Colloque Intelligence Artificielle
Toulouse - 16-20 Septembre l985

Publication du Laboratoire C.F.Picard n° 58

M A T H E T M E T A

Dominique Pastre

U.E.R. de Mathématique et Informatique de l'Université Paris VII
et
Laboratoire C.F.Picard de l'Université Paris VI

Résumé : Pour les systèmes travaillant à un niveau méta, les mathématiques offrent-elles une particularité ? Les réalisations ont-elles des points communs ? Le problème est-il particulièrement difficile ? Ce papier est un essai de réflexion sur ce sujet, à la lumière de quelques systèmes faisant des mathématiques, et raisonnant au niveau méta.

1. Introduction

Rares sont les systèmes d'Intelligence Artificielle qui traitent les *mathématiques* et travaillent à un *métaniveau* (qu'i1 s'agisse de métaconnaissances, métarègles, métaheuristiques, métathéorèmes, ...).

Les mathématiques offrent-el1es une particularité au niveau méta ? Les réalisations ont-elles des points communs ? Les besoins sont-ils spécifiques ? Le problème est-il particulièrement difficile ?

Ce papier est un essai de réflexion sur ce sujet, à la lumière des systèmes suivants :
- AM (1976) et EURISKO (1982) de D.B.Lenat [LEN76 à 83],
- Un Programme de Démonstration de Théorèmes (1966) de J.Pitrat [PIT66/70],
- MUSCADET (l984) de l'auteur [PAS84-85].

Le système CAMELIA (1984) de M.Vivet offre également un bon support de réflexion. Il a été analysé à diverses reprises par son auteur [VIV84].

AM découvre des concepts mathématiques. EURISKO le généralise, découvre aussi des heuristiques et manipule des métaheuristiques. Le Démonstrateur de Théorèmes de J.Pitrat, appelé D.T. dans la suite de ce texte, découvre des théorèmes, découvre et utilise des métathéorèmes. MUSCADET, qui démontre des théorèmes dans divers domaines mathématiques dont certains difficiles, et CAMELIA, qui résout des problèmes difficiles en Calcul Formel, utilisent des métarègles et des métaconnaissances, générales ou spécifiques aux mathématiques.

Ces systèmes présentent des points communs et des analogies, mais aussi de grandes différences. Des confusions sont possibles du fait qu'aux mêmes mots, ne correspondent pas toujours les mêmes concepts. Les distinctions entre les différents niveaux ne sont pas les mêmes.

Néanmoins cette étude fait apparaitre que les mathématiques offrent un domaine d'application particulier pour l'Intelligence Artificielle.

2. Les mathématiques sont-elles une discipline à part ?

Dans la *vie courante*, tout le monde parle des mathématiques, ce qui n'est pas le cas, par exemple, pour la physique et la biologie qui sont des disciplines scientifiques comparables. Tout le monde en a fait, et peu ont bien compris. Tout le monde s'en sert un peu (par exemple pour faire des calculs) ou aimerait s'en servir.

L'enseignement des mathématiques a fait couler plus d'encre que tous les autres réunis. On y parle de difficultés spéciales, de blocages, de problèmes affectifs. Les titres sont révélateurs : *Echec et Math* de S.Barruck, *Mathématiques et affectivité* de J.Nimier, les locutions consacrées et passe-partout aussi : *j'ai toujours été nul(le) en math*, la *matheuse à lunettes*, ... Rien de tel en physique ! Les I.R.E.M. (Instituts de Recherche sur l'Enseignement des Mathématiques) ont été, et de loin, les premiers tels Instituts créés.

Des raisons plus profondes font des mathématiques une discipline *à part*, D'abord son caractère *abstrait* et son absence de support concret ; on part de *rien*, (ou plutôt de ce que l'on s'est *donné*). Puis la nécessité absolue de *comprendre*, et pas seulement d'*apprendre* . On y rencontre à la fois le *raisonnement* et la *formalisation du raisonnement.* Enfin, les mathématiques peuvent s'appliquer à elles-mêmes ; on parle quelquefois de la Logique comme des mathématiques appliquées aux mathématiques pures.

En Intelligence Artificielle, les mathématiques ont été un des premiers domaines étudiés, sans doute parce que c'était un domaine bien formalisé, et bien connu des chercheurs en Intelligence Artificielle. Depuis, par l'intermédiaire de la Logique, tous les systèmes en font un peu. De plus, beaucoup de systèmes, initialement conçus pour d'autres domaines, ou bien se prétendant généraux, essaient d'en faire. Mais il y a souvent alors confusion entre des concepts, flous, de la vie courante et les concepts mathématiques, précis, qui les formalisent. Par exemple, les ensembles sont souvent définis en extension, ce qui serait très limité en mathématiques. Une conséquence fâcheuse en est la suivante : si on n'a pas le *fait* $x \in A$, on en déduit parfois que $x \notin A$, ce qui serait correct dans un *monde clos*, mais les mathématiques ne constituent pas un monde clos : l'appartenance à A peut être difficile à démontrer ou même être une conjecture !

Enfin, le caractère spécial des mathématiques est apparu avec force à. D.B.Lenat au cours de ses travaux puisqu'il dit en 1982 [LEN83b] : "Our original 1976 assumption was that heuristics could be treated just like math concepts, and we would apply the same methods (heuristic search) to discover new ones. But ... heuristics *are* like most other domains, it's *mathematics* that's special ... ."

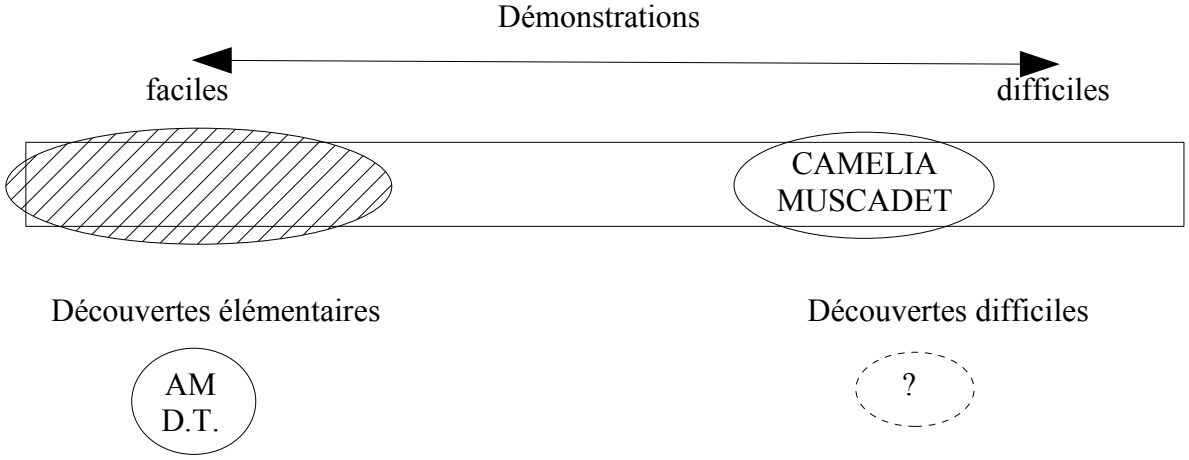
3. Découverte et démonstration

Découverte et *démonstration* sont des activités différentes, bien que complémentaires.

