SOMMAIRE

Introduction

L'équipe Métaconnaissance du LIP6 et plusieurs chercheurs d'EDF se sont réunis dans l'île de Berder du 22 au 24septembre 1999. Ce rapport contient les textes de quelques unes des interventions qui y ont été faites.

Un concept "méta" important est apparu directement ou indirectement dans de nombreux papiers : c'est le concept de monitoring où le système examine ce qu'il fait pour mieux le faire ensuite. Après une présentation générale de l'intérêt et des problèmes soulevés par le monitoring, nous allons le retrouver plusieurs fois. Un outil indispensable pour mettre en œuvre un monitoring est d'avoir un métamoteur : il permet de savoir ce qui se passe et de changer éventuellement la stratégie d'utilisation des connaissances. Le métamoteur de SEPIAR2 peut même faire des expériences en cours de résolution afin de déterminer dynamiquement les meilleurs paramètres pour traiter l'expertise de base utilisée ; les résultats obtenus montrent effectivement une amélioration des performances. La structure de métamoteur également adoptée pour IDRES permet aussi d'avoir des métarègles d'introspection qui permet au système de savoir ce qu'il fait et pourquoi il le fait, donc de changer dynamiquement de stratégie de résolution. Pour le système MARECHAL, le monitoring est au cœur du projet, puisqu'il s'agit, dans une application à des wargames, d'arbitrer les conflits qui peuvent venir de ce que plusieurs méthodes peuvent être utilisables au même moment ; il faut analyser ce qu'elles donnent pour en déduire une solution globale.
Le monitoring est apparu de façon moins centrale dans plusieurs autres papiers. Par exemple, GENSAM doit généraliser à partir de petits échantillons. Il existe de nombreuses méthodes qui permettent de le faire, chacune ayant ses avantages et ses inconvénients qu'il faut balancer dans chaque situation. La métaexpertise de contrôle donnée dans la section 5.2 est un excellent exemple d'expertise de monitoring. Dans l'annexe 6 de son article, D. Pastre donne un exemple du monitoring qu'elle a fait au cours de sa recherche de la démonstration d'un théorème. Par ailleurs, l'analyse des résultats du tournoi de Phutball montre des erreurs des programmes dues à une insuffisance du monitoring, par exemple quand ils ne voient pas qu'ils se retrouvent après chaque coup dans une position quasi-identique à la précédente, mais un peu plus défavorable.

Plusieurs papiers présentent des travaux en cours de réalisation dans l'équipe. D'abord deux applications à la démonstration de théorèmes. Nous avons une analyse des résultats obtenus par MUSCADET à la compétition de démonstrateurs organisée dans le cadre des conférences CADE ; nous voyons aussi l'enrichissement du système pour mieux résoudre les problèmes de la TPTP Problem Library. A mi-chemin entre la démonstration de théorèmes et l'EIAO, le système de J.-P. Spagnol devra être capable de produire des démonstrations adaptées au niveau des élèves. Enfin, un système d'EIAO, comme COMBIEN ?, se doit d'avoir des interfaces bien adaptées aux divers dialogues que le système aura avec l'élève.
Un problème délicat, mais qui se présente souvent, est la généralisation à partir d'un nombre restreint d'observations ; le système GENSAM en expérimente des solutions dans le domaine médical. Le projet CASSICE reconnaît automatiquement les manœuvres exécutées par le conducteur d'un véhicule. Naturellement, les jeu sont toujours un terrain de prédilection pour l'IA. Trop souvent, on a étudié les jeux à information complète où de nombreuses difficultés n'apparaissent pas. C'est pourquoi F. Kocik prend comme sujet d'étude les jeux de carte, mais naturellement dans le cadre de systèmes généraux qui reçoivent comme donnée les règles du jeu particulier auquel ils doivent jouer. Enfin, T. Cazenave fait le compte-rendu de l'expérience qu'il a organisée ; nous y avons tous pris part en créant dans un temps limité un programme pour jouer à un jeu que nous ne connaissions pas : le football des philosophes, appelé aussi Phutball.

Nous remercions Fabrice Kocik et Tristan Pannérec pour avoir si parfaitement préparé l'organisation matérielle de ce colloque, qui a pu avoir lieu grâce à un contrat EDF. Ils ont également pris en charge l'édition de ce rapport.

Jacques Pitrat

Monitorer la recherche d'une solution

Jacques Pitrat
CNRS–LIP6
Université Pierre et Marie Curie

Résumé : Quand un sujet monitore la recherche d'une solution, il se place au niveau méta : il examine ce qu'il est en train de faire pour mieux diriger cette recherche. A partir de l'examen de la résolution d'un problème nouveau et difficile, ce papier montre l'importance du monitoring ; il met également en évidence la variété des méthodes utilisées et la difficulté de les implémenter dans un système d'IA. Nous verrons enfin qu'il ne suffit pas de monitorer la résolution d'un problème, il faut aussi savoir monitorer son monitoring.

Mots-clés : monitoring, résolution de problèmes, métaconnaissance

1. Introduction

Pendant qu'un sujet humain cherche à résoudre un problème, il s'observe souvent. Il tire des conséquences de ses observations pour prendre ensuite des décisions. Ce processus est plus efficace si le sujet a fait des prévisions sur ce qu'il allait obtenir ; il lui arrive alors d'être surpris, surprise qu'il analyse pour mieux comprendre le problème. Cette activité est un dialogue entre le niveau de base où l'on tente de résoudre le problème et le niveau méta où l'on examine comment se passe la résolution du problème. Cet examen entraîne souvent un changement de comportement au niveau de base.
Le monitoring est particulièrement utile pour :

* Bien choisir les essais initiaux. Avant de se lancer dans une voie, on examine les diverses possibilités qui existent ; on choisit celle qui est la plus prometteuse. Naturellement, en faisant cette analyse, on peut souvent faire des prédictions sur ce qui devrait se passer.

* Eviter de s'attarder dans une voie qui était prometteuse et se révèle décevante. En particulier, si l'on a fait des prédictions dans la phase précédente, on compare ce qui a été obtenu et ce que l'on pensait pouvoir obtenir. Si le désaccord est trop important, on peut abandonner.

* Corriger la direction prise initialement. Comme dans le cas précédent, les résultats ne sont pas à la hauteur des prédictions. Mais, en analysant les raisons de ce désaccord, on voit une possibilité de rectifier le tir.

* Trouver des erreurs éventuelles. C'est important pour nous humains qui faisons constamment des erreurs. Un des avantages du monitoring est de nous permettre d'arriver à des résultats corrects malgré nos erreurs. Pour cela, nous examinons avec soin les résultats obtenus et réagissons quand ils ne correspondent pas à ce qu'il était raisonnable d'obtenir.

* Acquérir une expertise. L'analyse des essais fructueux et infructueux conduit à un apprentissage. On apprend par exemple quelles méthodes réussissent ou échouent selon les situations, quelles caractéristiques sont importantes pour justement choisir les bonnes méthodes. Après plusieurs résolutions faites avec monitoring, on a tous les éléments pour construire une expertise qui détermine pour chaque situation les méthodes qui ont de grandes chances de conduire au résultat.

* Etre autonome. Un système autonome risque de dériver vers des maxima secondaires et d'y rester englué. Le monitoring peut déceler de telles éventualités et sortir le système de situations de blocage.

Nous commencerons par présenter deux études psychologiques qui montrent l'importance du monitoring chez les humains. Nous verrons ensuite l'énoncé d'un problème que j'ai résolu en observant le monitoring que j'utilisais ; il servira d'exemple pour montrer les avantages de monitorer la recherche d'une solution. Nous examinerons quelques aspects caractéristiques du monitoring et les raisons pour lesquelles il est utile. Nous parlerons enfin de quelques systèmes d'IA dont les performances ont été améliorées par le monitoring.

2. Deux études psychologiques sur le monitoring

Quelques psychologues ont été amenés à étudier comment les êtres humains tiraient parti du monitoring pendant qu'ils cherchaient la solution d'un problème. Nous allons présenter brièvement deux expériences dont le but était complètement différent. Dans la première, le monitoring a été utilisé par un non-expert pour diriger les essais qu'il faisait alors que dans la deuxième il a permis à certains sujets de trouver la solution bien qu'ils aient fait des erreurs.

2.1. Bien choisir ses essais

Alan Schoenfeld [Schoenfeld 1985] a étudié des humains en train de résoudre des problèmes. Il proposait un problème de mathématiques à ses sujets qui devaient penser à voix haute. Par exemple, il leur demandait de construire une parallèle à la base d'un triangle donné de telle façon qu'elle partage ce triangle en deux surfaces de même aire. Il a observé trois types de comportement :

1. L'expert du domaine va directement à la solution. Son comportement est impressionnant : il examine le problème, voit immédiatement ce qu'il faut faire et trouve la solution à la suite d'une série de décisions qui sont toutes bonnes. Pour lui, le monitoring n'est pas nécessaire, il sait dès le début le plan qu'il met ensuite en œuvre. Malheureusement, il n'aura certainement pas ce comportement dans un domaine où il n'est pas encore expert.

2. La spécialité du novice est la "chasse aux oies sauvages". Il part sans réfléchir sur la première idée qui lui passe par la tête, il la poursuit très longtemps sans se poser de questions sur son bien fondé. Enfin, il se lasse et il l'abandonne sans davantage de raisons, parfois au moment où elle allait aboutir, pour une autre idée qui lui es venue et qu'il n'examine pas davantage. Naturellement, ses performances sont faibles.

3. Le métaexpert n'est pas un expert du domaine, où il en sait souvent moins que le novice ; par contre, il est expert en résolution de problèmes. Il commence par analyser le problème, choisit la plus prometteuse des voies qu'il a envisagées, vérifie qu'elle se développe bien comme prévu. Si les résultats ne sont pas conformes aux prévisions, il essaye de comprendre pourquoi et, soit continue dans la même direction après une éventuelle correction, soit repart dans une direction nouvelle. Le métaexpert n'est pas aussi rapide que l'expert ; toutefois, grâce à son excellent monitoring de la recherche de la solution, il est capable de résoudre des problèmes dans une bien plus grande variété de domaines.

L'expert étudié par Schoenfeld considérait de haut le métaexpert, mais il avait tort. La comparaison aurait sans doute tourné à son désavantage si on leur avait proposé un problème dans un domaine inconnu des deux. De plus, l'expert avait eu sans doute un comportement analogue avant de devenir expert ; ce comportement lui avait justement permis de devenir expert.

2.2. Trouver ses erreurs

Carl Allwood [Allwood 1984] a étudié comment des étudiants trouvent leurs erreurs pendant qu'ils résolvent des problèmes de statistiques. L'erreur est partout présente dans le comportement humain, même des experts commettent des erreurs que l'on croirait réservées à des débutants. Toutefois, une des supériorités des experts vient de ce qu'ils sont capables de trouver et de corriger leurs erreurs. Là encore, l'auteur a montré l'importance du monitoring dans le processus de recherche d'erreurs. Il a observé quatre comportements liés à l'erreur dans les protocoles :

1. L'évaluation affirmative. Le sujet s'arrête pour s'assurer que ce qu'il trouve correspond à ce qu'il attendait : " 56 sur 350, oui, cela paraît raisonnable".

2. La vérification standard. Après avoir fait une opération qu'il maîtrise mal, le sujet fait automatiquement une opération de vérification qui ne le fait pas avancer vers la solution, mais qui l'assure qu'il n'a pas commis d'erreur dans une étape où il sait qu'il se trompe souvent. Par exemple, on fait une preuve par 9 après une multiplication ou l'on fait l'addition inverse après une soustraction.

3. La création directe d'hypothèse d'erreur. Le sujet dit qu'il a fait une erreur et la corrige immédiatement : "Y=50 plus, cela devrait être moins…".

4. La suspicion d'erreur. Quand nous avons un résultat qui ne correspond pas à nos attentes, nous supposons qu'il y a une erreur et nous commençons à la rechercher. Il est important d'être vigilant, car plus on est proche de l'erreur et plus on a de chance de la découvrir. Beaucoup de résultats peuvent nous amener dans un état de suspicion d'erreur. Cela arrive bien évidemment si l'on trouve une probabilité supérieure à 1, mais il y a bien d'autres déclencheurs. Allwood indique le cas d'un sujet qui avait lancé une recherche d'erreur parce qu'il avait trouvé un résultat égal à 1666, les trois derniers chiffres égaux lui paraissant louches. Il arrive que l'on parte en suspicion d'erreur pour une fausse raison et l'on perd beaucoup de temps, car il n'est rien de plus difficile que de trouver une erreur qui n'existe pas. Cela arrive communément à l'informaticien ! Allwood estime que, pour les sujets qu'il a observés, la suspicion d'erreur correspond à une erreur réelle dans les deux tiers des cas.

L'activité de recherche d'erreur, qui est si importante chez les humains, est un peu négligée en IA où l'on suppose trop souvent que le système ne fait pas d'erreur et que ses connaissances sont correctes.

3. Une expérience de monitoring

Je vais illustrer mon propos sur le monitoring par une expérience que j'ai faite en m'observant en train de résoudre un problème. Je ne donne pas tous les détails qui figurent dans [Pitrat 1999] ; je commence ici par présenter le problème.

J'avais écrit un programme [Pitrat 1996] qui trouve des phrases réflexives, c'est à dire des phrases qui décrivent le nombre des lettres qu'elles contiennent. Le titre de l'article qui le décrit est une phrase de ce type. La méthode utilisée, dérivée d'une idée due à D. Hofstadter [Hofstadter 1985], est évolutive. A un certain moment, la phrase n'est pas une solution parce que, pour au moins une lettre, le nombre d'occurrences qu'elle affiche, que nous appellerons le nombre "affiché", ne correspond pas au nombre réel d'occurrences qu'elle contient, que nous appellerons le nombre "réel". Par exemple, elle dit qu'elle contient vingt-trois I alors qu'il y en a vingt-cinq ; le nombre affiché est "vingt-trois" et le nombre réel est "vingt-cinq". Le programme remplace alors vingt-trois par vingt-cinq et continue jusqu'à obtenir une phrase où tous les nombres affichés sont identiques aux nombres réels. Le programme part d'un début de phrase donné appelé "corps" qui correspond à la partie de la phrase qui ne contient pas les nombres. Il se peut très bien qu'il n'existe aucune phrase réflexive que l'on puisse bâtir sur un corps donné. Il se peut aussi qu'il existe une solution, mais que la méthode précédente n'arrive pas à la trouver.

Pour augmenter ses chances de trouver une solution, le programme peut construire un très grand nombre de corps sur un "noyau"qui lui est donné en essayant divers ajouts en différents points de la phrase. Par exemple, si la fin du noyau est "et Z", l'on peut ajouter entre le "et" et le nombre de Z des éléments de phrase comme "pour finir", "enfin", "pour terminer", "à la fin", "en dernier",… Toutes les phrases ainsi créées ont un sens acceptable ; avec plusieurs points où un ajout est permis et plusieurs valeurs pour chacun de ces points, on arrive ainsi à engendrer automatiquement des milliers de corps à partir du même noyau. Comme il existe souvent au moins une solution pour un corps donné (et souvent plusieurs solutions pour le même corps), le programme a ainsi engendré de nombreux corps à partir desquels il a trouvé des milliers de phrases réflexives.

