résultat prouvé lors de l'exploration d'une branche de l'arbre de démonstration pourra être exploitée par la suite pendant l'exploration des autres branches.

Le second algorithme de parcours en largeur de l'arbre de démonstration utilise aussi une file d'attente contenant des piles de problèmes.

On traite aussi le premier problème en tête de cette file, mais au lieu d'enfiler séparément chacun des énoncés qui contribuent à sa résolution, on enfile, dans la succession du problème courant une seule nouvelle pile par conjonction. Cette

conjonction viendra remplacer le problème courant. On enfile moins de piles mais on se trouve dans l'obligation de mémoriser dynamiquement la règle utilisée. Celle-ci ne peut donc pas être exploitée lors du parcours des autres branches et en définitive, l'efficacité semble moins bonne que pour la méthode précédente.

```
(de etablir (le_pb)
  (setq file (list (list le_pb)))
  (setq queue file)
  (tag sol (while file (un_pas (car file) (nextl file)))
           'echec))

(de un_pas (u)
  (let (((ppb . preuve) u))
   (prin '|.|) (flush)
   (cond ((null ppb)
          (exit sol (dem le_pb (append sol preuve))))
         ((resolu (car ppb))
          (setq queue (placdl queue (rplaca u (cdr ppb)))))
         (t (let ((s (succ (car ppb))))
              (when s
                (if (any (lambda (conj)
                           (when (every 'resolu conj)
                             (newl sol (cons (car ppb) conj))))
                         s)
                    (setq queue
                          (placdl queue (rplaca u (cdr ppb))))
                    (unless (any (lambda (x) (assoc x preuve))
                       (setq queue
                          (cons (append conj (cdr ppb))
                                (cons (cons (car ppb) conj)
                                      preuve))))))))))
```

Compilation des théorèmes de géométrie :

Il s'agit pour l'instant d'une "compilation manuelle": Chaque théorème de géométrie fait l'objet d'un module dans la fonction successeur décrite plus haut. Il serait intéressant d'automatiser cette écriture. Il faudrait définir une syntaxe pour énoncé les théorèmes de géométrie et les traduire automatiquement en une section de programme intégrée dans la fonction de génération du graphe de démonstration. Mais ceci est un autre travail...

Conclusion :

La démarche suivie pour concevoir ce logiciel a été de limiter nos ambitions à ce qui nous semblait raisonnablement possible. Nous ne nous sommes pas imposé de cahier des charges rigide à l'avance, mais nous avons progressé tout au long de la mise au point dans le sens d'une efficacité accrue et nous sommes arrivés à un niveau de performances proche semble-t-il de celui d'un élève de premier cycle.

Si le présent travail ne marque qu'une étape de notre projet, il a été riche d'enseignements tant du point de vue de l'informatique et de la programmation que de la pédagogie de la démonstration. A ce propos il a été amusant de constater que la machine comme l'homme a besoin d'une figure pour mener une démonstration.

Notre travail, certes, n'est pas parfait, mais nous espérons qu'il permettra à d'autres d'entreprendre avec plus de succès encore un projet analogue et dont nous serions heureux d'avoir communication des résultats.

Bibliographie :

[1] P. Boizumault ; PROLOG l'implantation ; MASSON 1988.

Cet ouvrage décrit de façon très détaillée divers interprètes PROLOG, écrits en LISP.

[2] J. Chailloux ; Le_Lisp manuel de référence ; INRIA 1984.

Tout ce qu'il faut savoir sur Le_Lisp...

[3] Farreny et Ghallab ; Eléments d'intelligence artificielle ; Hermes, 1987.

C'est dans cet ouvrage que nous avons trouvé ce qui nous a permis de mettre au point nos moteurs d'inférence.

[4] S. Such et alii ; Mathématiques Classe de $3^{ème}$; Collection Such-Durrande ; Bordas 1989.

Ce manuel contient une liste de procédés de démonstrations proposés aux élèves de collèges.

Titre : Ecriture Automatique de démonstrations en géométrie

Auteurs :
Philippe GUILLOT, Luc LEPERD, Claude MOULIN.

Public concerné :
Professeurs de Mathématiques du secondaire. Tout public intéressé par l'informatique, la géométrie et l'intelligence artificielle.

Résumé :
Présentation d'un programme d'écriture automatique de démonstrations en Géométrie. Développement des problèmes posés par ce type de logiciel et des solutions qui y sont apportées.

Mots clefs :
Géométrie. Démonstration automatique. Informatique. Intelligence artificielle. Système expert.

Date 1990

Nombre de pages 22

Prix 20 f

Publication :
ISBN 2-86239-026-7
Dépos légal : 1° trtimestre 1990

IREM Université de Rouen

rue Thomas Becket

76130 Mont Saint Aignan