AM ne fait que des découvertes et énonce des conjectures mais ne démontre rien. D.T. découvre et prouve ce qu'il fait. Il peut aussi démontrer les conjectures qui lui sont proposées. MUSCADET ne fait que démontrer les théorèmes qui lui sont proposés, CAMELIA effectue des calculs et des démonstrations sur les expressions formelles qui lui sont proposées.

La *découverte* est une activité beaucoup plus difficile que la *démonstration*. Beaucoup de systèmes démontrent des résultats faciles et quelques uns s'intéressent à des problèmes difficiles (sans aller jusqu'à une extrême difficulté) . Aucun système ne fait des découvertes dans des

domaines difficiles.



Bien que AM et D.T. soient des travaux extrêmement intéressants, et leurs performances remarquables, ils ne travaillent que dans des domaines conceptuellement simples (les mathématiques élémentaires pour l'un, le calcul propositionnel pour l'autre), et ne peuvent dépasser un certain degré de difficulté, sous peine de saturation.

Par contre, si les hommes (d'un certain niveau) restent meilleurs que les systèmes en *démonstration*, ils peuvent être dépassés en *découverte*. D.B.Lenat s'est aperçu, en expérimentant EURISKO, qui généralise AM à n'importe quel domaine, qu'un système est avantagé par rapport à l'homme si celui-ci n'a pas beaucoup d'expérience dans le domaine, ce qui est le cas pour le jeu de *wargame* dont les règles ne sont pas toujours conformes au monde réel, et pour l'étude des *circuits intégrés* (*VLSI*) qui est une discipline nouvelle.

C'est aussi le cas des *systèmes formels* et du *calcul propositionnel* où D.T. a parfois de meilleures performances que l'être humain. Il s'agit en effet d'un travail de formalisation qui peut conduire à une gymnastique mentale en dehors de l'intuition, que peu de mathématiciens ont effectué. Mais en mathématiques classiques, cela fait tellement longtemps que les hommes cherchent qu'il n'y a aucun espoir de faire mieux à court terme.

De plus, je pense que si un système partait vraiment de *rien*, il inventerait sans doute d'autres mathématiques, un autre monde. Si nous voulons faire découvrir des choses que nous connaissons déjà, nous donnons inconsciemment comme propriétés désirées, celles que nous nous attendons à voir découvertes. Un système serait-il capable de redécouvrir la roue ? A mon avis, oui s'il reçoit comme spécifications celles de la roue ! Certainement non dans le cas contraire ; il découvrirait peut-être quelque chose d'aussi bien mais différent (l'aile volante, pourquoi pas ... ?).

4. Les découvertes sont-elles vraiment des découvertes ?

Ainsi, pour revenir aux réalisations concrètes qui nous intéressent, ces systèmes n'ont-ils pas reçu, d'une certaine manière, ce qu'ils ont découvert ? Il ne s'agirait plus alors de *découverte* mais d'une *transformation* de connaissances ?

4.1. AM est très lié à LISP. Certains disent même que *tout est dans* LISP !

En effet, chaque concept de départ reçoit une définition (c'est-à-dire un programme) LISP. D'autre part, LISP est très lié aux mathématiques. D.B.Lenat, lui-même, va jusqu'à dire que "AM was actually not walking around in the space of mathematical concepts, it was walking around in

the space of 'small LISP predicates' ", ce qui explique ses succès tant que les *codes* LISP sont courts.

A titre d'exemple, essayons de résumer la découverte du concept *nombre*.

Alors que, dans l'histoire de l'humanité, le concept de *nombre* a d'abord été découvert, pour les raisons évidentes de son utilité, puis le *nombre zéro* (qui a présenté les mêmes difficultés pour être accepté que l'ensemble vide de nos jours), puis, longtemps après, le concept d'*ensemble*, enfin le concept d'*ensembIe vide* n'est pas encore assimilé par tout le monde.

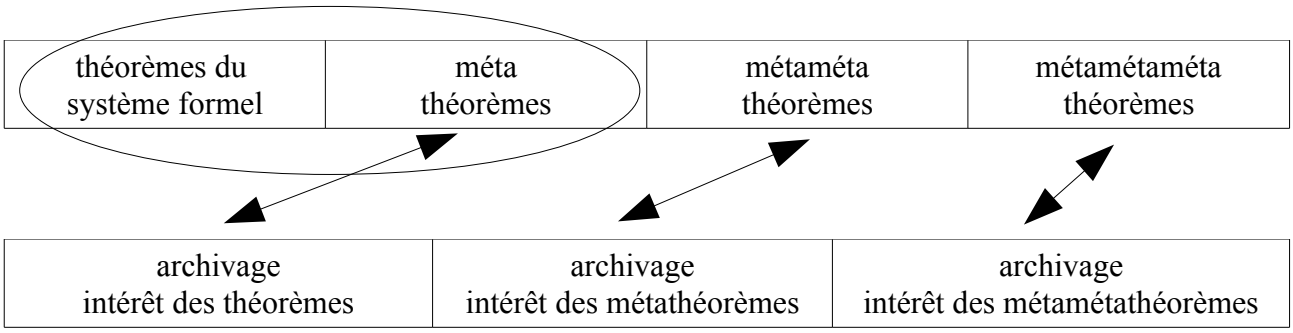
AM, lui, connait initialement des notions ensemblistes élémentaires, en particulier le concept d'*ensemble* et 1'*ensemble vide* qui sert à le définir *récursivement* : un *ensemble* est soit 1'ensemble vide ({} ou Φ), soit une *structure non ordonnée* (concept initial également) telle que, si on lui retire un élément, ce qui reste est un *ensemble*.

De là, AM construit des exemples d'ensembles, soient {Φ} renommé 1 par D.B.Lenat, {Φ,1} renommé 2, {Φ,1,2} renommé 3, ... C'est la construction des ordinaux finis.

Des concepts *égalité*, puis *même longueur*, on découvre qu'il y a isomorphisme entre ces ensembles et tous les autres ensembles, d'où la définition de l'ensemble des structures canoniques (notion de cardinal) renommé *ensemble des nombres*. Plus que la notion de nombre qui sert à compter, c'est cet isomorphisme qui a été découvert.

4.2. D.T. manipule des énoncés formels de 4 types : les théorèmes, les métathéorèmes, les métamétathéorèmes , et les métamétamétathéorèmes. Ces énoncés sont gérés à un autre niveau par un programme. Des règles permettent de générer de nouveaux énoncés ; des heuristiques décident de l'*intérêt* d'un nouvel énoncé, et de son archivage éventuel. Il y a des heuristiques pour chaque type d'énoncé.