Georges Perec a écrit un roman, intitulé "La disparition", qui a une caractéristique exceptionnelle : il ne contient pas une seule occurrence de la lettre E. D'où l'idée de réaliser une phrase réflexive qui ne contiendrait pas la lettre E. Je me suis posé ce problème et je me suis observé pendant que je le résolvais. Le monitoring a eu manifestement un rôle crucial pendant cette recherche et je prendrai mes exemples dans cette expérience. Je l'ai complétée en essayant de résoudre d'autres problèmes analogues, comme la recherche d'une phrase réflexive sans E en anglais ou la recherche d'une phrase réflexive sans N en français.

Pour construire une phrase réflexive sans E en français, nous ne pouvons utiliser que douze nombres : 1, 3, 5, 6, 8,10, 18, 20, 23, 25, 26 et 28. Le nombre suivant sans E est 1.000.000 ! Ces nombres seront appelés par la suite "permis". Quatorze lettres figurent dans ces nombres : C,D, G, H, I, N, O, Q, R, S, T, U, V, X. Les onze lettres, en omettant le E, qui ne figurent pas dans cette liste seront appelées "stables", car les nombres qui leur sont affectés ne dépendent que du corps et ne changent pas si l'on change les nombres. Notons que le G et le V apparaissent toujours ensemble et seulement dans "vingt" et ses dérivés.

J'ai naturellement commencé par essayer d'utiliser le programme existant ; bien sûr, je ne devais accepter une solution que quand elle disait qu'elle ne contenait aucun E. Malheureusement, cela ne s'est jamais produit, car il restait toujours beaucoup trop de E. De plus, le mécanisme de création automatique des corps s'est révélé inutilisable dans ce contexte : il fallait les engendrer un par un, ce qui est très difficile si l'on s'interdit d'utiliser la lettre E.

Dans un autre essai infructueux, j'ai écrit un programme basé sur un principe complètement différent ; je l'appellerai par la suite le deuxième programme. Ce programme considérait toutes les combinaisons possibles des nombres permis pour les lettres qui ne sont pas stables. La combinatoire n'était pas grande, car on arrivait à se ramener à onze nombres à déterminer, les nombres pour G, I et V pouvant être précisés à l'avance. On pouvait aussi ramener chaque nombre à six valeurs possibles (3, 5, 6, 8, 10, 18). On tenait aussi compte du fait que deux phrases qui ne diffèrent que par une permutation des nombres affectés aux diverses lettres ont le même nombre total de lettres ; il suffit donc de n'engendrer qu'un seul exemplaire de chaque permutation. Avec toutes ces améliorations, nous avons une procédure de décision :le programme donne toujours une réponse : il y a ou non des solutions, et s'il y en a une, il donne les nombres d'occurrences de chaque lettre. Ce programme est très rapide, la réponse est toujours obtenue en moins d'une seconde. Malheureusement, les réponses ont toujours été négatives. Comme un examen de la méthode, plus approfondi mais trop tardif, a montré qu'elle avait très peu de chances de succès, cette voie a été abandonnée.

J'ai fini par trouver la solution en utilisant une méthode qui ne fait pas appel à l'ordinateur. Je commence par fixer a priori les nombres d'occurrences de toutes les lettres qui figureront dans la phrase solution, c'est à dire quels seront les nombres affichés. Je suppose que sont présentes toutes les lettres figurant dans les nombres permis, c'est à dire ceux qui n'ont pas de E. Je décide ensuite quelles lettres stables seront admises, par exemple A et P qui n'apparaissent dans aucun nombre permis et leurs nombres d'occurrences. J'ai bien ainsi choisi les nombres affichés de toutes les lettres. Leurs nombres d'occurrences réelles sont partiellement connus : on sait combien de fois elles apparaissent dans les nombres précédents et on sait de plus qu'elles apparaissent une fois pour indiquer leur nombre. Par différence entre les nombres affichés, qui devront être aussi réels, et les nombres ainsi calculés, on a le nombre d'occurrences de chaque lettre pour la partie du corps qu'il reste à déterminer. Au lieu de partir du corps et de trouver les nombres affectés aux lettres, je pars des nombres affectés aux lettres et je détermine les lettres composant le corps. Je cherche enfin un anagramme de ces lettres qui devra avoir un sens acceptable.

Supposons que le choix des nombres d'occurrences des lettres dans la phrase que l'on va construire soit le suivant :

A	8
C	5
D	6
G	3
H	6
I	25
L	5
N	10
O	8
Q	5
R	10
S	10
T	18
U	10
V	3
X	8

J'ai choisi d'utiliser deux lettres qui apparaissent dans aucun nombre sans E : A et L. Il y a donc au total 16 nombres qui se répartissent ainsi (en décomposant "dix-huit" en "dix" et "huit" et "vingt-cinq" en "vingt" et "cinq") : 2 "trois", 4 "cinq", 2 "six", 4 "huit", 5 "dix" et 1 "vingt". J'ai cumulé les occurrences des lettres de ces nombres et je leur ai ajouté une unité pour tenir compte de ce que chaque nombre est suivi de la lettre qu'il représente. J'ai obtenu ainsi les nombres d'occurrences suivants pour la partie de la phrase qui a été déjà fixée :

A	1
C	5
D	6
G	2
H	5
I	19
L	1
N	6
O	3
Q	5
R	3
S	5
T	8
U	5
V	2
X	8

Par différence entre les deux tableaux, j'obtiens le nombre de lettres que je dois utiliser pour construire le début de la phrase réflexive :

A	7
G	1
H	1
I	6
L	4
N	4
O	5
R	7
S	5
T	10
U	5
V	1

J'ai maintenant dû faire un anagramme géant ;il ne s'agit pas seulement de trouver un mot, mais un ensemble de mots qui ait un sens et qui respecte les règles de la syntaxe en utilisant toutes les lettres précédentes. Voici la phrase réflexive sans E que j'ai trouvée à partir des choix précédents :

> Ha, l'on trouva la solution ; il rit, titra tout, sans tri, sur un trait gras : dix N, cinq Q, vingt-cinq I, trois V, cinq C, trois G, dix S, huit X, six H, dix R, cinq L, dix-huit T, six D, huit A, dix U, huit O.

4. Quelques caractéristiques du monitoring

Nous allons examiner quelques caractéristiques du monitoring en les illustrant par des exemples qui sont apparus lors de la solution du problème précédent.

4.1. Découverte de métathéorèmes

De nouveaux moyens de résoudre le problème peuvent se révéler très utiles ; les méthodes ainsi trouvées sont rigoureuses. Nous prouvons des théorèmes sur le problème qui sont donc des métathéorèmes.

Le principal métathéorème dans le cas de la phrase réflexive est :

> On ne change pas le nombre de lettres d'une phrase réflexive en permutant les nombres affectés aux chiffres.

C'est une évidence. Si l'on dit qu'il y a "cinq C, huit G" ou "huit C, cinq G", les nombres affichés sont différents, mais les nombres réels sont bien sûr les mêmes, car on a dans les deux cas une occurrence de "huit" et une de "cinq".Ce métathéorème est intervenu constamment au cours de la recherche de phrases réflexives. Par exemple, on peut très bien trouver une phrase qui n'est pas correcte parce que les nombres réels de C et de G, qui sont 5 et 8, ne sont pas égaux aux nombres affichés qui sont respectivement 8 et 5. Le premier programme que j'avais réalisé pour trouver des phrases réflexives faisait cette vérification et trouvait une solution immédiatement en permutant les nombres de C et de G.

Une autre utilisation est intervenue dans la découverte de la phrase sans E. Nous avons vu que l'on déterminait les nombres d'occurrences des lettres dans le corps. D'après le métathéorème précédent, on peut les permuter. Dans le dernier tableau donné plus haut, il n'y avait aucun C, mais un H. Comme la lettre H est plus difficile à placer que le C, on peut permuter le nombre de C et de H dans les lettres affichées, c'est à dire avoir "six C, cinq H" au lieu de "cinq C, six H". On peut ainsi obtenir directement une solution, ou plus facilement corriger la solution précédente en enlevant un H et en ajoutant un C. Cela est facile en remplaçant le "Ha" initial par "ça", ce qui donne une autre solution :

> L'on trouva la solution ; il rit, titra tout ça, sans tri, sur un trait gras : dix N, cinq Q, vingt-cinq I, trois V, six C, trois G, dix S, huit X, cinq H, dix R, cinq L, dix-huit T, six D, huit A, dix U, huit O.

Ce métathéorème facilite la découverte d'une solution en nous permettant :

* De partir d'un ensemble de lettres plus favorable pour trouver un anagramme.

* De modifier encours de route la répartition des lettres quand on voit qu'il vaudrait mieux avoir plus d'occurrences d'une lettre et moins d'une autre.

* De créer une solution à partir d'une solution déjà obtenue.

J'ai établi d'autres métathéorèmes qui ont toutefois eu moins d'importance. Par exemple, si toutes les lettres de l'alphabet ne figurent pas, il est toujours possible d'obtenir une solution à partir d'une phrase qui serait une solution si le nombre de U et de N réels était augmenté d'une unité en ajoutant un exemplaire de cette lettre, par exemple "un W" si le W n'était pas présent. Après cet addition, on aura bien utilisé le U et le N de trop et le nombre de W est correct.

4.2. Utiliser les métathéorèmes

Il ne suffit pas de connaître un métathéorème pour bien l'utiliser. L'expérience que j'ai faite m'a montré que je le faisais doublement mal.

D'abord, je n'ai pas toujours pensé à me servir d'un métathéorème connu quand j'en aurais eu besoin. Nous avons vu que le métathéorème fondamental du domaine est celui de la permutation des nombres affichés. J'ai eu beaucoup de peine à m'en servir à bon escient, ce qui est apparu dans la construction de la phrase réflexive sans E où je n'ai pensé que trop tard à améliorer la répartition des lettres constituant l'anagramme. Au cours de l'expérience, cette mauvaise utilisation est arrivée plusieurs fois. Ce n'est pas parce que l'on dispose d'une(méta)connaissance que l'on pense à s'en servir au bon moment. Un des rôles du monitoring est de s'assurer que l'on pense à utiliser un résultat que l'on sait essentiel.

Mais la situation inverse s'est aussi produite : j'utilise trop souvent un métathéorème qui ne mérite pas tant d'intérêt. Cela s'est arrivé pour le résultat indiquant que l'on peut ajouter comme on veut un U et un N : si, au cours de la réalisation de l'anagramme, l'on se trouve avec un U et un N de trop, on peut les éliminer simultanément. J'ai beaucoup trop voulu me servir de ce résultat ; par exemple, j'ai compliqué le deuxième programme en tenant compte de ce métathéorème. J'y ai aussi beaucoup pensé pendant que je construisais l'anagramme. En réalité ce résultat n'a jamais servi ; j'aurais pu me rendre compte a priori qu'il ne pouvait être utile que dans des circonstances exceptionnelles.

4.3. Découvrir de nouveaux métathéorèmes

Le monitoring doit savoir arbitrer entre la recherche de la solution du problème et le travail au niveau méta où l'on trouvera par exemple de nouveaux métathéorèmes qui faciliteront cette recherche ultérieurement. J'ai trouvé les métathéorèmes par hasard pendant que je cherchais la solution et je n'ai jamais essayé systématiquement de voir quels métathéorèmes je pourrais démontrer. Un métathéorème intéressant indiquait que si l'on a réservé un nombre suffisant de "deux", on peut ajouter la lettre que l'on veut dans l'anagramme. Par exemple, si l'on a besoin d'un P qui n'était pas prévu au départ, on peut l'utiliser à condition d'avoir encore un "deux" inutilisé, la deuxième occurrence intervenant quand on dit qu'il y a "deux P". L'avantage est que l'on diffère le choix, car si besoin est on pourrait mettre un M ou un J à la place du P si cela facilite la construction de la phrase. J'ai trouvé ce résultat justement parce que je ne pouvais pas l'utiliser : pour construire la phrase sans E, "deux" était interdit et le plus petit nombre sans E était "trois".

Mais alors, si l'on avait besoin d'un J, il fallait en caser encore un autre ailleurs dans la phrase, ce qui était contraignant pour une lettre rare. En voyant cette contrainte, je me suis rendu compte que la solution serait plus facile si "deux" était permis, ce qui est le cas pour construire des phrases sans T, sans N ou sans S. J'avais un résultat de prêt, très utile pour aborder des problèmes voisins.

4.4. Profiter des surprises

Il est bien connu que la surprise est une cause importante de progrès. Elle montre que l'on n'a pas compris quelque chose et nous incite donc à augmenter nos connaissances. Mais pour pouvoir être surpris, il faut s'attendre à avoir ce quelque chose que l'on n'aura pas. Pour cela, on peut avoir une idée générale de ce que l'on ne doit pas avoir ; la surprise arrive quand on a une valeur interdite, par exemple une probabilité supérieure à 1. On peut aussi avoir une théorie prédictive du domaine qui nous dit ce que l'on doit s'attendre à trouver à l'étape suivante ; on est surpris si les résultats ne correspondent pas aux prévisions. Dans l'expérience que j'ai faite, j'ai eu plusieurs fois des surprises du deuxième type, surprises qui étaient trop souvent mauvaises.

Commençons par un exemple de surprise bénéfique. Quand j'ai commencé l'étude du problème, j'ai considéré les nombres permis, c'est à dire ceux qui ne contiennent pas de E : 1, 3, 5, 6, 8, 10, 18, 20, 23, 25, 26, 28. Dans une deuxième étape, j'ai déterminé pour chaque lettre les nombres où elle apparaissait, en me contentant des nombres simples qui sont formés d'un seul mot : les six premiers nombres et le nombre 20. Les cinq autres nombres permis sont composés, ils sont la combinaison de deux des nombres précédents. Au total, 14 lettres apparaissent dans ces nombres :

C	5
D	10
G	20
H	8
I	3, 5, 6, 8, 10, 20
N	1, 5, 20
O	3
Q	5
R	3
S	3, 6
T	3, 8, 20
U	1, 8
V	20
X	6, 10

Le cas du "un" est un peu spécial, car si une lettre n'apparaît qu'une fois pour dire qu'elle est là, on peut aussi bien ne pas la mettre. La surprise vient de ce que le nombre I apparaît exactement une fois dans chacun de ces nombres simples, en excluant le "un", donc aussi exactement deux fois dans chacun des nombres composés. Pour avoir été surpris, je devais croire implicitement que chaque lettre apparaissait dans peu de chiffres ; je n'avais aucune raison de penser que le I, qui est seulement la cinquième lettre en fréquence du français apparaîtrait dans tous les nombres permis. Cette surprise a été utile, car il est devenu presque possible de prédire le nombre de I dans les nombres qui figureraient dans la phrase, donc de diminuer la combinatoire du problème.

Par contre, j'ai eu une mauvaise surprise quand j'ai essayé d'utiliser pour la recherche de la phrase sans E le programme que j'avais écrit pour la recherche de phrases réflexives. Nous avons vu que ce programme a un mécanisme de création de corps à partir d'un noyau auquel on peut ajouter des éléments de phrase à certains endroits. On peut ainsi créer des milliers de corps pour lesquels on cherche ensuite une solution. Je pensais qu'il suffisait de prendre un noyau et des ajouts sans E pour adapter le système. A priori, j'estimais que j'aurais beaucoup de solutions avec six E résiduels, un peu moins avec cinq E, encore moins avec 4, etc., mais que j'arriverai à en avoir sans aucun E. En réalité, cela n'a pas fonctionné comme cela, toutes les solutions avaient beaucoup de E, la moins mauvaise en ayant encore 5. A la suite de cette mauvaise surprise, j'ai essayé d'en voir les raisons et j'ai compris que pour la plupart des corps, il ne pouvait y avoir de solution. En effet, considérons une lettre qui n'apparaît dans aucun des nombres permis, comme A, lettre que j'appelle "stable", car son nombre d'occurrences est indépendant des nombres. Il faut que son nombre d'occurrences dans le corps soit un des nombres permis. Mais si l'on combine des ajouts, cela ne sera pas vrai dans la plupart des cas puisqu'il y a peu de nombres permis et que le nombre de A va varier selon le nombre de A que contient chacun de ces ajouts : il ne vaudra pas en général un nombre permis. Si c'est le cas, on ferait mieux d'arrêter tout de suite les frais, le corps n'a certainement pas de solution puisque le nombre de A reste stable quels que soient les nombres affichés. Il faut de plus que cela soit vrai non seulement pour le A, mais pour toutes les lettres stables utilisées. On pourrait penser de faire un tri a priori des corps quand on les engendre à partir d'ajouts pour vérifier que les nombres de lettres stables sont tous permis et d'éliminer ceux qui ne satisfont pas ce critère. Cela serait possible s'il était possible d'engendrer des centaines de milliers de corps à partir d'ajouts ; on ne garderait que ceux dont le nombre de lettres stables sont des nombres permis. Mais, du fait que l'on n'a pas le droit d'utiliser le E, il est très difficile de trouver, pour chaque point d'insertion dans le noyau, plusieurs ajouts sans E qui aient un sens. En analysant les raisons de cette mauvaise surprise, j'ai décidé, peut-être à tort, d'abandonner cette voie de recherche.

La surprise permet de découvrir des propriétés inattendues qui aideront à trouver la solution ou au contraire de faire prendre conscience rapidement qu'une voie de recherche est sans espoir. La mauvaise surprise a quand même un effet bénéfique en indiquant qu'une direction est mauvaise, donc en privilégiant les voies qui n'ont pas encore été réfutées : elle aide à canaliser la recherche.

4.5. Les croyances et le raisonnement approximatif

J'ai constamment utilisé dans le monitoring des croyances qui me guidaient, mais qui ne reposaient sur rien de rigoureux. Elles étaient basées sur un raisonnement approximatif où l'on établit des faits douteux à partir d'autres faits douteux en utilisant des règles douteuses. Mais "doute" ne signifie pas incertitude totale. Il faut gérer ce type de raisonnement en évaluant le degré de doute des résultats de façon à ne pas utiliser des croyances qui auraient aussi bien pu être tirées au hasard. Le fait d'utiliser des croyances au niveau monitoring n'est pas gênant : même si elles sont très inexactes, elles n'amèneront jamais à faire prendre pour vrai quelque chose qui ne l'est pas. Simplement, en les utilisant, on ne trouvera pas de résultat pour la plupart des problèmes posés.

Beaucoup de ces croyances ont une origine expérimentale, confortée par quelques justifications. Un exemple de croyance est :

> Un saut important dans la séquence des nombres permis augmente beaucoup la difficulté du problème.

Un exemple de saut dans la séquence des nombres permis est le passage de "dix" à "dix-huit" pour les nombres sans E en français ou, pire, celui de "six" à "thirty" pour la phrase sans E en anglais. Cette croyance est justifiée expérimentalement : pour construire l'anagramme, on est obligé pour la lettre T de choisir entre "dix T" et "dix-huit T". Dans un cas, on manque de T et dans l'autre on en a trop. L'écart est tel en anglais que l'on peut conclure à l'impossibilité du problème. Cela est aussi justifié par le bon sens, car on sent bien qu'il vaut mieux avoir des petits manques un peu partout qu'un gros manque. Mais ces justifications ne sont pas une preuve et cette croyance peut bien être fausse.

Dans d'autres cas, la croyance est basée sur un raisonnement de type mathématique où l'on fait de nombreuses simplifications sans les justifier rigoureusement. Parfois, il serait possible d'être complètement rigoureux, par exemple en encadrant les valeurs possibles d'un résultat, mais le coût de la rigueur serait trop élevé. Pour résoudre un problème moyennement difficile, nous ne devons pas être amenés à résoudre, au niveau méta, un problème bien plus complexe que le problème initial. Le travail au niveau méta ne doit pas être plus important que celui au niveau de base, sans cela on part dans une ascension infinie : on aurait à passer autant de temps au niveau métaméta pour monitorer le niveau méta et il n'y a pas de raison de s'arrêter.

Un exemple de tel raisonnement approximatif s'est produit quand j'ai évalué les chances de succès du deuxième programme combinatoire qui examinait les combinaisons possibles des nombres permis pour onze lettres. J'ai fait le "raisonnement" suivant :pour les nombres d'occurrences inférieurs à 12, on a une chance sur deux d'avoir un nombre permis (1, 3, 5, 6, 8, 10, ce qui en fait bien six sur douze). Si le corps est court, les nombres réels d'occurrences seront vraisemblablement inférieurs ou égaux à 12.Comme il y a onze nombres à déterminer, la probabilité de succès pour un corps donné est2^{-11}, soit 1/2000. Comme il est difficile d'engendrer plusieurs milliers de corps sans E, la voie est sans espoir. Ce raisonnement est bien évidemment très douteux, il n'y a aucune raison pour que les nombres réels se répartissent uniformément entre 1 et 12 ; pourtant, je suis convaincu que cette hypothèse n'est pas loin de la réalité et que la valeur rigoureuse de la fréquence des solution (que je ne vois pas comment évaluer rigoureusement) est nettement inférieure à un millième.

Nous avons une idée de la valeur de l'approximation que nous faisons dans un raisonnement approximatif. Nos systèmes devraient pouvoir faire de telles approximations tout en évaluant une fourchette autour du résultat ainsi obtenu. Les experts humains sont habiles à déterminer rapidement des ordres de grandeur ; nous devrions découvrir comment ils le font pour donner la même capacité à nos systèmes.

4.6. Estimer les chances de succès

Estimer des chances de succès est une activité essentielle du monitoring. Elle se présente sous plusieurs aspects. D'abord, il faut savoir estimer si un problème ou un sous-problème a des chances d'avoir une solution pour ne pas perdre son temps dans une voie sans espoir. Ensuite, il faut savoir évaluer si une méthode particulière a des chances de résoudre le problème posé. Enfin, il est utile de sentir si l'on est proche ou non du succès pour décider si l'on doit continuer dans la voie présente ou s'il vaut mieux l'abandonner.

4.6.1. Un problème a-t-il une solution ?

Il vaut mieux éviter de perdre son temps à vouloir résoudre un problème insoluble. Cela arrive rarement dans les problèmes académiques parce qu'ils ont en général une solution. Par contre, cela se produit souvent pour les problèmes réels ; de plus, au cours de la solution d'un problème académique, on est amené à considérer un sous-problème qui, lui, peut être insoluble.

Dans l'expérience que j'ai faite, la recherche d'une phrase anglaise réflexive sans E s'est révélée insoluble. En examinant la séquence de nombres permis, j'ai vu le saut entre "six" et "thirty" ; j'en ai conclu à l'impossibilité de trouver une telle phrase si l'on veut qu'elle garde une taille raisonnable. La justification dépend de nos croyances. Pour moi, avec l'expérience de la construction de la phrase sans E en français, l'importance du saut suffisait. Si l'on n'en est pas convaincu, on peut utiliser des raisonnement approximatifs. Par exemple, on ne peut se contenter de nombres inférieurs ou égaux à 6 pour de nombreuses lettres, vu la taille de la phrase qui contient déjà obligatoirement une douzaine de nombres. Cela oblige que plusieurs lettres aient des nombres d'occurrences au moins égaux à 30. La phrase sera ainsi très longue, donc il y aura encore plus de lettres qui auront au moins trente occurrences. On aboutira sans doute à une solution, mais elle sera un monstre inacceptable comme phrase. Un tel raisonnement est une modélisation approximative du processus de construction d'une phrase réflexive. Là encore, les systèmes d'IA sont dépourvus de cette possibilité !

4.6.2. Chances de succès d'une méthode

On peut souvent choisir entre plusieurs méthodes pour résoudre un problème. On essayera d'abord celle quia le plus de chances de succès s'il est possible de les évaluer. C'est ainsi que j'ai estimé pour la recherche d'une phrase réflexive sans N que le programme général avait des chances de mener au succès parce que la plupart des nombres inférieurs ou égaux à dix-huit étaient permis : 2, 3, 4, 6, 7, 8, 10, 12, 13, 14, 16, 17, 18. Cette évaluation contraint aussi éventuellement la valeur de certains paramètres pour que l'on ait une chance d'avoir une solution. Toujours dans l'exemple de la phrase sans N, l'absence de nombres permis entre "dix-huit" et "mille" m'a fait conclure qu'il fallait une phrase courte, donc aussi bien le noyau que les ajouts devaient comporter peu de caractères. Il valait mieux prévoir peu d'endroits où l'on pourrait mettre un ajout pour limiter la taille du corps. Pour arriver à un nombre de cas suffisants, il fallait donc prévoir beaucoup de valeurs possibles pour chaque emplacement.

L'analyse des chances de succès d'une méthode ne se contente donc pas de choisir la méthode la plus prometteuse, elle précise des caractéristiques de cette méthode. Il pourrait aussi être intéressant au cours de cette analyse de prévoir ce qui doit se passer. Il faudrait ensuite de vérifier régulièrement que tout se passe comme prévu et examiner les raisons d'un écart éventuel entre les prévisions et la réalité.

4.6.3. Evaluation de la proximité de la solution

Il est utile de savoir si l'on est près ou non de la solution. Si l'on pense que l'on approche du but, on n'hésite pas à faire encore des essais pour s'en rapprocher ; par contre, si l'on pense que l'on en est encore très loin, on abandonnera sans regret cette voie. Cette sensation de proximité est facilitée si l'on a une mesure quantitative de cette proximité.

Au cours de la recherche de la phrase réflexive sans E, cela est souvent intervenu. Par exemple, j'ai commencé par utiliser l'ancien programme. Le nombre de E présents dans les phrases solutions mesuraient bien la proximité de la solution puisque l'on voulait que ce nombre soit nul. Comme il n'est jamais descendu au-dessous de 5 et qu'il lui était en général largement supérieur, j'ai rapidement abandonné cet essai. Par contre pour la phrase sans N, j'avais souvent des nombres de N égaux à 2. Cette proximité m'a conduit à poursuivre ; quelques légères modifications apportées à la méthode ont finalement amené des solutions.

De même, j'ai passé deux après-midis à chercher l'anagramme pour la phrase sans E. A la fin du premier, j'avais une certitude absolue que je trouverai une solution par cette méthode, car j'avais des solutions où j'avais utilisé presque toutes les lettres de départ. Le nombre de lettres résiduelles était là une excellente mesure de la proximité de la solution où il doit là aussi être nul. Comme ce nombre était faible, j'ai été encouragé à poursuivre.

4.7. Estimer au préalable la qualité du résultat

Il ne suffit pas d'avoir un résultat, car les résultats possibles ne sont pas tous équivalents : nous voulons aussi un bon résultat. Je n'ai pas commencé tout de suite par l'anagramme parce que je pensais que cette méthode ne donnerait pas des résultats d'aussi bonne qualité que ceux du premier programme. En effet, avec ce programme, on maîtrise assez bien la phrase que l'on construit puisqu'on en donne le noyau au départ et que l'on choisit les ajouts possibles. Certes, certaines combinaisons d'ajouts ne sont pas totalement satisfaisantes, mais on arrive en général à avoir une phrase réflexive qui exprime assez bien ce que l'on voulait dire.

Par contre, quand on construit un anagramme, on est contraint par les lettres existantes. On essaye de trouver une phrase correcte syntaxiquement et sémantiquement, mais il est quasiment impossible de choisir son sens a priori. Il faut prendre ce qui vient d'après les mots que l'on arrive à créer. Je prévoyais que la qualité du résultat serait bien inférieure, ce qui s'est effectivement produit.

Un autre aspect de la qualité de la solution est d'être sûr de l'obtenir si elle existe ou d'avoir l'indication qu'il n'existe pas de solution. Le deuxième programme était une procédure de décision ; cela avait lourdement fait pencher la balance pour m'inciter à l'écrire.

Nous sommes loin de savoir faire estimer par un ordinateur la qualité du résultat qu'il obtiendra par une certaine méthode. Pour cela, nous devrons lui donner la possibilité de modéliser, là encore approximativement, la méthode que nous voulons mettre en œuvre.

4.8. Estimer au préalable le coût d'une méthode

Pour décider de la méthode que l'on va choisir, il ne suffit pas d'évaluer ses chances de succès et la qualité des résultats obtenus, on doit aussi avoir une idée de son coût, en particulier du temps qu'il faudra pour trouver la solution. Par exemple, j'avais choisi d'écrire le second programme parce que je savais qu'il donnerait la réponse en un temps inférieur à la seconde. Une autre raison pour laquelle j'avais choisi de l'écrire était que cela ne me demanderait pas beaucoup de temps d'écrire un programme d'une centaine d'instructions. Dans ce cas, le coût est intervenu positivement à deux niveaux : au niveau de base parce que le programme créé serait rapide et au niveau méta (donc du métamonitoring) parce qu'il ne me faudrait pas beaucoup de temps pour créer ce programme.

4.9. Le métamonitoring

En général plusieurs critères peuvent s'appliquer pour choisir une décision, aussi donnent-ils souvent des appréciations divergentes. Il faut alors savoir arbitrer entre eux et surtout ne pas oublier d'envisager un critère important. Un exemple d'insuffisance de la gestion du monitoring a été ma décision d'écrire le second programme. Il avait trois très grandes qualités : il donnait rapidement sa réponse, il trouvait sûrement la solution si elle existait et, s'il trouvait une solution, elle serait d'excellent qualité. C'était un gros progrès par rapport au premier programme qui n'avait pas les deux premières de ces qualités ; cela a suffi pour m'inciter à écrire ce programme. Malheureusement, je n'avais pas évalué un autre aspect : la probabilité qu'il y ait une solution. Comme le programme n'en trouvait pas, je l'ai évaluée selon la méthode décrite dans la section 4.6., ce qui m'a conduit à abandonner une voie dans laquelle je n'aurais jamais dû m'engager.

Il se pose donc un nouveau problème, celui du métamonitoring qui pilote le monitoring et s'assure qu'il se fait bien. Cela peut présenter de multiples aspects :

* Il faut vérifier qu'une décision n'a été prise qu'après examen de tous les facteurs importants, sans en oublier un. Dans l'exemple précédent, j'avais omis de considérer les chances de succès.

* Il ne faut pas passer trop de temps à évaluer des facteurs secondaires, il faut commencer par examiner les critères essentiels, les autres critères ne servant au plus qu'à départager les ex-æquo. Je n'ai jamais pris en compte la taille de la phrase qui serait engendrées tant qu'elle restait dans des limites raisonnables. Pourtant une phrase réflexive courte est plus satisfaisante.

* Il faut ne pas évaluer un critère avec une précision supérieure à celle qui est nécessaire pour prendre une décision : pour l'insuccès du deuxième programme, il a suffi de savoir que sa probabilité de succès pour un corps était certainement inférieure à un pour mille. Sa valeur réelle n'avait plus aucune importance.