On pourrait se demander s'il n'y a pas de correspondances entre les énoncés formels jugés intéressants, et les heuristiques du programme, les théorèmes étant liés, eux, aux métathéorèmes pour des raisons exposées plus loin.



5 . Le démonstrateur de théorèmes de J.Pitrat (D.T. dans ce texte)

Le domaine en est l'étude des systèmes formels pour le calcul propositionnel.

5.l. Systèmes formels - Calcul propositionnel

Un *système formel* est constitué d'*axiomes* qui sont des énoncés posés au départ comme étant des théorèmes, et de *règles de dérivation* qui, à partir des théorèmes déjà connus, permettent d'obtenir de nouveaux théorèmes.

Voici un exemple, avec les connectives ⊃ (implique) et ~ (non), les autres connectives

pouvant être définies à partir de ces deux-là :
Axiomes : $[p\supset(q\supset r)]\supset[(p\supset q)\supset(p\supset r)]$
$p\supset(q\supset p)$
$(\sim p\supset\sim q)\supset(q\supset p)$
Règle de dérivation : $p, p\supset q \Rightarrow q$ (modus ponens)

De nombreuses axiomatiques ont été étudiées pour le calcul propositionnel. Celles qui sont intéressantes sont celles qui sont consistantes, bien sûr, c'est-à-dire non contradictoires, mais aussi *complètes*, c'est-à-dire telles que toutes les *tautologies* (énoncés toujours vrais) sont des *théorèmes* (énoncés qui peuvent être déduits des axiomes). C'est le cas de l'exemple précédent.

On a cherché à minimiser le nombre de connectives primitives : on en prend souvent deux, ~ et $\supset$, ou ~ et $\vee$ (ou) , ou ~ et $\wedge$ (et) , à partir desquelles on peut définir toutes les autres ; on peut aussi n'en prendre qu'une, la *barre de de Sheffer* | [p | q = $\sim(p\wedge q)$] qui permet de définir toutes les autres.

On a aussi cherché à minimiser le nombre et la longueur des axiomes, par exemple
(p|(q|r))|((s|(s|s))|((t|q)|((p|t)|(p|t)))) suffit pour déduire toutes les tautologies ;
((p⊃q)⊃r)⊃((r⊃p)⊃(s⊃p)) permet de déduire toutes les tautologies où ne figure que l'implication, et il n'existe pas d'axiome unique plus court qui le permette.

5.2. Le Démonstrateur

Son but est double : démontrer des théorèmes, mais surtout trouver des théorèmes *intéressants*. Pour cela, J.Pitrat a introduit les métathéorèmes, métamétathéorèmes, et métamétamétathéorèmes.

Les *théorèmes* sont donnés (ce sont les axiomes) ou dérivés (c'est le but du travail). Ceci est purement formel, mais les systèmes formels intéressants sont ceux qui ont une interprétation sémantique logique.

Les *métathéorèmes* ou *productions* sont des règles de dérivation. Ce sont des énoncés, formels aussi, mais distincts des précédents, qui permettent de déduire de nouveaux théorèmes à partir d'anciens. $A\Rightarrow B$ signifie que, si A est un théorème, alors B est un théorème. On peut aussi avoir des productions à deux antécédents $A, B \Rightarrow C$. Les métathéorèmes sont spécifiques au système formel étudié. Certains sont donnés, ce sont les règles de dérivation initiales, par exemple le modus ponens $p, p\supset q \Rightarrow q$. D'autres sont construits. Supposons par exemple que, partant du théorème A, on arrive au théorème B par une suite de dérivations $A \Rightarrow ... \Rightarrow ... \Rightarrow ... ... \Rightarrow B$, les théorèmes intermédiaires n'étant pas intéressants, alors on en déduit la nouvelle production $A\Rightarrow B$ qui peut être considérée comme une sorte de court-circuit, un moyen de se rappeler qu'une certaine succession de dérivations s'est avérée utile. Ceci est à rapprocher de la création de nouveaux concepts dans AM, qui permet aussi de mémoriser certaines propriétés que l'on vient de découvrir.

Les *métamétathéorèmes*, de la forme $A \rightarrow B$ ou $A; B \rightarrow C$, permettent de déduire de nouveaux métathéorèmes à partir d'anciens. Certains sont donnés et généraux, par exemple $A\Rightarrow B$; $B\Rightarrow C \rightarrow A\Rightarrow C$ exprime la transitivité de la dérivation. D'autres sont spécifiques à certaines théories et doivent être démontrés, par exemple $p\supset q \rightarrow p\Rightarrow q$ qui n'a de sens que si on a l'implication $\supset$ dans la théorie.

Les métamétamétathéorèmes, de la forme $A \rightsquigarrow B$ ou $A\Box B \rightsquigarrow C$, servent à démontrer les métamétathéorèmes précédents, par exemple $a, b \Rightarrow c \rightsquigarrow a \rightarrow b\Rightarrow c$. Ils sont tous généraux, donc peuvent être donnés et on peut s'arrêter à ce niveau métamétaméta.

Les métamétathéorèmes (au nombre de 8) et métamétamétathéorèmes (au nombre de 5) expriment ce que font les mathématiciens, bien qu'en général ils ne prennent pas la peine de les

expliciter, et que de toute façon, ce niveau reste informel. Théoriquement, deux métamétathéorèmes généraux (a; $a \Rightarrow b \rightarrow b$ et a ; b ; $a,b \Rightarrow c \rightarrow c$) auraient suffit car ils auraient pu engendrer les autres, mais ceux-ci sont donnés dans un but heuristique. Ceci est à rapprocher de certaines heuristiques de AM, qui ne sont pas indispensables, mais accélèrent le processus de découverte.

Le fonctionnement du système comprend :
- un algorithme de dérivation à tous les niveaux : appliquer un méta^{i+1}théorème à un ou deux méta^{i}théorèmes donne un nouveau méta^{i}théorème;
- la décision d'archiver ou non un méta^{i}théorème ainsi généré ;
- le choix des essais à faire.

Tout ceci repose sur des considérations sur la forme, la complexité, le nombre des méta*théorèmes engendrés, la ressemblance entre partie gauche et partie droite des énoncés, ainsi que de deux calculs d'*intérêts* (un pour simplifier et un pour compliquer).

Le calcul d'intérêt des théorèmes est moins bon que les calculs d'intérêts des méta^{i}théorèmes, car on n'a pas de sémantique. On utilise alors un artifice : on lie au théorème les métathéorèmes qui en sont dérivés, établissant ainsi un lien entre langue et métalangue. Les résultats sont alors bons pour les systèmes formels qui ont une interprétation logique.