* Il ne faut pas que le temps passé à savoir si l'on va faire un essai soit supérieur à celui que prendrait cet essai ! C'est une des raisons pour lesquelles j'ai réalisé le second programme : l'écriture d'une centaine d'instructions ne m'a pas demandé beaucoup plus de temps qu'une évaluation plus approfondie de l'intérêt de le réaliser.

Avec ce dernier exemple, nous voyons apparaître le métamétamonitoring, puisque le dernier critère m'a conduit à faire un essai que j'aurais pu éviter en suivant le premier critère. J'ai mal géré le métamonitoring parce que mon métamétamonitoring était trop sommaire. Nous pourrions penser que nous allons alors avoir une ascension infinie des niveaux méta, avec un métamétamétamonitoring pour piloter le métaméta-monitoring. Il n'en est rien car le mécanisme change de nature quand on passe au niveau du métamonitoring. En effet, le monitoring de base n'est pas spécialisé à un domaine ; à cause de sa généralité, cela peut mal se passer dans certaines applications. Par contre le métamonitoring est spécialisé dans le monitoring du monitoring. Nous devrions pouvoir réaliser un système simple et robuste, puisque spécifique. Nous pourrons lui donner un métamétamonitoring très sommaire qu'il sera, lui, complètement inutile de monitorer. Il pourra se passer de métamétamétamonitoring, les connaissances nécessaires à un bon méta-métamonitoring étant figées une fois pour toutes. De toutes les façons, il suffit d'éviter de commettre de grosses erreurs. Le métamonitoring est un garde-fou qui limite en particulier le temps que l'on passe à faire du monitoring. Il n'a pas besoin d'une grande précision, donc besoin que d'un niveau supérieur très rudimentaire.

5. Quelques travaux d'IA mettant en œuvre le monitoring

En général, les systèmes d'IA n'ont pas de possibilités de monitoring, le travail au niveau méta étant fait par le concepteur du système. Celui-ci s'arrange pour que son système ait des performances satisfaisantes pour la famille de problèmes qu'il va rencontrer : il donne implicitement à son système de bons choix de monitoring pour l'environnement limité qu'il va rencontrer. Ceci est manifestement insuffisant quand on réalise des systèmes généraux. Aussi, quelques travaux d'IA, qui ont pour but la réalisation de systèmes généraux et efficaces, font du monitoring. Nous allons examiner quelques unes de ces réalisations.

5.1. ALICE

La première version d'ALICE [Laurière1976] présentait déjà des aspects de monitoring. En particulier, quand le système doit chercher une solution optimale, il estime la difficulté du problème sans optimisation. Si le problème est jugé très difficile, il essaye d'abord de trouver une solution et il le résout donc comme s'il ne cherche pas un optimum. Ce n'est qu'une fois trouvée une première solution qu'il ajoute une contrainte qui lui fait chercher une solution de qualité supérieure. Par contre, si le problème est jugé facile, il essaye dès le départ de trouver une bonne solution. Dans le premier cas, on privilégie le choix le plus informant alors que dans le deuxième on privilégie le choix qui coûte le moins au sens de la fonction économique. Le système connaît trois cas de solution réalisable évidente ; le problème est alors trivial tant qu'il n'y a pas optimisation. Cela arrive quand l'on a une solution si l'on prend le premier sommet dans la liste des images possibles, par exemple si l'on a une solution en donnant la valeur 0à chacun des X_i.

Dans la dernière version d'ALICE [Laurière1996], le système peut engendrer un programme pour résoudre de façon combinatoire un sous-problème. Au cours de la résolution d'un problème, il lui arrive ainsi d'engendrer une dizaine de programmes, le système entremêlant des phases de fonctionnement normal de type ALICE avec l'exécution aussi efficace que possible d'une méthode combinatoire qu'il a traduite dans un programme C. Naturellement, il ne s'agit pas de créer constamment un programme, car il faut tenir compte des temps d'écriture du programme, de sa compilation et de son exécution. Pour décider s'il vaut mieux continuer sous forme ALICE ou écrire et exécuter un programme C, le système compare les estimations du temps pour l'écriture et la compilation du programme et le temps pour trouver la solution suivant la méthode ALICE. Si l'on y gagne un écrivant un programme, le système résout à l'aide de ce programme qu'il écrit le sous-problème en question. Mais la démarche ALICE est toujours utile pour les cas où l'approche combinatoire serait trop coûteuse, ce qui se produit souvent au début de la résolution d'un problème. ALICE écarte rapidement bien des possibilités et il modifie l'énoncé de ce problème, en particulier par l'adjonction de nouvelles contraintes. Il prépare ainsi les éléments utiles pour écrire un programme combinatoire performant.

Cette nouvelle approche est très prometteuse. La meilleure chose à faire est parfois d'écrire un programme combinatoire aussi rapide que possible. Le système doit être assez métaintelligent pour reconnaître que dans certaines situations il ne sert à rien d'être intelligent et que la force brute est alors la meilleure chose à essayer. C'est au système d'IA et non au chercheur en IA de déterminer dans quelles situations il vaut mieux écrire un programme ; c'est également au système d'écrire ce programme.

5.2. EURISKO

Le système EURISKO [Lenat 1982] tient compte des contraintes de temps et de place mémoire grâce au prédicat "If-resources-available" qui est lié à une règle. Il n'exécutera la règle en question que si les ressources nécessaires sont disponibles.

Par ailleurs, ce système, dont le but est de découvrir des concepts nouveaux, prend des précautions contre des erreurs éventuelles [Lenat 1983a], tout à fait comme les sujets étudiés dans la section 2. Par exemple, quand il a généralisé un concept C dans un nouveau concept G, une règle fait spécialement attention au cas où le concept G est en réalité identique au concept C, et où l'on a donc une fausse généralisation.

Enfin il est arrivé que le système parte à la dérive [Lenat 1983b]. Les règles peuvent au départ tout modifier, aussi est-on arrivé à des aberrations comme la règle qui a décidé que toutes les règles étaient mauvaises et qu'il fallait donc les éliminer toutes. Pour y remédier, Lenat a introduit un petit métaniveau de code protégé que le système n'a pas le droit de modifier.

5.3. Sepiar

Le système Sepiar [Parchemal 1988] monitore continuellement son fonctionnement. Son but est d'exécuter efficacement les règles d'un système expert quelconque. Il s'observe pour déterminer s'il est intéressant de maintenir certains ensembles. Par exemple, si des règles souvent considérées demandent qu'un objet X soit un gros cube rouge, il va maintenir l'ensemble des gros cubes rouges. Il y aura ainsi accès directement et ne sera pas obligé de chercher les cubes, puis, parmi ceux-ci, ceux qui sont gros et sélectionner enfin les rouges parmi ceux qui restent. Pour qu'il soit intéressant de maintenir un ensemble, il faut d'abord qu'il serve souvent et aussi qu'il soit assez stable. Si un robot avec des pots de peinture passe son temps à changer la couleur des objets, il sera plus intéressant de ne maintenir que l'ensemble des gros cubes. Ces ensembles sont détectés en cours d'utilisation du système expert, par auto-observation du fonctionnement du moteur.

Le système regarde aussi comment les règles sont déclenchées. Si la règle R5 est toujours déclenchée avec succès après la règle R3, on peut mémoriser ce fait de façon à automatiquement lancer la règle R5 chaque fois que l'on déclenche R3. Le système remarque ceci toujours par observation de ce qu'il fait, mais il a en plus la possibilité de faire des expériences pour vérifier que ce qu'il a vu n'est pas une simple coïncidence. Par exemple la formule suivante, qu'il crée, lui indique une série d'expériences à faire, le crochet fermant ramenant le système à l'état où il se trouvait au moment du crochet ouvrant correspondant. Cette formule :

[[[R3], R2, R3],[R1, [R3], R2, R3]]

indique de faire successivement les essais suivants à partir de la situation initiale : on pourra comparer les exécutions de R3, puis de R2 suivi de R3, puis de R1 suivi de R3, et enfin deR1 suivi de R2 et de R3. On saura donc la règle R2 a une influence surR3 et on verra ensuite si R1 y est pour quelque chose. Cette possibilité de faire des expériences est très importante dans tout apprentissage, car on ne se contente plus d'espérer la venue d'un événement intéressant qui permettra de trancher entre deux hypothèses et qui ne se produira peut-être jamais.

5.4. Sade

Le système Sade [Kornman 1993] examine le fonctionnement d'un système expert pour y retrouver des erreurs. Quand il détecte une erreur, il peut dans certains cas la corriger. Son module de détection d'anomalies voit par exemple que l'on a fait beaucoup de déclenchements de règles sans avoir eu de résultat. Quand il a soupçonne une erreur, il entre dans un autre mode de fonctionnement où il crée une trace qu'il va ensuite analyser. Si cette analyse lui permet de découvrir une erreur, il applique un remède lié à cette erreur, par exemple inhiber momentanément l'utilisation d'une règle. Il prend ensuite des précautions pour vérifier que le remède a bien fait son effet. Si tout va bien, il repasse en mode normal où la surveillance de ce qu'il fait est beaucoup plus légère.

Le système surveille aussi l'expertise de surveillance. Cela amène la possibilité théorique d'une ascension infinie, qui ne s'est pas présentée dans les expériences faites. Toutefois, elle pourrait se produire ; cela nécessite de compléter le système, par exemple avec une partie intouchable comme avec EURISKO.

6. Conclusion

Lors de l'expérience que j'ai faite, mon monitoring a été loin d'être performant, en particulier parce que mon métamonitoring a été quasiment inexistant. Mais un mauvais monitoring est bien supérieur à une absence totale de monitoring. Evidemment, avec un bon métamonitoring, l'avantage apporté par le monitoring serait encore plus grand. Aussi, un système d'IA doté de telles possibilités devrait-il avoir, dans un domaine nouveau, des performances largement supérieures à celles des meilleurs résolveurs de problèmes humains. Toutefois, pour réaliser des systèmes qui utiliseraient à bon escient aussi bien le monitoring que le méta-monitoring, nous devons d'abord maîtriser des techniques à peu près complètement ignorées des recherches actuelles en IA, et en particulier la possibilité de faire des raisonnements non rigoureux. En effet, traités rigoureusement, les problèmes de monitoring sont extrêmement complexes : la rigueur est alors trop coûteuse. Elle va à l'encontre du but du monitoring qui est de gagner du temps. De plus, elle est inutile, on n'a pas besoin de résultats parfaits puisqu'une erreur, bien que regrettable, ne peut avoir de conséquence plus grave qu'une absence de résultat, jamais de faire prendre pour vrai un résultat faux. Mais ne pas être rigoureux va à l'encontre de notre culture mathématique. Même si nous en acceptons l'idée, nous n'arrivons pas à la mettre en œuvre car nous connaissons encore trop mal les mécanismes qui permettent de faire des approximations et de les gérer.

Bien que le monitoring soit essentiel pour les êtres humains quand ils résolvent un problème, il est presque toujours absent des systèmes de résolution de problèmes, qui manquent donc totalement de généralité. Leur intelligence vient de leur concepteur qui leur a donné l'expertise particulière à un domaine. Le monitoring n'est effectivement pas nécessaire à l'expert et au système expert qui l'ont implicitement dans leurs heuristiques. Mais peut-on parler de l'intelligence d'un système qui ne fait que répéter comme un perroquet les comportements que son concepteur a prévus ? La prochaine étape passe par la réalisation de systèmes métaexperts en résolution de problèmes qui, grâce au monitoring, sauront s'adapter à la résolution de problèmes nouveaux.

7. Références

[Allwood 84] Allwood C., *Error Detection Processes in Statistical Problem Solving*, CognitiveScience 8, 413-437, 1984.

[Hofstadter 85] Hofstadter D., *Metamagical Themas*, Basic books, 1985.

[Kornman 93] Kornman S, *SADE : un Système Réflexif de Surveillance à Base de Connaissances*, Thèse de l'Université Paris 6, 22-1-1993.

[Laurière 76] Laurière J-L., *Un langage et un programme pour énoncer et résoudre des problèmes combinatoires*, Thèse de doctorat d'état, 21-5-1976.

[Laurière 96] Laurière J.-L., *Propagation de contraintes ou programmation automatique ?*, Rapport Laforia 96/19, 1996.

[Lenat 82] Lenat D., *The Nature ofHeuristics*, Artificial Intelligence 19, 189-249, 1982.

[Lenat 83a] Lenat D., *TheoryFormation by Heuristic Search*, Artificial Intelligence 21, 31-59,1983.

[Lenat 83b] Lenat D., *EURISKO: A Program that Learns NewHeuristics and Domain Concepts*, Artificial Intelligence 21, 61-98,1983.

[Parchemal 88] Parchemal Y., *SEPIAR : un système à base de connaissances qui apprend à utiliser efficacement une expertise*, Thèse de l'Université Paris 6, 22-12-1988.

[Pitrat 96] Pitrat J., *Ce titre contient quatre 'a', un 'b', cinq 'c', cinq 'd', dix-neuf 'e', deux 'f', un 'g', deux 'h', treize 'i', un 'j', un 'k', un 'l', un 'm', seize 'n', trois 'o', quatre 'p', sept 'q', sept 'r', sept 's', quinze 't', dix-huit 'u', un 'v', un 'w', six 'x', un 'y' et quatre 'z'.*, Rapport LAFORIA 96/26, Université Paris 6, 1996.

[Pitrat 99] Pitrat J., *Une expérience de monitoring*, Rapport de recherche LIP6 1999/014, 1999.

[Schoenfeld 85] Schoenfeld, A., *Mathematical Problem Solving*, Academic Press, 1985.

Compilation dynamique de connaissances déclaratives : le système SEPIAR 2

Yannick PARCHEMAL
Centre de Recherche en Informatique de Paris 5
Université René Descartes
45, rue des Saints-Pères
75270 PARIS Cedex 06

Résumé: L'utilisation efficace de connaissances déclaratives est un défi important pour l'intelligence artificielle. Le système Sepiar 2 est un système à base de connaissances qui analyse et modifie dynamiquement son propre fonctionnement en vue d'une utilisation efficace de la base de connaissances. Nous présentons ici ce système, en particulier le langage d'expression des connaissances déclaratives, la représentation interne dont la principale composante est un réseau de relations, les capacités de paramétrage du système et enfin une présentation de la méta-expertise qui détermine comment utiliser les connaissances en fonction de l'analyse de la trace et du résultat des expériences qu'elle génère.

Mots Clés: Monitoring, connaissances déclaratives, résolution de problèmes, méta-connaissances, apprentissage.

8. Introduction

Un bon système à base de connaissances devrait permettre à l'utilisateur de donner ses connaissances sous la forme qu'il estime la plus appropriée: sous une forme déclarative (sans indication de la façon et du moment où les connaissances doivent être utilisées), procédurale (à l'aide d'un programme) ou sous des formes intermédiaires (par exemple des règles).

L'utilisation de connaissances déclaratives pose toutefois encore des problèmes d'efficacité en particulier lorsque la taille de la base de connaissances est importante. La conséquence directe de ce problème est que, en pratique, l'utilisateur a souvent recours à des formes plus procédurales que celles qu'il préférerait soit parce que le système ne lui propose pas autre chose soit par autocensure pour éviter une trop grande inefficacité. Dans les deux cas de figure, la tâche de l'utilisateur en est compliquée et l'intérêt du système à base de connaissances diminué.

La solution choisie le plus souvent pour améliorer l'efficacité de l'utilisation des connaissances déclaratives consiste à trouver de nouveaux algorithmes plus performants. Les résultats qu'ils obtiennent sont très contrastés et dépendent des bases de connaissances utilisées. De plus ils ne permettent pas toujours une expression très déclarative des connaissances.

Nous ne pensons pas que cette direction soit la bonne. En effet, les situations rencontrées sont très différentes d'une base à l'autre et, pour une même base de connaissances, d'une connaissance à une autre. Une même méthode peut se révéler très bonne pour utiliser une connaissance et se révéler inefficace pour une connaissance présentant des caractéristiques apparentes similaires.

L'analyse syntaxique de la base n'est pas suffisante pour déterminer les bonnes méthodes d'utilisation des connaissances : de nombreux paramètres essentiels sont en effet en pratique inaccessibles par analyse syntaxique mais peuvent par contre être évalués facilement en cours de fonctionnement. La solution permettant de résoudre ce problème nécessite l'utilisation d'informations dynamiques collectables uniquement en cours de session.

Le système Sepiar2 a été conçu à cet effet et permet une adaptation dynamique des méthodes d'utilisation des connaissances. Son principe de fonctionnement (figure 1) est le suivant : lors de l'utilisation de l'expertise de base, une trace est produite. Cette trace est régulièrement analysée par une méta-expertise qui peut demander, en fonction du résultat de ses analyses, le changement de certaines méthodes appliquées pour utiliser l'expertise de base. Le système Sepiar2 permet une adaptation fine des méthodes aux connaissances : un changement de méthodes n'affecte pas toute l'expertise mais uniquement les connaissances concernées.

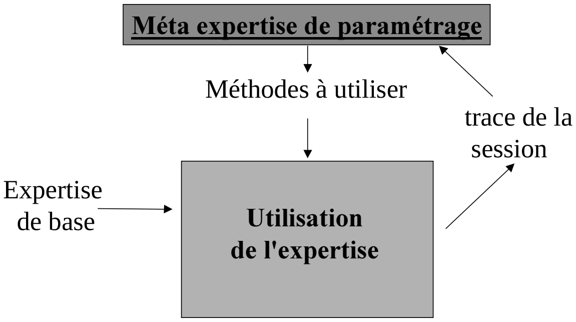


***figure 1 :** schéma simplifié de Sepiar 2*

Les principaux modules du système Sepiar2 (figure2) sont un module de compilation qui produit la représentation interne de la base de connaissances, un moteur d'inférence dont le rôle est de déclencher des règles et la méta-expertise de paramétrage qui génère des expériences pour collecter des informations et indique au système les méthodes à utiliser.
La représentation externe des connaissances est décrite au chapitre 9. Les entités sont des objets instances de classes. Les connaissances sont exprimées de façon déclarative par la définition de relations tandis que les règles permettent de fournir des connaissances sous une forme plus procédurale.

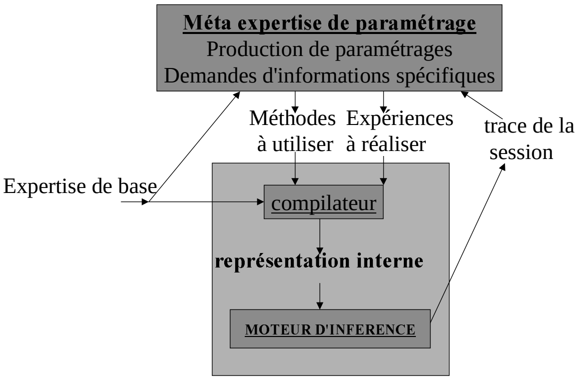


***figure2 :** les modules du système Sepiar2*

Au chapitre suivant, nous parlons de la représentation interne des connaissances et plus particulièrement de la représentation interne des relations dérivées. Quatre types de relations dérivées permettent de manipuler des conjonctions, des disjonctions, des cardinaux d'ensembles et des suites ordonnées d'éléments. Ces relations sont regroupées au sein d'un réseau qui permet d'effectuer les mises à jour nécessaires lors de la modification de la base de faits. Nous décrivons également dans ce chapitre la correspondance entre les représentations externes et internes des relations.
Le système Sepiar 2 a des capacités de paramétrage importantes que nous présentons au chapitre 11. Ces capacités ont trait en particulier à la mémorisation des relations, à l'ordonnancement des expressions d'une conjonction, au mode de propagation des modifications dans le réseau et à la topologie du réseau de relations. Pour chaque relation, les différentes options de paramétrage sont réifiées dans un objet appelé le paramétrage actif. Il est constitué d'un ensemble de conseils permettant au compilateur du système de savoir quelles structures de données utiliser et quels algorithmes générer. A la première utilisation de l'expertise de base, le paramétrage actif est un paramétrage par défaut dont nous présentons les différentes caractéristiques. Plusieurs paramétrages peuvent être définis pour une même relation. Nous montrons comment définir de nouveaux paramétrages et comment changer le paramétrage actif. Ce chapitre se termine par un regard sur les différentes catégories d'algorithmes générés.
La méta-expertise du système Sepiar 2 permet de déterminer les paramétrages de façon autonome. Le chapitre 12 décrit le principe de fonctionnement de cette méta-expertise et plus particulièrement les expériences générées pour obtenir des informations pertinentes. Certaines expériences permettent d'obtenir

des informations qui sont utilisées pour générer de nouveaux paramétrages. D'autres permettent de comparer en terme d'efficacité deux paramétrages possibles : ceci permet en particulier à la méta-expertise de comparer en terme d'efficacité un paramétrage qui vient d'être créé avec le paramétrage actif et de demander le changement de paramétrage si le paramétrage actif ne semble plus être le meilleur. Le système Sepiar2 obtient des résultats intéressants que nous donnons et discutons en fin de chapitre.

9. Le langage d’expression des connaissances

Les connaissances peuvent être exprimées sous des formes déclaratives et procédurales. Les connaissances les plus déclaratives sont exprimées en définissant des relations. Les règles permettent de définir des actions conditionnelles et sont regroupées dans des paquets de règles. Un langage procédural (ressemblant à Lisp) permet quant à lui de définir et d'utiliser des actions et de fournir ainsi des connaissances purement procédurales qui peuvent être utilisées soit dans la partie action des règles soit directement par l’utilisateur du système.

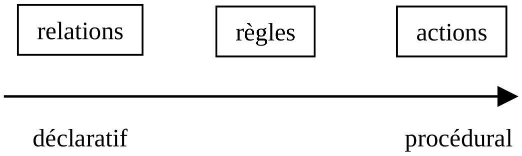


figure 3 : *différentes formes de connaissances*

Après avoir présenté le problème des cryptadditions (choisi ici comme expertise de base pour illustrer les différents exemples), nous décrivons le langage d'expression des connaissances, en particulier les relations et les règles.

9.1. Le problème des cryptadditions

Une cryptaddition est une addition classique avec des lettres à la place des chiffres. Le problème est de trouver un chiffre différent pour chaque lettre de telle sorte que l'addition soit valide, les lettres à gauche de chaque ligne ne pouvant correspondre au chiffre 0. Le problème est résolu par propagation de contraintes et backtrack chronologique.

colonne	4	3	2	1	0
ligne					
2		S	E	N	D
1		M	O	R	E
0	M	O	N	E	Y

figure4 : *la cryptaddition SEND+MORE=MONEY*

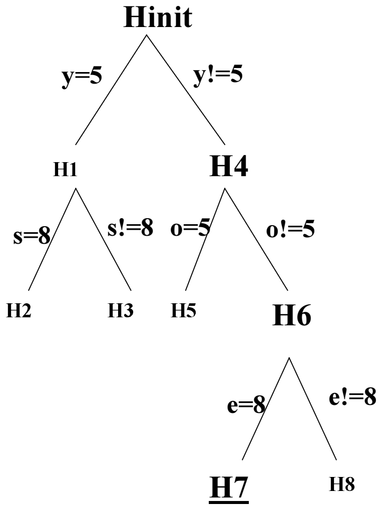


figure5 : *l'arbre des hypothèses*

Lorsque la propagation de contraintes s'avère insuffisante et que des nouvelles hypothèses doivent être considérées, deux hypothèses filles sont créées : une où la valeur d'une variable est fixée, l'autre où au contraire cette valeur est interdite. Ainsi lorsque la propagation sur H6 s'est révélée insuffisante, deux hypothèses filles ont été créées : H7 où la valeur de e est 8 et H8 où 8 est une valeur interdite pour e. L'hypothèse en cours d'utilisation est appelée ici l'hypothèse courante (dans l'exemple : H7). Les hypothèses ancêtres de l'hypothèse courante (dans l'exemple : Hinit, H4, H6) ainsi que l'hypothèse courante sont dites actives.

La position de l'occurrence d'une lettre dans la cryptaddition est repérée par ses numéros de lignes et de colonne comme indiqué en figure4.

9.2. Règles et paquets de règles

Un paquet de règles est un ensemble de règles. L’exécution d’un paquet de règles se traduit par le déclenchement des règles déclenchables du paquet et se termine lorsque plus aucune règle n’est déclenchable.
Ainsi, pour résoudre une cryptaddition on exécute le paquet nommé CRYPT.
Une règle est une action conditionnelle : si certaines conditions sont vérifiées, alors on exécute un certain nombre d’actions.
Les règles présentées en figure 6, qui appartiennent au paquet de règles CRYPT, nous permettent d'avoir dès à présent un premier aperçu des possibilités du système.
La première permet d’interdire la valeur 0 à toutes les lettres se trouvant à gauche d’une ligne. Elle est composée d’une seule prémisse et d’une conclusion et dit que si une lettre ?l se trouve à gauche d’une colonne de ?crypt alors 0 est une valeur interdite pour la lettre ?l dans la cryptaddition ?crypt. La deuxième, composée de 5 prémisses et d’une conclusion, permet d'affecter une valeur à une lettre dans le cas où seule une valeur est possible pour cette lettre.
Plus généralement, la syntaxe de définition d'une règle est la suivante:
(DEF_REGLE <nom> <paquet> {<prio>} SI <prémisse>* ALORS <conclusion>*)
où prio est une donnée facultative fixant la priorité de la règle qui est un entier positif ou négatif et dont la valeur par défaut est 0.

```
// définition du paquet CRYPT
(DEF_PAQUET "CRYPT" [ ])
// définition d'une règle pour le paquet CRYPT
(DEF_REGLE "zéro est une valeur interdite"  CRYPT
SI // une lettre ?l se trouve à gauche d’une colonne de ?crypt
    (LettreAGauche [1]?l ?crypt)
ALORS // 0 est une valeur interdite pour ?l de ?crypt
         (AFFIRMER (valeur_interdite ?crypt ?l  0)))
(DEF_REGLE "AffecterValeur" CRYPT
SI // ?x est un élément de l'ensemble de départ de ?f
  (depart_application ?x ?f)
  // ?v est une valeur possible pour ?f(?x)
  (valeur_possible ?f ?x ?v)
   // il n'y a pas de valeur affectée pour ?f(?x)
  (NON (valeur_affectee ?f ?x ?[2] ) )
  // le nombre de valeur possible pour ?f(?x) est égale à 1
  (==  [3]#(valeur_possible ?f ?x ? )   1)
   // l'hypothèse courante est ?h
  (hypothese_courante ?h)
ALORS  // affecter ?v  à ?f(?x) dans l'hypothèse courante
    (AFFIRMER (valeur_affectee_hyp ?f ?x ?v ?h)))
```

figure 6 : *deux règles du paquet CRYPT*

9.3. Classes et objets

Les entités sont des instances de classe. Les classes dérivent d'une super-classe, exceptée la classe racine de la hiérarchie. En figure 7, trois classes sont définies comme sous-classes de la classe prédéfinie "ObjetNommé" et un objet "crypt_SMM", instance de la classe CryptAddition est créé.

---

1 les identificateurs de variables commencent par un ?

[2] ? désigne une variable anonyme

[3] # désigne le cardinal

```
(DEF_CLASSE "CryptAddition" ObjetNommé)
(DEF_CLASSE "hypothese" ObjetNommé)
(DEF_CLASSE "application" ObjetNommé)
(CREER_OBJET "crypt_SMM" CryptAddition)
```

figure 7 : *définitions de classes et création d'objets*

9.4. Relations

Le concept principal pour la représentation déclarative des connaissances est le concept de relation entre entités. Il existe trois catégories de relations: les relations de base, les relations primitives, et les relations dérivées.

9.4.1. les relations de base

Les relations de base sont des relations pour lesquelles les instances sont explicitement ajoutées ou supprimées par exemple dans les parties actions des règles ou directement par l'utilisateur du système. Les instances des relations de base composent l'espace de travail. L'ajout (resp. le retrait) d'une instance est réalisé par l'action prédéfinie AFFIRMER (resp. INFIRMER).

Un exemple de relation de base pour le problème de la cryptaddition est la relation PositionCrypt (figure 8) permettant de fournir la position des différentes lettres de la cryptaddition.
(PositionCrypt ?crypt ?ch ?lig ?col) signifie "la lettre ?ch est située ligne ?lig et colonne ?col dans la cryptaddition ?crypt". La cryptaddition SMM est donnée par 13 instances (une par occurrence de lettre) de PositionCrypt en utilisant l'action AFFIRMER. La partie PROPRIETES de la définition de PositionCrypt précise que pour une cryptaddition, une ligne et une colonne données ne correspond qu'une seule lettre.

```
/* définition d'une relation de base */
(DEF_RELATION  "PositionCrypt"
   / *les quatre paramètres de la relation */
   [1][ [?crypt CryptAddition] [?ch chaine]
      [?lig entier] [?col entier]]

          PROPRIETES
      /* connaissant  ?crypt ?lig et ?col , une valeur au plus est possible pour ?ch */
       [[?crypt ?lig ?col]  monovalue]
   )

/* les 13 faits définissant la cryptaddition*/
(AFFIRMER (PositionCrypt  "crypt_SMM" "S" 2 3))
(AFFIRMER (PositionCrypt "crypt_SMM" "E" 2 2))
……
(AFFIRMER (PositionCrypt "crypt_SMM" "Y" 0 0))
```

figure 8 : *la relation de base PositionCrypt*

Dans la règle AffecterValeur (figure 6), deux autres relations de base sont utilisées : depart_application (en prémisse 1) et valeur_affectee_hyp (dans la conclusion). La définition de ces deux relations est donnée figure 9.

```
// ?x est un élement de l'ensemble de départ de ?f
(DEF_RELATION "depart_application"
[[?x chaine][?f application]])

// ?f(?x) vaut ?v dans l'hypothèse ?h
(DEF_RELATION "valeur_affectee_hyp"
[?f application][?x chaine][?v objet][?h hypothèse]])
```

figure 9 : *deux autres relations de base*

---

[1] [et] sont les séparateurs de liste

9.4.2. les relations primitives

Ce sont des relations prédéfinies correspondant à des relations arithmétiques dont la définition utilise des programmes C.
Par exemple, "==" (prémisse 4) est une relation primitive. Il en est de même de relations concernant la comparaison d'entiers ou la somme d'entiers.