Revenons au schéma de la fin du §4. Assimilant les théorèmes et les métathéorèmes, on peut considérer qu'ils concernent le raisonnement. Les métamétathéorèmes concernent alors le métaraisonnement. Les métamétamétathéorèmes, concernant le métamétaraisonnement, sont généraux, et on s'arrête. On retrouve alors l'idée défendue par ailleurs qu'il n'est pas nécessaire de dépasser le niveau métaméta, car celui-ci est général.

Cette difficulté est évidemment propre aux mathématiques . C'est la seule discipline dans laquelle on puisse confondre l'objet sur lequel on travaille et le raisonnement qui permet de travailler dessus.

Le système est également capable de démontrer des conjectures qu'on lui propose, par un fonctionnement voisin, mais cette approche n'apporte pas de résultats aussi intéressants que la précédente. En effet, si les théorèmes proposés sont faciles, le système était capable de les découvrir lui-même ; s'ils sont difficiles, il se heurte à un problème de saturation de mémoire par tous les résultats intermédiaires indispensables découverts. Là encore, il y a analogie avec le comportement de AM qui sature dès qu'on le pousse à étudier des choses difficiles.

6. AM

AM travaille sur des concepts mathématiques qui sont représentés par des *unités.* Chaque unité comporte des attributs qui sont remplis, au départ par l'utilisateur, puis par le système. L'article de B.Starynkevich [STA85], dans cette même publication, décrit cette structure de représentation dans le langage RLL-1. On peut trouver aussi une description succincte et ... en français, de AM dans [PAS8l].

A l'époque de l'écriture de AM (1976), les attributs étaient figés, le système ne pouvait pas en créer.

Voici deux exemples de tels concepts:

Ensemble:
définition: <programme LISP>
spécialisations: ensemble non vide, ensemble d'ensembles,
extrême: Φ, singleton, doubleton

généralisations: collection, structure non ordonnée, ...
exemples: {Φ}, {A}, {A,B}, {3}, ...
conjectures: toutes les structures non ordonnées sont des ensembles

Nombre premier:
définition: c'est un nombre dont l'ensemble des diviseurs est un doubleton
exemples: 2, 3, 5, 7, 11, 13, 17, ...
généralisations: nombre, ...
conjectures: factorisation unique en nombres premiers,
conjecture de Goldbach

Le système comporte d'autre part des heuristiques pour:
- remplir des attributs;
- évaluer des intérêts;
- créer des concepts.

Rappelons que le système ne démontre rien, mais se contente de remplir des attributs, ce qui peut être momentanément faux, et être corrigé par la suite. Ainsi, un *exemple* peut ne pas être tout à fait un exemple, ce qui amènera peut-être le système, plus tard, à modifier légèrement la définition du concept ou à en créer un nouveau. De même, les conjectures seront affinées petit à petit.

AM crée un concept quand il découvre certaines propriétés . On peut le considérer comme une façon de mémoriser un chemin, un raisonnement, comme les métathéorèmes dans D.T.

Ce système a de bons résultats en *théorie naïve des ensembles* et en *théorie élémentaire des nombres*, puis il tourne en rond. Son auteur a perçu alors la nécessité de pouvoir créer de nouvelles heuristiques, ainsi que de nouveaux attributs, ce qui a abouti au système EURISKO, général, qui peut être applique en particulier au domaine de AM.

7. Eurisko applique à AM

7. l. EURISKO a été expérimenté en recevant les concepts initiaux de AM, c'est-à-dire des unités comportant des attributs remplis par l'utilisateur, et une cinquantaine d'heuristiques, également sous forme d'attributs

Ainsi, concepts et heuristiques sont structurés de la même façon, ce sont des unités. Ce ne sont pas les seules unités ; chaque attribut, par exemple, est aussi décrit comme une unité (voir [STA85])·

Le concept *heuristique* est aussi une unité. Une *métaheuristique*, qui est une heuristique sur une heuristique, est donc une heuristique sur une unité, c'est-a-dire simplement une heuristique. Heuristiques et métaheuristiques ne sont donc pas de natures différentes, contrairement à D.T. qui sépare rigoureusement les différents niveaux *méta*, et comme MUSCADET qui ne fait pas non plus de différence entre les connaissances et les métaconnaissances.

Mais un phénomène curieux s'est passé. Quand une unité devient trop grosse, elle est éclatée en deux ou plusieurs unités, selon les propriétés de ses exemples. Ainsi, l'unité *heuristique* a été éclatée en deux, par le système, pour . . . séparer les métaheuristiques des autres !

Au début, une unité-heuristique , comporte deux attributs: *si* et *alors* qui sont tous les deux remplis par des fonctions LISP. Ces unités subissent des transformations, en particulier des créations d'attributs. C'est ainsi que les attributs *si-identité*, *si-constante, si-inchangé*, . . . d'une part, *alors-conjecture*, *alors-écrire*, . . . d'autre part, pourront être créés, en fonction des fréquences avec

lesquelles on les rencontre. L'heuristique sera alors décomposée en plusieurs morceaux. L'intérêt est que l'on peut alors travailler sur ces attributs par des règles générales du genre:

si, pour beaucoup d'unités, les attributs r et s .. . ,
alors créer une nouvelle unité . . .

La nouvelle unité créée pourra être un concept, une heuristique, un attribut, … .

7.2. Quelques exemples

7.2.1. Voici d'abord un exemple de (méta)heuristique qui permet d'éliminer les concepts nouveaux (heuristiques nouvelles) pauvres:

si on vient de généraliser C en G, alors vérifier que G n'est pas identique à C

7.2.2. L'exemple suivant est une construction de plusieurs heuristiques mathématiques (qui étaient données à AM) à partir d'une heuristique générale, soit:

si deux attributs s1 et s2 d'une structure F peuvent avoir le même type
alors définir une nouvelle heuristique :
si f est un F intéressant, et ses attributs s1 et s2 sont de même type
alors définir et étudier la situation dans laquelle les valeurs de s1 et s2 de f sont égales

On pourra l'appliquer aux mathématiques de plusieurs façons:

a) avec F = fonctions,
s1 = premier argument,
s2 = deuxième argument,
on obtient:
si f est une fonction intéressante et ses deux arguments sont de même type
alors définir et étudier g(a)=f(a,a)

Cette heuristique est également utilisée dans MUSCADET dans un cas particulier.

b) avec F = fonctions,
s1 = argument,
s2 = valeur,
on obtient :
il est intéressant de trouver les points fixes d'une fonction

c) avec F = fonctions,
s1 = ensemble de départ,
s2 = ensemble d'arrivée,
on obtient:
une fonction d'un ensemble dans lui-même est intéressante

7.2.3. Voici un autre exemple du même type. EURISKO reçoit l'heuristique générale suivante :

si f(Exemples(A),Exemples(B)) est presque extrême,
alors combiner les définitions de A et B pour obtenir une nouvelle définition

L'application aux mathématiques donne:

a) avec f = différence symétrique,
extrême = petit,

combiner = prendre l'intersection,
on obtient :
si quelques uns (mais non la plupart) des exemples de A sont aussi des exemples de B
et quelques uns (mais non la plupart) des exemples de B sont aussi des exemples de A
alors définir et étudier l'intersection de A et B,
ce nouveau concept est une spécialisation de A et de B

b) avec f = différence,
extrême = petit,
combiner = x,y $\rightarrow$ x $\wedge$ ~y,
on obtient :
si seulement quelques exemples de A ne sont pas exemples de B
alors définir et étudier le concept être A et non B.

7.2.4. L'exemple suivant est une heuristique qui peut s'appliquer à elle-même.

H: *si s1 et s2 sont des attributs remplis par des valeurs de même type,*
et s1(A) est plus intéressant que s2(A)
alors qu'habituellement les valeurs de s1 sont moins intéressantes que celles de s2
alors définir et étudier une nouvelle unité A' similaire à A sauf que s2(A') =s1(A)

a) avec A = un concept,
s1 = non exemples,
s2 = exemples,
on obtient:
si un concept a un complément qui est beaucoup plus petit
alors définir explicitement et nommer ce complément.

b) avec A = cette heuristique H elle-même,
s1 = si assez de temps,
s2 = si intéressant,
on obtient l'heuristique H' obtenue à partir de H en ajoutant à l'attribut *si intéressant* la clause *il y a assez de temps.*
H' s'avère plus utile que H car moins coûteuse. (Ceci est un cas exceptionnel, en général, l'intérêt est plus important que le temps.)

7.2.5. Enfin, l'exemple suivant montre comment EURISKO peut découvrir une heuristique qui aurait été bien utile à AM, à propos des nombres premiers, à savoir que
la factorisation unique en nombres premiers (conjecture A) est plus intéressante que
la conjecture de Goldbach (conjecture B).
La raison, en gros, en est que A a à voir avec la multiplication et la division, que B a à voir avec l'addition et la soustraction, et que le concept *nombre premier* est lié à la multiplication et la division. On retrouvera cette même idée dans MUSCADET pour décider, dans un espace vectoriel topologique, du choix de l'addition ou de la multiplication, dans un contexte contenant, soit un segment [O x], *lié* à la multiplication ([O x] = $\{\lambda * x \mid \lambda \in [0\ 1]\}$), soit un segment [x y] *plus fortement lié* à l'addition si x et y ne sont pas 0 ([x y] = $\{\lambda * x + (1-\lambda) * y \mid \lambda \in [0\ 1]\}$)·

Sans rentrer dans les détails, on va voir que cette découverte est accélérée par la création d'attributs pour représenter des heuristiques.
L'attribut *défini par* du concept *nombre premier* contient le concept *diviseurs*, car les

nombres premiers ont été définis comme les nombres ayant exactement deux diviseurs.

On va s'apercevoir d'autre part que l'attribut *défini par* est un sous-attribut de l'attribut *unités pour bonnes conjectures*, en appliquant, avec r = *défini par*, s = *unités pour bonnes conjectures*, l'heuristique suivante :

si pour beaucoup d'unités, la plupart des éléments de l'attribut r sont aussi sur l'attribut s,
alors affirmer que r est un sous-attribut de s

Cette relation, *r sous-attribut de s* ($r \subset s$), est représentée de la façon suivante : l'attribut *super attribut* de l'unité *r*, (ici *défini par*) contient *s* (ici *unités pour bonnes conjectures*)·

Ceci signifie alors :

si un concept X est défini en termes de Y (c'est-à-dire l'attribut défini par *de X contient Y),*
alors on peut s'attendre à ce qu'il y ait une bonne conjecture entre X et Y
(c'est-à-dire l'attribut unités pour bonnes conjectures *de X contient Y)*.

On en déduit alors que l'attribut *unités pour bonnes conjectures* du concept *nombre premier* contient le concept *diviseurs*.

Puis, pour beaucoup de concepts, on va créer l'attribut *relations entre éléments* de unités pour bonnes conjectures, qui contient *inverse*, *répétition*, *composition*, en appliquant, avec s = *unités pour bonnes conjectures*, l'heuristique suivante :

si un attribut s est important, et tous ses éléments sont des unités,
alors créer un nouvel attribut qui contient toutes les relations entre éléments de s.

Ensuite, l'heuristique suivante:

si pour beaucoup d'unités, l'attribut s contient les mêmes valeurs,
alors ajouter ces valeurs à l'attribut éléments attendus *de l'unité* attribut typique,

appliquée avec s = *relations entre éléments* de unités pour bonnes conjectures, permet d'essayer, entre autres, *inverse* comme élément de l'attribut *relations entre éléments* de unités pour bonnes conjectures du concept *nombre premier*.

Alors, puisque *diviseurs* est dans l'attribut *unités pour bonnes conjectures* de *nombre premier*, son *inverse*, c'est-à-dire la *multiplication*, est aussi dans l'attribut *unités pour bonnes conjectures* de *nombre premier*, ce qui signifie qu'on va plutôt s'intéresser aux conjectures entre *nombre premier* et la *multiplication*, c'est-à-dire à la *factorisation unique en nombres premiers*.

8. MUSCADET

MUSCADET utilise des métarègles. Comme dans EURISKO, il n'y a pas de séparation entre les différents niveaux méta. Le moteur ignore si la règle qu'il considère est une règle ou une métarègle.

Je voudrais faire la différence entre règle/heuristique d'une part, métarègle/métaheuristique d'autre part, les (méta)heuristiques exprimant vraiment des heuristiques, alors que les (méta)règles, s'il leur arrive de traduire des (méta)heuristiques, peuvent aussi n'être que des outils, techniques, pour accomplir des actions. Ici encore, tout est mélangé, le moteur ignore si la (méta)règle qu'il applique est une réécriture élémentaire ou une heuristique mathématique spécifique, ou un maillon d'un plan qui exprime une heuristique complexe, ... Ce choix, fait pour des raisons historiques, a l'inconvénient de diminuer la lisibilité de la base de connaissances. D'autre part, les heuristiques sont rarement énoncées explicitement d'une manière déclarative, mais plutôt traduites par des paquets de règles et métarègles.

8.1. Activation des règles - Ordre

Il y a plusieurs *types* de règles dans MUSCADET. La base de règles est en partie donnée par l'utilisateur, en partie construite par le système.

D'autre part, chaque base de faits (il peut y en avoir plusieurs) contient, entre autres, des *liens*, c'est-à-dire une liste de concepts qui ont quelque chose à voir avec le problème étudié, et une liste de règles, dites *actives*, dans l'ordre de leur activation, selon, en gros, leur type d'abord, leurs liens ensuite, éventuellement réordonnée.