9.4.3. les relations dérivées

Les relations dérivées sont des relations dont les instances sont données implicitement par la définition associée. Nous donnons ci-dessous quatre exemples de relations dérivées. La syntaxe est la même que pour les relations de base avec en plus le mot clé VRAISI suivi de l'expression de définition de la relation.
La relation LettreAGauche est une conjonction de deux expressions, la deuxième expression étant une négation.

```
// ?ch est à la gauche d'une ligne de ?crypt
(DEF_RELATION "LettreAGauche"
          [[?ch chaine][?crypt CryptAddition]]
  VRAISI
  (ET //il est vrai que ?ch est une lettre en colonne ?col
      (PositionCrypt ?crypt ?ch ?lig  ?col  )
      // il n'existe pas de lettre pour la même cryptaddition sur la même ligne en colonne ?col+1.
      (NON(PositionCrypt ?crypt ?x ?lig (+ ?col 1))))
```

La relation valeur_affectee est une conjonction de deux expressions relationnelles.

```
// ?v a été affecté à ?f( ?x) dans une hypothèse active */
(DEF_RELATION "valeur_affectee"
          [[?f application][?x objet][?v objet]]
  VRAISI
     (ET// ?h est une hypothèse active
          (hypotheseActive ?h)
         //?v a été affecté à ?f( ?x) dans l'hypothèse ?h
         (valeur_affectee_hyp ?f ?x ?v ?h)))
```

La relation hypotheseCourante est une conjonction de deux expressions, la deuxième expression étant la négation d'une conjonction.

```
/* L'hypothèse courante est ?h */
 (DEF_RELATION "hypothèseCourante"
          [ [?h hypothèse]]
VRAISI (ET // ?h est une hypothèse active
             (hypotheseActive ?h)
             // aucune hypothèse fille n'est active
             (NON (ET(hypothese_fille ?hf ?h)
                         (hypotheseActive ?hf))))
```

La relation valeur_possible est définie par une disjonction de 3 expressions, les deux dernières étant des conjonctions.

```
// ?v est une valeur possible pour ?f(?x)
 (DEF_RELATION "valeur_possible"
          [[?f application][?x objet][?v objet]]
 (OU
  //1. ?v est la valeur affectée à ?f(?x)
  (valeur_affectee ?f ?x  ?v)
  //2. cas où ?f est une injection
  (ET // ?f est une injection
       (type_application "injection" ?f)
       //?v est un élément de l'ensemble d'arrivée de ?f
       (arrivee_application ?v ?f)
       //?x est un élément de l'ensemble de départ de ?f
       (depart_application ?x ?f)
       // ?v n'est pas une valeur interdite pour ?f( ?x)
       (NON(valeur_interdite ?f ?x ?v))
       // il n'y a pas de valeur affectée pour ?f( ?x)
       (NON(valeur_affectee ?f ?x ?))
       // ?v n'est pas une valeur déjà affectée
       (NON(valeur_affectee ?f ? ?v))
))
  //3. cas où ?f n'est pas une injection
  (ET (arrivee_application ?v ?f)
       (depart_application ?x ?f)
       (NON(valeur_interdite ?f ?x ?v))
       (NON(valeur_affectee ?f ?x ?))
       ?f n'est pas une injection
       (NON(type_application "injection" ?f)))))
```

9.4.4. Attributs et relations binaires

Lors de la définition d'une classe, on peut définir pour cette classe des attributs. Par exemple, on peut définir la classe application avec deux attributs nommés "arrivee" et "type".

```
(DEF_CLASSE "application" objetnomme
    ["arrivee" objet multivalue]
    ["type" chaine monovalue])
```

"arrivée" est un attribut dont la valeur associée est un ensemble d'objets de la classe objet. "type" est un attribut dont la valeur associée est un objet de la classe chaîne. Ceci n'est qu'une facilité syntaxique et cette définition est en fait équivalente à la définition simple de la classe suivie de la définition de deux relations binaires:

```
(DEF_CLASSE "application" objetnomme)
(DEF_RELATION "arrivee_application"
[["arrivee" objet]["application" application]])
(DEF_RELATION "type_application"
     [["type" chaine]["application" application]]
    PROPRIETES
     [[2] monovalue])
```

Les attributs sont aussi utilisés avec profit dans les définitions de relations dérivées. Par exemple, l'expression[1]

```
[?a application ^type "injection"  ^depart ?x  ^arrivee ?v]
```

permet de remplacer la conjonction des trois expressions relationnelles suivantes:

```
(type_application "injection" ?f)
(arrivee_application ?v ?f)
(depart_application ?x ?f)
```

---

[1] les attributs sont repérés syntaxiquement par un ^

Les attributs n'ont pas d'existence dans le système Sepiar2 (i.e. il n'y a pas de classe attribut). En effet, les transformations en relation binaire que nous venons d'évoquer ont lieu dès la lecture des définitions.

9.4.5. fonctions associées aux relations

Le concept de fonction associée à une relation est important dans le système Sepiar2. Une fonction permet, étant données des valeurs pour certains paramètres d'une relation de connaître les valeurs possibles pour les autres paramètres de cette relation[1].

Considérons la relation valeur_possible. C'est une relation ternaire et (valeur_possible ?f ?x ?y) signifie que ?y est une valeur possible pour ?f(?x). Supposons que le système ait besoin des différentes valeurs possibles pour ?f(?x). Ceci correspond au problème suivant : étant données deux valeurs fixées pour les deux premiers paramètres ?f et ?x , déterminer l'ensemble des valeurs possibles pour ?y. La fonction qui à ?f et ?x associe l'ensemble des valeurs de ?y correspondantes est notée F12_valeur_possible, les chiffres 1 et 2 indiquant que les deux premiers paramètres de la relation sont fournis.

Plus généralement, si rel est une relation d'arité n, la fonction qui, étant donnée la valeur des paramètres de numéro a_1, a_2, a_p associe l'ensemble des (n-p)uplets (ou l'unique (n-p)-uplet si la fonction est monovaluée) de valeurs possibles des paramètres restants est notée :

F$a_1a_2\ldots a_p$_rel.

Il y a quelques cas particuliers où la notation ne suit pas cette forme générale pour des raisons de commodité de lecture, en particulier :

- si tous les paramètres sont donnés (la valeur est alors booléenne), la fonction porte le même nom que la relation.
- si la relation est une relation binaire et que des noms ont été donnés aux deux paramètres lors de la définition de la relation, les fonctions d'arité 1 ont un nom plus « naturel ». Si p1 et p2 sont les noms des deux paramètres,
    - la fonction qui associe à une valeur du premier paramètre le second est notée p2_de_p1,
    - la fonction qui associe à une valeur du premier paramètre le second est notée p1_de_p2.

Ainsi, pour la relation (etat_hypothèse ?e ?h) vrai ssi l'état de l'hypothèse ?h est ?e, on notera etat_de_hypothese la fonction qui, à une hypothèse, associe son état.

La figure 10 donne des exemples d'utilisation de fonctions de la relation valeur_possible.

```
//Quelles sont les instances de la relation valeur_possible ?
## (F_valeur_possible)
= {[!F "N" 6!] [!F "N" 7!] [!F "O" 0!] [!R 1 1!] [!F "E" 6!] [!R 4 1!][!F "E" 5!] [!F "Y" 2!] [!R 3 1!]
[!R 4 0!] [!F "D" 7!] [!F "R" 8!] [!F"D" 6!] [!R 0 1!] [!F "S" 9!] [!F "M" 1!] [!R 2 0!] }
## // 2 est-elle une valeur possible pour F("N")?
## (valeur_possible  F "N" 2)
 = !! // faux
## // 6 est-elle une valeur possible pour F("N")?
## (valeur_possible  F "N" 6)
 = 1 // vrai
// on connaît les deux premiers paramètres, on veut le troisième
## // quelles sont les valeurs possibles pour F("N")
## (F12_valeur_possible  F "N")
 = {6 7 }
## //Quelles sont les couples du produit cartésien possible pour  F?
## (F1_valeur_possible F)
 = {[!"D" 6!] [!"E" 5!] [!"N" 6!] [!"O" 0!] [!"N" 7!] [!"D" 7!] [!"M"
1!] [!"S" 9!] [!"R" 8!] [!"Y" 2!] [!"E" 6!] }
## //Quelles sont les éléments de l'ensemble de départ de F
## //pour lesquelles 6 est une valeur possible ?
##  (F13_valeur_possible F 6)
 = {"N" "D" "E" }
```

figure 10

---

[1] ici, nous n'employons le terme fonction que dans ce sens

9.4.6. syntaxe des définitions externes

La définition donnée par l'utilisateur est tout d'abord expansée, en particulier:

- (NON <expression>) est expansée en (== #<expression> 0) , la négation est en effet dans Sépiar2 une négation par l'absence.
- #<expression> est expansée en (CUMUL Fun <expression>) où Fun désigne la fonction constante de valeur 1. (CUMUL Fun <expression>) a comme valeur la somme des résultats de la fonction Fun appliquée à chacun des éléments de <expression>, ce qui correspond donc bien au cardinal de <expression>.
- les attributs sont transformés en relation binaire

Une définition externe expansée est une expression booléenne dont la syntaxe est la suivante :

```
<expression_booleenne>::=
   (<fonction_booléenne> <argument>*)/
   (ET <expression_booleenne>*)/
   (OU <expression_booleenne>*))
<argument> ::=<expression>/<atome>
<expression>::=
   (<fonction> <argument>*) /
   (CUMUL <fonction> <expression>)
<expression_simple>::=(<fonction > <atome>*)
<atome>::=<variable>/<nombre>/<chaîne>/<objet>
```

10. Représentation interne

Nous décrivons dans ce chapitre la représentation interne des connaissances dans Sepiar2 en commençant en 10.1 par la définition interne des relations dérivées et en 10.2 par celle des règles. Lors de la première utilisation d'une base, le système doit transformer la définition externe en définition interne et nous montrons le principe de cette transformation en 10.3. Les relations ainsi que les règles sont associées dans un réseau dont le principe est donné en 10.4.

10.1. définition interne des relations dérivées

Nous décrivons ici les quatre types de définition interne de relations dérivées : conjonction, disjonction, cumul et suite.

10.1.1. les conjonctions

La définition interne de telles relations est une conjonction d'expressions simples. Elles sont définies par:

- les variables correspondant aux paramètres
- la classe de certaines variables [1]
- les expressions simples[2] de la conjonction

Exemple:
une définition interne de la relation valeur_affectee est :
(CONJONCTION [?f ?x ?v]
        [ ]
        [(hypotheseActive ?h)
                  (valeur_affectee_hyp ?f ?x ?v ?h)])

10.1.2. les disjonctions

La définition interne d'une disjonction R est la liste des relations dont la relation R réalise l'union et s'écrit :
(DISJONCTION [R_1 … R_n])
Une définition interne de valeur_possible est ainsi:
(DISJONCTION [Rvaleur_possible1 Rvaleur_possible2 Rvaleur_possible3]) où chacune de ces trois relations correspond à l'élément correspondant de la définition externe vu en 9.4.

---

[1] celles que l'on ne peut retrouver par analyse syntaxique des expressions de la conjonction

[2] une expression simple est une expression fonctionnelle dont les arguments sont des atomes (pas des expressions)

10.1.3. cumul

Les relations ayant une définition interne de ce type sont générées par le système et correspondent le plus souvent à la représentation de négations ou de cardinaux.

Supposons que le système ait besoin de connaître le nombre de valeurs possibles à un moment donné pour ?f(?x). (F12_valeur_possible ?f ?x) étant l'ensemble des valeurs possibles, la définition interne de la relation cumul est:

(CUMUL Fun F12_valeur_possible), Fun représentant la fonction constante de valeur 1.

La relation correspondante a pour nom CARDF12_valeur_possible et (CARDF12_valeur_possible ?f ?x ?n) est vrai ssi le nombre de valeur possibles pour ?f(?x) vaut ?n.

Plus généralement, soit une relation R d'arité n et fR une fonction de R d'arité nf. Soit une fonction fnum à valeur entière dont l'arité et la classe de ses paramètres sont compatibles avec les paramètres d'arrivée de fR. La relation RC dont la définition est (CUMUL fnum fR) est une relation d'arité nf+1, (RC $?x_1$ $?x_2$ … $?x_{nf}$?r) étant vrai ssi, le cumul des valeurs de la fonction fnum appliquée aux éléments de l'ensemble (fR $?x_1$ $?x_2$ … $?x_{nf}$)[1] est égal à ?r.

Lorsque fnum est la fonction constante qui renvoie 1, ?r correspond alors au cardinal de l'ensemble et si ?r = 0, cela signifie qu'il y a absence d'élément ce qui correspond à la négation par l'absence utilisée dans le système Sepiar2.

10.1.4. suite

il est parfois utile de pouvoir parler du $p^{ème}$ élément d'un ensemble étant donné un critère d'ordre total sur les éléments de cet ensemble. Sepiar 2 permet la définition de telles relations qui sont souvent utilisées pour parler de minimum ou de maximum.

Soit une relation R d'arité n et fR la fonction de R d'arité nf dont les éléments de l'ensemble de départ correspondent aux paramètres p_{fR1}, p_{fR2} …p_{fRnf}. Soit une liste de fonction $fnum_i$ à valeur entière dont l'arité et la classe de leurs paramètres sont compatibles avec les paramètres d'arrivée de fR.

La relation RS dont la définition est (SUITE [fnum1 fnum2 …] fR) est une relation d'arité n+1 dont les nf premiers paramètres correspondent aux paramètres de la fonction fR, les (n-nf) suivants aux autres paramètres de R et le dernier à l'indice de l'élément recherché. (RS ?x1 ?x2 … ?xn ?rang) est donc vrai ssi, l'indice dans l'ensemble (fR ?x1 ?x2 … ?xnf) de l'élément [?xnf+1 … ?xn] est égal à ?rang. La comparaison de deux éléments de l'ensemble est fait grâce aux fonctions fnumi qui correspondent à des critères de tri ordonnés. Si les deux éléments à comparer sont x1 et x2, le résultat de la comparaison est celui correspondant à la première fonction $fnum_i$ rendant un résultat différent pour x_1 et x_2 et est vrai ssi $fnum_i(x_1) < fnum_i(x_2)$.

Un exemple simple est l'ordonnancement des valeurs possibles pour ?f(?x). La définition interne est (SUITE [fId] F12_valeur_possible) où fId est la fonction identité. Si la relation s'appelle Tvaleur_possible, alors (Tvaleur_possible ?f ?x ?v 1) est vrai si ?v est la plus petite valeur pour ?f(?x).

10.2. définition interne des règles

La définition interne d'une règle est de la forme (ACTCOND paquet prio R ACT) où paquet est le paquet de règles auquel appartient cette règle, prio est un entier qui indique la priorité de la règle, R est une relation correspondant à la condition et ACT correspond aux actions à exécuter. Nous voyons en 10.3 un exemple de définition interne de règle.

10.3. passage de la définition externe à la définition interne

Lors de la première utilisation d'une relation, le système doit, à partir de la représentation externe de la relation, produire la représentation interne (ceci n'est pas refait lors des utilisations suivantes car la représentation interne peut être stockée sur fichier avec la représentation externe comme nous le verrons en 11.3).

Notre objectif ici n'est pas de décrire en détail cet algorithme de passage mais d'en donner une idée générale à partir d'un exemple.