Enfin, on dispose de méta-actions, c'est-à-dire d'un moyen de décrire des actions complexes par des paquets de règles, de la façon suivante :

Pour <méta-action> Règle <nom> si <conditions> alors <actions>
Règle ... si ... alors ...
...

L'activation des règles est faite par la méta-action ACTIVREG qui comporte une seule règle sans condition :

Pour ACTIVREG ACTI0 ACTI1 ACTI2 ... ORDOREG

ACTI0 crée les *liens* (item LIENS) . Ce sont, d'une part, les concepts apparaissant dans l'énoncé du problème, dans les définitions des concepts précédents, etc ... (c'est technique), d'autre part des concepts importants, dans le domaine étudié, pour des raisons mathématiques (c'est heuristique).

Les autres méta-actions ACTIi activent les règles de types Ri liés *d'une certaine façon* aux concepts de l'item LIENS. La façon dont on exprime ce lien n'est pas forcément la même pour tous les types. Une grande souplesse est donc laissée à l'utilisateur.

Enfin, ORDOREG réordonne certaines règles actives :

Pour ORDOREG si Rl et R2 sont deux règles actives,
R1 *est susceptible de créer a tel que P,*
R2 *est susceptible de créer b tel que Q,*
P est plus général que Q,
alors R2 avant Rl.

A quoi correspondent les types et leur ordre ? Actuellement, c'est l'utilisateur qui décide. Dans certains cas, on pourrait certainement trouver des raisons générales, et travailler avec des métarègles. Dans d'autres cas, les raisons sont assez empiriques, ou découlent d'un certain apprentissage. Il s'agit alors essentiellement de mathématiques.

8.2. Types de règles

Dans les expérimentations actuelles, les types ont été les suivants, dans l'ordre de l'activation des règles correspondantes :

8.2.l. Le type RO (Règles Obligatoires) comporte des règles non dangereuses, c'est-à-dire ne modifiant rien d'important. Elles peuvent, par exemple, être de simples réécritures, ou ajouter des informations d'une manière linéaire, non expansive.

Elles peuvent être données, ou construites par le système à partir des définitions ; par exemple, la règle :

si on a A étoilé par rapport à x, $x \in A$, $t=[x\ y]$, *alors on a* $t \subset A$

provient de la définition:

A étoilé par rapport à x $\Leftrightarrow \forall y \in A\ \ [x\ y] \subset A$

Ce type peut contenir aussi des connaissances simples sur le domaine (par exemple *l'addition est continue*) ou des *savoir-faire* non dangereux, soit par leurs conditions très sélectives, soit par leurs actions non dangereuses.

8.2.2. Quatre types de règles sont à utiliser assez prudemment, car elles modifient la base de faits un peu plus sérieusement que les précédentes, ou bien créent des éléments avec un risque d'expansivité. Ce sont le remplacement de la conclusion par sa définition (deux cas suivant que c'est un prédicat ou un symbole fonctionnel), les traitements des hypothèses existentielles et disjonctives.

L'ordre respectif de ces quatre types est hérité d'un précédent système dans lequel ces actions étaient programmées [PAS76/78], et est purement empirique.

8.2.3. Des paquets de *savoir-faire* de types SAV-FAI i sont à utiliser extrêmement prudemment.

On y trouve des heuristiques et des outils mathématiques, des cas particuliers de propriétés plus générales. Par exemple, le segment [0 x] est lié à la multiplication $*$; tout élément de ce segment est de la forme $\lambda * x$ avec $\lambda \in [0\ 1]$; le segment [x y], par contre, si x et y ne sont pas 0, est plutôt lié à l'addition + ; tout élément de. ce segment est somme de deux éléments appartenant respectivement à [0 x] et [0 y]. On retrouve ici une idée déjà exprimée à propos de AM (voir §7.2.5)

On peut y trouver aussi des propriétés générales, mais exprimées dans le langage de règles, directement utilisables par le moteur, par exemple des propriétés sur la multiplication ou l'inverse.

On y exprime également, combinées avec des heuristiques, des propriétés ou des méthodes provenant de définitions, d'axiomes, ou de *résultats connus*. On verra plus loin quelques exemples de règles *données* qu'il serait souhaitable de *construire*.

8.2.4. Enfin, le type GESTION comporte des éclatements, des créations, ou d'autres actions très compliquées.

8.3. Général ou particulier

Contrairement à EURISKO qui crée des heuristiques, en particularise ou généralise, MUSCADET n'en est pas capable. Ceci est dû à la difficulté du domaine qui est loin d'être complètement maitrisé, et à une différence d'objectif et de méthodologie. Dans EURISKO, on travaille sur les heuristiques et on examine les résultats que l'on peut en obtenir. Dans MUSCADET, on s'intéresse à un domaine mathématique et on cherche quelles heuristiques y sont pertinentes, quelles heuristiques les mathématiciens utilisent.

Ainsi, la formulation de la métarègle ORDOREG qui est donnée au système n'est pas celle du §8.l., où les propriétés écrites en italique sont très générales, et que le moteur ne saurait pas interpréter, mais la suivante :

si Rl et R2 sont deux règles actives,
Rl *contient une action* AJHYP $\exists x \in$ A P
R2 *contient une action* AJHYP $\exists x' \in$ A Q
Q contient une conjonction de termes dont l'un est égal à P modulo x et x'
alors R2 avant Rl

L'avantage de cette formulation, outre qu'elle s'exprime dans le langage, est qu'elle est plus rapide à vérifier. Etant donnés les choix (heuristiques ou méthodologiques) de représentations et d'expressions qui ont été faits par ailleurs, cette formulation est tout à fait suffisante. Néanmoins, il serait souhaitable qu'un système la construise à partir de la formulation générale.

8.4. Construction de règles

C'est un des points essentiels du système. Il n'était pas question de *donner* au système toutes les règles de type RO qui peuvent se déduire des définitions (ainsi que des hypothèses universelles), par exemple celle du §8.2.l. sur les ensembles étoilés.

Des métarègles construisent ces règles. Mais leur nombre est important et leur expression est

très complexe et pas très lisible. C'est sans doute parce que le langage n'est pas adapté à ce genre de travail.

En fait, les métaheuristiques ne résident pas dans ces métarègles, mais dans la forme des règles que l'on désire obtenir à partir des énoncés, les métarègles ne font qu'exprimer des techniques pour obtenir cette forme.

Il serait souhaitable de ne donner au système que les contraintes sur la forme des règles que l'on désire obtenir et qu'alors il construise lui-même ces métarègles, ou utilise d'autres méthodes pour construire ces règles.

8.5. Règles de *savoir-faire*

Certaines règles données comme savoir-faire mathématiques pourraient être construites par le système.

8.5.1. Construite par le système à partir de la définition de *étoilé*, la règle de type RO :

si on a A étoilé par rapport à x
$y \in A$
$t = [x\ y]$
alors ajoute $t \subset A$

est non dangereuse et utilisée fréquemment. Elle est appliquée si on a déjà introduit le segment [x y]. Néanmoins, la règle suivante, qui crée ce segment, est quelquefois indispensable :

si on a A étoilé par rapport à x
$y \in A$
$x \neq y$ (c'est-à-dire x non identique à y mais ils peuvent être égaux)
alors créer le segment $t = [x\ y] \subset A$

Actuellement donnée comme savoir-faire, elle pourrait être construite systématiquement par une technique qui créerait ce genre de règles à chaque fois qu'une définition comporte une propriété existentielle ou un symbole fonctionnel, c'est-à-dire qui garderait comme règle de *savoir-faire*, à utiliser parfois et avec prudence, l'expression intermédiaire exprimant une création.

8.5.2. Des résultats connus simples sont exprimés par un mathématicien sous forme d'un énoncé *déclaratif* simple. Ici, pour ces *résultats connus*, l'utilisateur doit fournir :
- une règle dans le langage de règles (c'est-à-dire des actions, donc ce n'est pas vraiment déclaratif),
- éventuellement des conditions d'application (ou plutôt de non application),
- éventuellement des actions de désactivation (d'elle-même et peut-être de règles relatives à certains concepts),
- un type (RO ou SAVOIR-FAIRE).

Par exemple, pour exprimer que l'inverse d'une translation est la translation opposée, soit :

$f_a^{-1} = f_{-a}$ avec $f_a : x \rightarrow a+x$

on écrit la règle de type SAVOIR-FAIRE :

si on a f = projection de + par rapport à a
g = inverse(f)
on n'a pas b=-a
alors créer b, ajouter b=-a et g = projection de + par rapport à b

Il serait évidemment souhaitable que le système ne reçoive que la connaissance purement déclarative et non la règle constructive.

Un autre exemple de ce type est la méthode, issue de la propriété *toute union d'ouverts est*

un ouvert, c'est-à-dire

$$\forall i \in I\ (A_i \text{ ouvert}) \Rightarrow (\cup_{i\in I} A_i) \text{ ouvert}$$

que l'on exprime en disant que, si on veut montrer qu'une union d'ensembles est un ouvert, il *suffit* de montrer que chaque ensemble est ouvert:

si la conclusion est (B ouvert)
on a l'hypothèse $B = \cup_{i\in I} A_i$
alors remplacer la conclusion par $\forall i \in I\ (A_i \text{ ouvert})$

8.5.3. Parmi les *résultats connus*, certains conduisent à ce que l'on pourrait qualifier de *méthodes*.

Le théorème suivant est démontre par le système :
Si U est un voisinage de O,
alors il existe un voisinage V de O, inclus dans U, étoilé et symétrique par rapport à O.
Puis il est intégré comme *résultat connu* pour la poursuite du travail, non sous son expression formelle :

$$U \text{ voisinage de } O \Rightarrow \exists V\ [O \in V \wedge V \subset U \wedge V \text{ voisinage de } O \wedge V \text{ étoile par rapport à } O \wedge V \text{ symétrique}]$$

mais par la règle suivante :

si on a U voisinage de O
on cherche V tel que, entre autres, V voisinage de O
(c'est-à-dire la conclusion est une conjonction et contient la propriété *V voisinage de O*)
U n'a pas de sous-ensemble
alors créer W contenant O
inclus dans U
voisinage de O
étoilé symétrique par rapport à O
désactiver cette règle.

Ceci fournit une méthode (règle de type SAVOIR-FAIRE) pour construire des voisinages, qui sont des intermédiaires indispensables dans les démonstrations de certains théorèmes du style :
Si U est un voisinage de O, alors il existe un voisinage V de O, inclus dans U, tel que ... .

Ce qui est important, c'est qu'on cherche, à partir d'un voisinage, à en construire un plus petit (au sens de l'inclusion), ayant certaines propriétés. On a plusieurs méthodes qui, à partir d'un voisinage, en construisent de plus petits. L'idée (consciente ou inconsciente) est alors de construire une suite descendante de voisinages pour arriver à celui ayant la propriété cherchée.

Le fait d'être un voisinage n'est plus important à ce niveau, il pourrait être remplacé par n'importe quelle propriété. C'est la construction descendante qui est importante.

On pourrait donc imaginer, pour construire la règle précédente à partir du *résultat connu* formel, la métarègle générale suivante:

si on a P(U) ,
on cherche $V \subset U$ *tel que* $P(V) \wedge Q1(V)$
on a le résultat connu $P(U) \Rightarrow \exists V \subset U\ (P(V) \wedge Q2(V))$
alors construire la règle:
si on a P(U)
on cherche $V \subset U$ *tel que* $P(V) \wedge \ldots$
U n'a pas de sous-ensemble (à préciser, peut-être)
alors créer $W \subset U$ *tel que* $P(W) \wedge Q2(W)$
se désactiver (éventuellement)

8.5.4. Des *méthodes* sont également issues des définitions. Mais un phénomène humain se produit ici, sans doute parce que ces méthodes sont très souvent employées : les mathématiciens semblent avoir en tête, non une règle générale, mais des règles particulières aux situations données, comme s'ils avaient inconsciemment court-circuité les intermédiaires.

C'est le cas de l'application de la continuité de + et $*$ dans un Espace Vectoriel Topologique. La règle générale est la suivante:

si f est une application continue de A dans B
U est un voisinage de y=f(x)
alors il existe un voisinage V de y dans A tel que $f(V) \subset U$

Si A est le produit de deux ensembles, soit Al×A2, V est le produit de deux voisinages Vl dans Al et V2 dans A2.

Les règles particulières, pour un Espace Vectoriel Topologique E sont :
- pour la continuité de $*$ en O : *si U est un voisinage de O, alors il existe* $\alpha \in \mathbb{R}$, *V voisinage de O, tels que* $]\alpha, \alpha[* V \subset U$
- pour la continuité de + en O : *si U est un voisinage de O, alors il existe V voisinage de O tel que* $V+V \subset U$

Cet exemple est commenté en détails dans [PAS84].

Ici, j'ai choisi de ne pas donner les règles particulières. Par une cascade de règles générales, de méta-actions, et d'heuristiques, on aboutit aux créations particulières sans s'être encombré d'objets inutiles qui auraient pu engendrer un nombre catastrophique d'autres objets.

Parmi les heuristiques mathématiques, citons :
- le choix de ne travailler que sur des voisinages, et même sur des bases de voisinages, choix suscité à partir des dialogues avec des mathématiciens ;
- l'utilisation des propriétés de l'ensemble des réels $\mathbb{R}$;
- le fait que les doublets (V, V) de voisinages V de x forment une base de voisinages pour les éléments (x, x) de la diagonale (cette propriété, précisée, à ma demande, par les mathématiciens, rejoint l'heuristique utilisée par EURISKO et analysée au §7.2.2.).