L'exemple choisi est la construction de la représentation interne de la règle suivante:

```
(DEF_REGLE "AffecterValeur" CRYPT
SI      (depart_application ?x ?f)
        (valeur_possible ?f ?x ?v)
        (NON (valeur_affectee ?f ?x ? ) )
```

---

[1] ou à l'unique élément si fR est monovaluée

```
           (==  #(valeur_possible ?f ?x ? )   1)
           (hypothese_courante ?h)
 ALORS (AFFIRMER(valeur_affectee_hyp ?f ?x ?v ?h)))
```

Le système génère une relation dont la définition externe correspond à la conjonction des prémisses de la règle et dont les paramètres sont les variables de la conclusion qui apparaissent en prémisses. Cette relation, dont le nom généré est Relreg_AffecterValeur, a comme définition externe:

```
(DEF_RELATION " Relreg_AffecterValeur "
      [[?f application][?x objet][?v objet][?h hypothese]]
VRAISI(Et       (depart_application ?x ?f)
                (valeur_possible ?f ?x ?v)
                (NON (valeur_affectee ?f ?x ? ) )
                (==  #(valeur_possible ?f ?x ? )   1)
                (hypothese_courante ?h))

)
```

et la définition interne de la règle est alors

```
(ACTCOND paquet_CRYPT 0
      RRelreg_AffecterValeur
      [[?A ?X ?Y ?H]
       (AFFIRMER (valeur_affectee_hyp ?A ?X ?Y ?H))
      ]).
```

La construction de la définition interne de la relation Relreg_AffecterValeur consiste à remplacer toutes les expressions non simples de la conjonction par des expressions simples en générant de nouvelles relations.

Les troisième et quatrième expressions de la conjonction ne sont pas des expressions simples. Le traitement de la troisième expression est le suivant:
Tout d'abord l'expression est expansée et l'on obtient l'expression:
(== (CUMUL Fun (valeur_possible ?f ?x ?) 0)
Le troisième argument de l'expression (valeur_possible ?f ?x ?) étant une variable anonyme, l'ensemble que l'on considère ici est la valeur de (F12_valeur_possible ?f ?x).
Le système génère la relation qu'il nomme CARDF12_valeur_possible dont la définition interne est (CUMUL Fun F12_valeur_possible).

On obtient ainsi, après un traitement similaire pour l'expression suivante de la conjonction, la définition interne suivante pour Relreg_AffecterValeur:

```
(CONJONCTION [?A ?X ?Y ?H]
       []
      [(depart_application ?X ?A)
      (F1_RUNF12_valeur_possible ?A ?X 1)
      (F1_RUNF12_valeur_affectee ?A ?X 0)
      (valeur_possible ?A ?X ?Y)
      (hypothese_courante ?H)])
```

Plus généralement, la définition interne des conjonctions est obtenue à partir des définitions externes en remplaçant les expressions composées par des expressions simples, ce remplacement étant obtenu grâce à la génération de relations dérivées.

10.4. Le réseau de relations

Le réseau de relations est un réseau dont les nœuds sont les relations et les prédécesseurs d'un nœud sont les relations utilisées dans la définition interne de cette relation. Les relations de base n'ont pas de prédécesseurs et constituent donc les entrées de ce réseau, les règles étant quant à elles des feuilles du réseau
La figure 11 montre une partie du réseau correspondant à l'exemple que nous avons traité en 10.3.
Il existe six catégories de nœuds correspondant respectivement aux relations de base, aux quatre catégories de relations dérivées et aux règles.

Ce réseau permet au système Sepiar 2 d'utiliser la base de connaissances pour obtenir des informations et pour propager des modifications de la base de connaissances:

- Lorsque le système souhaite connaître par exemple les instances d'une relation, il transmet cette demande au nœud associé à la relation. Selon les directives de paramétrages, s'il a l'information, il la retourne immédiatement sinon il la retrouve en émettant si besoin des requêtes à ses prédécesseurs.
- Quand une modification survient dans la base de faits (ajout ou suppression d'une instance d'une relation de base), la modification est transmise au nœud de la relation de base. Celui-ci transmet ensuite à ses successeurs des informations dont le contenu dépend du paramétrage.

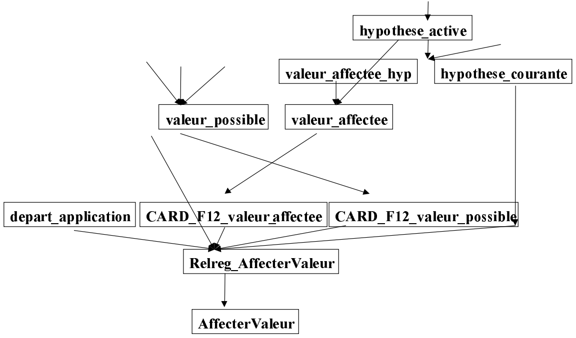


figure 11 : le réseau de relations

11. paramétrabilité

Le système Sepiar 2 a pour objectif d'adapter dynamiquement ses méthodes d'utilisation des connaissances. Cet objectif, pour être atteint convenablement, nécessite que le système ait des capacités de paramétrage importantes : il ne s'agit pas d'ajuster des paramètres numériques mais bien de mettre en œuvre au cas par cas des méthodes très différentes d'utilisation des connaissances. Chaque règle, chaque relation doit pouvoir être utilisée différemment des autres selon ses caractéristiques statiques et dynamiques.

Les paramétrages sont dans Sépiar 2 des objets à part entière. A chaque relation peuvent être associés plusieurs paramétrages dont le paramétrage actif qui permet au système de savoir comment la relation doit être utilisée. Nous décrivons en 11.1 les possibilités de paramétrage du système.

Le système Sepiar2 permet, comme nous le montrons en 11.2 sur des exemples, de pouvoir définir de nouveaux paramétrages et de pouvoir changer de paramétrage actif à tout moment.

La totalité des caractéristiques d'une relation, aussi bien les représentations externes, définitions internes, algorithmes et autres caractéristiques d'implémentation peuvent être affichés sous une forme récupérable par le système lors d'une session ultérieure. Nous donnons un aperçu de cet affichage en 11.3.

L'utilisation d'une relation est réalisée par l'exécution de nombreux algorithmes générés par le système et dépendant étroitement des paramétrages. Nous donnons en 11.4 les différentes catégories d'algorithmes et donnons quelques exemples d'algorithmes générés.

11.1. Les possibilités de paramétrage

Un paramétrage de relation définit une façon dont la relation peut être utilisée. Les briques de base permettant de définir un paramétrage sont appelées des conseils de paramétrage. Un conseil de paramétrage permet de préconiser une option de paramétrage ayant trait à:

- la mémorisation des relations et la façon de les mémoriser.
- l'ordre dans lequel sont envisagées les différentes expressions d'une conjonction.
- la façon et le moment de propager les modifications dans le réseau
- la définition interne des relations dérivées

11.1.1. Mémorisation des relations

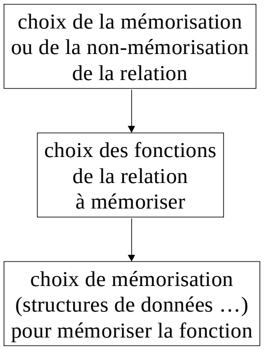


figure 13: *Les trois niveaux de choix pour la mémorisation*

Pour chacune des relations, plusieurs choix doivent être effectués. Le principal choix est celui de la mémorisation de la relation. Alors que cette mémorisation est obligatoire pour les relations de bases, elle n'est qu'une possibilité pour les relations dérivées, les instances pouvant être le plus souvent retrouvées par utilisation de la définition.

Le fait de pouvoir décider individuellement pour chaque relation de sa mémorisation est une caractéristique particulièrement intéressante du système. Nous pouvons en particulier reproduire les choix faits dans d'autres systèmes : mémoriser toutes les relations correspond à l'algorithme RETE [FOR82], mémoriser les règles uniquement correspond à TREAT [MIR87] tandis que l'absence de mémorisation (exceptée des relations de base) correspond à Snark [LAU86]. L'avantage évident de la mémorisation est d'accéder quasi directement aux valeurs recherchées tandis que dans le cas contraire, la recherche des valeurs passe par l'utilisation de la définition. Ceci dit, la mémorisation a un coût et si la relation n'est pas souvent utilisée, ce coût peut être parfois beaucoup plus élevé que celui de l'utilisation de la relation sans mémorisation.

Le deuxième choix à faire est, dans le cas où l'on a décidé de mémoriser, de savoir quelles sont les fonctions intéressantes à mémoriser. Prenons l'exemple de la relation ternaire (valeur_possible ?f ?x ?v) vrai si ?v est une valeur possible pour ?f(?x). Si l'on a souvent besoin des valeurs possibles des fonctions en un point, il sera peut-être intéressant de mémoriser F12_valeur_possible de telle sorte que pour une valeur de ?f et ?x, on ait directement accès à l'ensemble des valeurs possibles de ?f(?x). Si, par contre, on a besoin de connaître l'ensemble des antécédents ?x de la fonction ?f encore possible pour une valeur ?v, c'est la fonction F13_valeur_possible qui sera à envisager. Pour une relation mémorisée, une ou plusieurs fonctions peuvent être mémorisées.

Le troisième choix est celui de la structure de mémorisation à utiliser pour mémoriser une fonction. Doit-on par exemple préférer une liste pour sa simplicité d'utilisation, un arbre binaire de recherche pour ses performances et sa souplesse ou une table de hachage d'une certaine dimension ?

La figure 17 donne les types de conseils correspondants à ces trois niveaux de choix.

11.1.2. Ordonnancement des expressions d'une conjonction

Lorsque la définition d'une conjonction doit être utilisée, que ce soit pour déterminer l'ensemble des instances ou pour déterminer la valeur d'une fonction en un point donné, les expressions de la conjonction sont utilisées dans un ordre déterminé par l'algorithme d'ordonnancement du compilateur. Cette ordre est important car l'efficacité en dépend souvent de façon très étroite.

L'algorithme d'ordonnancement (figure 13) construit progressivement la liste ordonnée en n étapes (où n désigne le nombre d'expressions de la conjonction), la $i^{\text{ième}}$ étape consistant à choisir la $i^{\text{ième}}$ expression de la liste. A chaque étape, le choix est réalisé en fonction des variables déjà déterminées par les étapes précédentes (ou déjà connues initialement) et des expressions déjà choisies.

```
//Expressions désigne les expressions de la conjonction non encore utilisées
Expressions = les expressions de la conjonction
// VariablesConnues les variables déjà determinées par les étapes précédentes (ou déjà connues initialement)
VariablesConnues = variables connues initialement
//ExpressionsDéjàChoisies les expressions déjà choisies.
ExpressionsDéjàChoisies = {}

ListeOrdonnée=[ ]
répéter  n  fois //n désigne le nombre total d'expressions
  E = choisir une expression connaissant ExpressionsDéjàChoisies et VariablesConnues
  ListeOrdonnée = ListeOrdonnée + E
  ExpressionsDéjàChoisies=ExpressionsDéjàChoisies+E
  VariablesConnues=VariablesConnues U variables de E
```

figure 13 : *l'algorithme d'ordonnancement*

Le paramétrage ne consiste pas à donner la séquence ordonnée des expressions mais à donner des conseils sur les choix de l'expression à considérer à chaque étape de cet algorithme.
Un tel conseil est une méta-connaissance faisant intervenir une liste d'expressions (LE), une liste de variables (LV) et une liste ordonnée d'expressions (LO) et peut s'énoncer de la façon suivante:

Si toutes les expressions de LE ont déjà été utilisées
Si toutes les variables de LV ont déjà une valeur connue
Alors
les meilleurs choix sont les éléments (ordonnés) de la liste d'expressions LO

Considérons par exemple la définition interne de la relation Relreg_AffecterValeur (10.2):

```
 (CONJONCTION [?A ?X ?Y ?H]
          []
 1        [(depart_application ?X ?A)
 2        (F1_RUNF12_valeur_possible ?A ?X 1)
 3        (F1_RUNF12_valeur_affectee ?A ?X 0)
 4        (valeur_possible ?A ?X ?Y)
 5        (hypothese_courante ?H)])
```

et les trois conseils suivants:

```
// dans tous les cas , un bon choix est la première expression
 [    [ ]    [ ]    [1]]
// Si ?A et ?X sont connues et que l'expression 1 est déjà utilisée, considérer par ordre d'intérêt les
expressions de numéro 2, 3 et 4
 [  [?A ?X] [1] [2 3 4]]
// Si ?A  ?X et ?Y sont connues, considérer l'expression 5
 [  [?A ?X ?Y] [] [5]]
```

Si aucune variable n'est connue initialement, l'ordre choisi par l'algorithme d'ordonnancement est 1 (1er conseil) , 2 (2ème conseil), 3 (2ème conseil), 4 (2ème conseil) et 5 (3ème conseil).

11.1.3. Propagation des modifications

Les choix de paramétrisation pour la propagation sont effectués pour chaque relation et concernent d'une part le moment où la propagation est effectuée et d'autre part la façon dont cette propagation est effectuée.
Les instances d'une relation dérivée sont susceptibles d'être modifiées si les instances de l'une des relations mère[1] sont modifiées. C'est pourquoi toute modification intervenant dans la composition d'une relation (qu'elle soit mémorisée ou non) doit être propagée aux relations filles.
Lorsqu'une relation reçoit de l'une de ses relations mères une information de propagation, elle peut:

---

[1] une relation mère d'une relation R est une relation qui intervient dans la définition interne de R et qui est donc un prédecesseur de R dans le réseau

- traiter immédiatement cette information en mettant son ensemble d'instances à jour et en propageant ses modifications aux relations filles : c'est la propagation immédiate.
- stocker cette information pour ne la traiter que plus tard par nécessité : c'est la propagation différée Tant que ce traitement n'est pas effectuée, la relation est invalide (son ensemble d'instances n'est plus à jour) et toutes les relations filles sont également invalides.

La propagation différée apporte souvent des améliorations notables dans l'efficacité car elle évite la répétition de calculs inutiles : par exemple, si la même information est transmise deux fois, la propagation différée peut permettre de ne la traiter qu'une fois. L'inconvénient de ce mode de propagation est la nécessité du stockage de l'information à propager.

```
Une information arrive à l'entrée du nœud de la relation R:
si la relation est à propagation différée alors
-   stocker cette information
-   noter que cette relation est invalide
sinon si la relation R est mémorisée
    alors  mettre à jour R
           transmettre les modifications
    sinon  transmettre les modifications possibles de R

Une relation R devient invalide:
-   noter que chacune des relations filles de R devient invalide

Pour valider une relation R
    valider les prédécesseurs de R
    prendre en compte toutes les informations stockées et transmettre aux successeurs les modifications

Pour prendre en compte toutes les informations stockées
    si le nombre d'informations stockées est supérieure au seuil
        alors remettre entièrement à jour le nœud
        sinon traiter une à une les informations stockées
```

figure 14 : *principes de fonctionnement du réseau*

La demande de validation d'une relation est le plus souvent provoquée par la réaction en chaîne consécutive à la demande de validation d'une règle. Lorsqu'une relation à propagation différée doit être validée, deux façons de procéder sont possibles:

- les informations stockées sont traitées une à une
- la relation est entièrement remise à jour indépendamment des informations arrivées.

Le choix entre ces deux façons de procéder dépend d'un nombre paramétrable pour chaque relation appelé seuil. Si le seuil est un nombre strictement positif et que le nombre d'informations stockées est supérieur ou égal à ce seuil, alors la relation est entièrement remise à jour sinon les informations sont traitées une à une.

11.1.4. Définitions interne des relations dérivées

Il existe en général plusieurs définitions internes possibles pour une même relation. Nous avons vu au chapitre précédent comment la première définition interne était créée par le système à partir de la représentation externe. D'autres définitions peuvent être considérées et utilisées par le système.