Il y a de plus quelques conditions filtres qui pourraient peut-être être générées automatiquement.

8.5.5. Il arrive que le système doive choisir entre deux méthodes proches dont les conditions d'application sont a priori les mêmes. La encore, ce problème est résolu par des conditions filtres établies d'une manière empirique ou inspirées des propos des mathématiciens, ou bien par des paquets *ordonnés* de SAVOIR-FAIRE.

Par exemple, pour montrer qu'un ensemble est un ouvert, on peut, entre autres, montrer que c'est une union d'ouverts, ou bien montrer que c'est l'image réciproque d'un ouvert par une application continue.

On a aussi des règles pour *forcer* l'union si on a choisi la première méthode, par exemple remplacer f(A×B) par $\cup_{x \in A} f_x(B)$ où f_x est la projection de f par rapport à x, soit $f_x : y \rightarrow f(x,y)$. Remarquons que ceci pourrait se déduire de la propriété $f(A) = \cup_{x \in A} f_x(B)$ qui est plus générale, mais sans intérêt direct.

9. Conclusion

Cette analyse de quelques systèmes faisant des mathématiques et utilisant un métaniveau a fait apparaitre un certain nombre d'analogies, mais aussi de différences entre les travaux. Il n'est certainement pas possible d'en tirer des conclusions générales globales.

On a opposé deux façons de faire des mathématiques : la découverte et la démonstration. En découverte, l'affirmation selon laquelle un système découvre à partir de rien peut être contestée. Les comportements des deux systèmes étudiés comportent des analogies. Par une sorte d'apprentissage, ces systèmes tournent et prennent des décisions en fonction des résultats obtenus. Mais, alors que le Démonstrateur de Théorèmes de J.Pitrat part plutôt du niveau méta pour en descendre, AM/EURISKO de D.B.Lenat part plutôt de la base pour générer des concepts plus élevés. Dans les deux cas, les résultats sont meilleurs si on donne, bien qu'on puisse s'en passer, soit des métamétathéorèmes, soit des métaheuristiques, que les systèmes pourraient trouver tous seuls. Dans les deux cas également, après de bons résultats, il y a saturation et les systèmes tournent en rond.

Les terminologies employées dans les différents systèmes ne désignent pas toujours les mêmes notions. Par exemple, il n'est même pas sûr que le préfixe *méta* désigne la même chose partout. En tout cas, les façons de le considérer sont variées.

Les différents niveaux méta peuvent être soit strictement séparés (D.T.), soit tout mélangés (AM/EURISKO, MUSCADET), le moteur ignorant à quel niveau il travaille, mais alors des surprises peuvent arriver (EURISKO séparant de lui-même le niveau méta).

On a d'autre part constaté une analogie entre les *métathéorèmes* de D.T. et les *nouveaux concepts* de AM.

Alors que les connaissances dans D.T. et AM/EURISKO sont toujours réellement déclaratives, il arrive que dans MUSCADET, elles aient dû être traduites dans le langage compris par le moteur d'une manière plus constructive, exprimant ainsi plus des méthodes que des connaissances. Il serait souhaitable que le système lui-même fasse ce travail. D'autre part, contrairement aux précédents, MUSCADET ne fait pas d'apprentissage. Les heuristiques utilisées sont soit générales, soit mathématiques mais générales pour les mathématiques, soit particulières aux domaines mathématiques étudiés. Si elles se justifient parfois mathématiquement, il arrive qu'elles soient purement empiriques, ou bien traduisent des heuristiques que les mathématiciens possèdent sans toujours pouvoir les expliquer consciemment.

D'autre part, on souhaiterait plus de lisibilité dans les bases de connaissances.

Enfin, il faudrait introduire un niveau *métaméta* et … supprimer le moteur.

Bibliographie

[LEN76] - D.B.LENAT, An artificial intelligence approach to discovery in mathematics as heuristic search, SAIL AIM-286. A.I.Lab. Stanford University (1976)

[LEN77] - D.B.LENAT, Automated theory formation in mathematics, IJCAI, Cambridge (1977), 833-842

[LEN78] -. D.B.LENAT, The ubiquity of discovery, Artificial intelligence 9(1978), 257-285

[LEN82a] - D.B.LENAT, The nature of heuristics, Artificial intelligence 19(1982), 189-249

[LEN82b] - R.DAVIS, D.B.LENAT, Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, 1982

[LEN83a] - D.B.LENAT - Theory formation by heuristic search: the nature of heuristics II, background and examples, Artificial intelligence 21(1983), 31-59

[LEN83b] - D.B.LENAT, EURISKO: a program that learns new heuristics and domain concepts, Artificial Intelligence 21(1983), 61-98

[PAS76] - D.PASTRE, Démonstration automatique de théorèmes en théorie des ensembles, Thèse de 3eme cycle, Paris VI(1976)

[PAS78] - D.PASTRE, Automatic theorem proving in set theory, Artificial intelligence 10(1978),

1-27
[PAS81] - D.PASTRE, Présentation de la thèse de D. B. Lenat : AM: An Artificial Approach to Discovery in Mathematics as Heuristic Search, Colloque Intelligence Artificielle, Toulouse, Juillet 1981, Publications du Laboratoire C.F. Picard, n°24
[PAS84] - D.PASTRE, MUSCADET: Un système de Démonstration Automatique de Théorèmes utilisant Connaissances et Métaconnaissances en Mathématiques, Thèse d'état, Paris VI, Novembre 84
[PAS85a] - D.PASTRE, Expérimentation d'une Expertise mise au point par observation du comportement de l'homme, COGNITIVA 85, Paris, Juin 1985
[PAS85b] - D.PASTRE, Quelques exemples d'utilisation de Métarègles en Démonstration Automatique de Théorèmes, Congrès AFCET RFIA, Grenoble, Novembre 1985
[PlT66] - J. PITRAT, Réalisation de programmes de démonstration de théorèmes utilisant des méthodes heuristiques, Thèse d'état, Paris (1966)
[PIT70] - J.PITRAT, Un programme de démonstration de théorèmes, Monographie AFCET, Dunod (1970)
[STA85] - B.STARYNKEVlTCH, Le langage RLL-1, Colloque Intelligence Artificielle, Toulouse, Septembre 1985, Publications du Laboratoire C.F.Picard, n°58
[VIV83] - M.VIVET, Un exemple d'utilisation de métarègles : la sélection des plans dans CAMELIA, Colloque intelligence Artificielle, Chambéry (1983), Publications du Laboratoire C.F. Picard, n°38, 339-363
[VIV84] - M.VIVET, Expertise mathématique et informatique : CAMELIA, un logiciel pour raisonner et calculer, thèse d'état, Paris VI, juin 1984
[VIV85] - M.VIVET, Regards différents autour de CAMELIA, Colloque Intelligence Artificielle, Aix-en-Provence, Septembre 1984, Publications du Laboratoire C.F. Picard, n°49