Cette possibilité est en particulier intéressante pour les conjonctions: il peut être judicieux de regrouper des expressions au sein d'une nouvelle relation ou au contraire d'expanser une expression dont la relation est une conjonction.

Considérons par exemple une conjonction R composée de quatre expressions. La relation peut être utilisée de cette façon mais plusieurs autres possibilités peuvent être envisagées: Par exemple, les deux premières expressions peuvent être regroupées en définissant une nouvelle relation S avec comme définition interne une conjonction de ces deux expressions. La définition interne obtenue pour R est une conjonction des 2 expressions restantes et de la nouvelle expression utilisant la relation S.

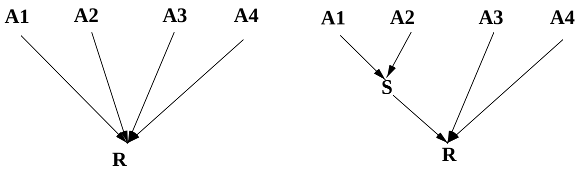


figure 15 : *deux définitions possibles pour R*

11.1.5. conseils et réponses aux questions du compilateur

Les conseils de paramétrage ont pour vocation d'être utilisés par le compilateur lorsqu'il produit la représentation interne (réseau, algorithmes). Seuls les conseils des paramétrage actifs sont susceptibles d'être pris en compte. Plus précisément, le compilateur du système Sépiar lorsqu'il désire compiler une relation se pose un certain nombre de questions : comment mémoriser la relation, comment propager les informations etc… . La réponse lui est fournie de la façon suivante:

- si la réponse est obligatoire (par exemple une relation de base doit obligatoirement être mémorisée) alors rendre cette réponse
- sinon si il existe un conseil du paramétrage actif permettant de répondre à la question (par exemple, s'il y a un conseil [MEMOREL NON], la réponse à la question "puis-je mémoriser la relation " est négative), rendre la valeur préconisée par le conseil

S'il n'y a pas de réponse obligatoire et qu'il n'y a pas de conseil, alors le système donne une réponse qui correspond à un conseil par défaut.

Ces conseils par défaut sont les suivants:

- la mémorisation est conseillée pour les conjonctions et les disjonctions ; elle est déconseillée pour les autres,
- pour les relations mémorisées, le système conseille la mémorisation des fonctions dont les algorithmes ont besoin,
- la propagation est différée pour toutes les relations du réseau,
- la définition interne est déduite "naturellement" de la définition externe,
- Les expressions de la conjonction sont ordonnées par utilisation de l'algorithme d'ordonnancement vu précédemment. A chaque étape, le choix par défaut est, par ordre de priorité:
    1. une expression dont toutes les variables sont connues
    2. une expression où une seule valeur est possible pour les variables (fonction monovaluée)
    3. la première expression non encore utilisée en considérant l'ordre où elles apparaissent dans la définition interne.

Quand une base de connaissances est utilisée pour la première fois, le système Sepiar2 génère pour chaque relation un paramétrage par défaut dont l'ensemble des conseils est vide. Ce sont donc les conseils par défaut qui sont utilisés au démarrage. Cette paramétrisation par défaut n'est pas très importante car elle est temporaire. Cependant, elle a été conçue pour permettre avant tout apprentissage une utilisation aussi efficace que possible de la base de connaissances.

La figure 17 montre les principaux types de conseils de paramétrage avec la question à laquelle ils permettent de répondre et la syntaxe utilisée pour ces conseils lors de la définition des paramétrages.

conseils de paramétrages modifiant la mémorisation de la relation

Est-il conseillé de mémoriser la relation ?
[MEMOREL OUI/NON]

Si la fonction est utilisée par un algorithme, puis la mémoriser ? [MEMOFONC <fonction> OUI/NON]

Si je mémorise une fonction, quel stockage dois-je utiliser ?
[STOCKAGE <fonction> <stockage>]

conseils pour l'ordre des expressions d'une conjonction

Connaissant certaines variables et ayant déjà utilisé certaines expressions de cette conjonction, quelle expression dois-je choisir maintenant ?
[CHOIXEXPRESSIONS [[<variable>*][<expression>*]
[<expression>*]]

conseil pour la propagation de l'information dans le réseau

Dois-je différer la propagation ?
[RETARD OUI/NON]

Quel est la valeur du seuil?
[SEUIL<entier>]

conseils de paramétrage proposant une autre définition interne

Quelle est la définition interne à utiliser ?

Cas du regroupement d'expressions

[DEFREL [def <n° def de base> <premisse>+]]

Cas de l'expansion de la définition d'une relation d'une expression

[DEFREL [def <n° def de base> <premisse>]]

figure 17 : *principaux types de conseils de paramétrage*

11.2. Définition et activation de paramétrages

En pratique, un paramétrage est défini à partir d'un paramétrage de base et d'un ensemble de conseils. Les conseils du nouveau paramétrage sont les conseils de cet ensemble plus les conseils du paramétrage de base qui ne rentre pas en conflit avec les nouveaux.

L'action permettant de définir un paramétrage est nommé DEF_PARAMETRAGE et prend trois paramètres:

la relation

le paramétrage de base. /*ou son numéro, local à la relation, le premier paramétrage (le paramétrage par défaut) étant numéroté 0*/

la liste des nouveaux conseils

L'exemple suivant:

```
# (DEF_PARAMETRAGE Rvaleur_affectee 0 [[MEMOREL non]])
 = %[P Rvaleur_affectee 1]
```

permet de définir un nouveau paramétrage pour la relation valeur_affectee. Le paramétrage de référence est le paramétrage numéroté 0 (le premier créé pour cette relation) et il y a un conseil suggérant la non mémorisation de la relation. La réponse du système indique ici que le paramétrage défini a le numéro 1.

Le changement de paramétrage actif est réalisé par l'action SET_PARAMETRAGE_ACTIF dont les deux paramètres sont la relation concernée et le nouveau paramétrage actif(ou son numéro). Ce changement est effectif immédiatement.

L'évaluation de l'expression suivante permet de rendre actif le paramétrage que nous venons de créer.

```
#(SET_PARAMETRAGE_ACTIF Rvaleur_affectee 1)
 = %[P Rvaleur_affectee 1]
```

11.3. description des relations

L'action DECRIRE_RELATION permet dans Sepiar2 d'obtenir une description complète d'une relation. Outre qu'elle reflète l'ensemble des connaissances que le système a sur la relation, elle peut, lors d'une

session ultérieure, être relu par le système permettant ainsi à la relation de se retrouver dans les mêmes conditions d'utilisation et les mêmes possibilités de modification de paramétrage.
Nous décrivons rapidement dans cette section les différentes parties de cette description en faisant référence à l'exemple de la figure 17.

11.3.1. DEF_RELATION

Les premières lignes reprennent la définition donnée par l'utilisateur : le nom de la relation, ses paramètres avec leur classe.

11.3.2. PROPRIETES

Deux types de propriétés peuvent être indiquées ici: les propriétés de monovaluation de certaines fonctions (comme en figure 8) et des propriétés de calculabilité par le système des fonctions associées à la relation. Ici, cela signifie que la fonction sans paramètre est calculable : le système sait obtenir l'ensemble des instances de la relation Rvaleur_affectée.

11.3.3. DEFINITION

Ce sont les définitions internes de la relation. Elles sont, comme les paramétrages, numérotées localement à la relation. Dans cet exemple, il n'y a qu'une définition numéroté 0 qui correspond à la définition générée à partir de la définition externe.

11.3.4. PARAMETRAGE

Nous avons ici deux paramétrages, le premier numéroté 0, est le paramétrage par défaut et le numéro 1 est celui créé en 11.2.

11.3.5. IMPLEMENTATION

Cette partie, que nous n'expliquons pas ici, correspond aux caractéristiques d'implémentation de la relation : fonctions effectivement mémorisées, numéro des algorithmes compilés, etc.

11.3.6. VRAISI

Nous retrouvons la définition externe analogue à celle qu'a donnée l'utilisateur.

```
# (DECRIRE_RELATION Rvaleur_affectee)
(DEF_RELATION "valeur_affectee"[[?appl application] [?dep objet] [?arr objet]]
  PROPRIETES [[] CALC]
  DEFINITION [0 (Et [?appl ?dep ?arr] []
           [(hypothese_active ?hyp)
            (valeur_affectee_hyp ?appl ?dep ?arr ?hyp)])
       "initiale"]
  PARAMETRAGE
    // numéro 0 avec la définition numéro 0
   [0 0]
    // et le numéro 1 avec la définition 0 et le conseil de non mémorisation , le "+" signale que c'est le
paramétrage actif
   ["+" 1 0 [MEMOREL RELATION "non"]]
  IMPLEMENTATION
       [[]  "MEMO" [[[STabr MULTI] 154 champZ]] /**/ ]
       [[1 2]         "MEMO" [[[STabr MONO] 16 champN]] /**/ ]
champN]] "COMPILE" [[Rgetm1 "C" 6838]]]
       [[1]            "MEMO" [[[STabr MULTI] 9 champNUP]] /**/ ]
       [RELATION "RETARD" ["majcompos" 0 2] /**/
            "ORDREPREM" [[[] [2 1]] [[?dep ?appl] [1 2]]]]
/**/
  VRAISI (Et    [?hyp hypothese]
            (valeur_affectee_hyp ?appl ?dep ?arr ?hyp)
            (hypothese_active ?hyp)))
```

figure 17

11.4. Algorithmes générés

Pour des raisons d'efficacité, des algorithmes propres à chaque relation sont générés. Ils sont dépendants du paramétrage de la relation et parfois de ceux de certaines relations environnantes dans le réseau.

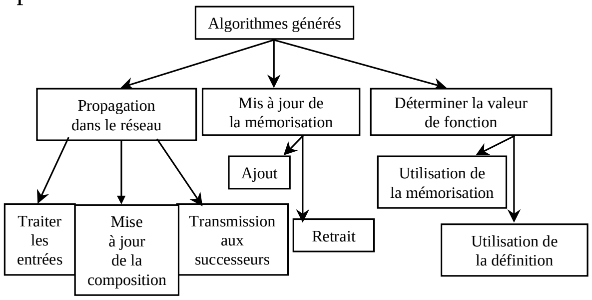


figure 19 : catégories d'algorithmes générés

Trois catégories d'algorithmes sont générées (figure 19). La première catégorie concerne la propagation des modifications dans le réseau, la deuxième catégorie concerne les algorithmes permettant de modifier la composition des relations et la troisième catégorie concerne les algorithmes qui retourne une valeur ou un ensemble de valeurs pour les fonctions associées aux relations.

11.4.1. Propagation dans le réseau

Les algorithmes liés à la propagation permettent:

- de réagir aux arrivées d'informations

à chaque entrée du nœud, un algorithme permet de déterminer ce qui doit être fait. En particulier, il sera différent selon le mode de propagation (immédiat ou différé)

- de mettre à jour la composition de la relation

des algorithmes permettent de déterminer le mode de mise à jour de la composition de la relation. Cette mise à jour est réalisée en comparant en certains points la mémorisation effective avec les vrais valeurs (obtenues par utilisation de la définition).

Prenons l'exemple de la relation valeur_possible. Supposons que valeur_possible soit une relation mémorisée et que, grâce aux propagations de modifications, on sache que l'ensemble (F12_valeur_possible ?v1 ?v2) a été modifié (pour des valeurs connues de ?v1 et ?v2). Le système pour mettre à jour la mémorisation compare les valeurs de (F12_valeur_possible ?v1 ?v2) obtenus en utilisant la mémorisation d'une part et la définition d'autre part. La différence entre les deux résultats obtenus permet la mise à jour de la mémorisation.

- de transmettre les informations de propagation

  ce qui correspond, pour les cas les plus classiques, à la transmission à chacun des successeurs des instances ajoutées ou supprimées.

11.4.2. Modification de la composition des relations

La modification de la composition des relations se traduit en particulier par l'ajout ou le retrait d'une instance de la relation. Les algorithmes correspondants dépendent des fonctions mémorisées.

Supposons par exemple que la relation valeur_possible soit mémorisée et que les fonctions mémorisées pour cette relation soient F12_valeur_possible et F_valeur_possible. L'ajout (resp. le retrait) d'une instance [!?f ?x ?v!] nécessite l'ajout (resp. le retrait) de l'élément ?v de l'ensemble (F12_valeur_possible ?f ?x) et l'ajout (resp. le retrait) du triplet [!?f ?x ?v!] de l'ensemble (F_valeur_possible).

11.4.3. Déterminer les valeurs de fonctions

La valeur des fonctions peut être déterminée en utilisant soit la mémorisation de la relation soit en utilisant la définition de la relation.

<u>utilisation de la mémorisation</u>

Supposons que le système ait besoin des valeurs de ?y pour lesquels (valeur_possible ?f ?x ?y) est vrai. Le système cherche donc à déterminer la valeur de F12_valeur_possible(?f ?x). L'algorithme recherchant cette valeur dépend de plusieurs facteurs.

Si, par exemple, la fonction F12_valeur_possible est mémorisée, l'algorithme généré va rendre l'ensemble accessible directement grâce à cette mémorisation.

Supposons maintenant que F12_valeur_possible ne soit pas mémorisée, et que parmi les fonctions mémorisées de la relation valeur_possible, F1_valeur_possible soit mémorisée. Si le système choisit de déterminer les valeurs pour F12_valeur_possible à partir des valeurs de F1_valeur_possible et si la structure choisie pour mémoriser les ensembles correspondant à F1_valeur_possible est l'arbre binaire équilibré alors l'algorithme généré ressemble à:

```
Pour déterminer (F12_valeur_possible  ?f ?x) en utilisant la mémorisation
    Res = ensemble vide
    En parcourant les éléments (?x1 ?v) de l'arbre binaire                    F1_valeur_possible(?f)
        si ?x1 est égal à ?x alors ajouter ?y à Res
    le résultat est Res
```

11.4.4. utilisation de la définition

La définition d'une relation dérivée est utilisée dans deux cas: pour trouver des valeurs d'une fonction non mémorisée et pour mettre à jour des relations mémorisées.

Considérons la définition interne de la relation Relreg_AffecterValeur (10.2):

```
(CONJONCTION [?A ?X ?Y ?H]
   []
  [(depart_application ?X ?A)
  (F1_RUNF12_valeur_possible ?A ?X 1)
  (F1_RUNF12_valeur_affectee ?A ?X 0)
  (valeur_possible ?A ?X ?Y)
  (hypothese_courante ?H)])
```

Si l'on désire déterminer grâce à cette définition l'ensemble des instances de la relation, nous devons donc déterminer tous les quadruplets [?A ?X ?Y ?H] tels que les cinq expressions de la conjonction soient vérifiées. L'algorithme généré dépend de l'ordre dans lequel les expressions doivent être considérées. Si aucun conseil n'est donné, les choix par défaut conduisent à considérer dans ce cas les expressions dans l'ordre où elles apparaissent dans la définition interne ce qui donne l'algorithme suivant:

```
Lres={}
pour tout [?X ?A] de (F_depart_application)
  Si (F1_RUNF12_valeur_possible ?A ?X 1)
    Si (F1_RUNF12_valeur_affectee ?A ?X 0)
      pour tout ?Y de (F12_valeur_possible ?A ?X)
        pour tout ?H de (F_hypothese_courante)
          ajouter [?A ?X ?Y ?H] à Lres
```

Si l'objectif est de connaître la valeur de (F14_Relreg_AffecterValeur ?A ?H), ?A et ?H étant connus, si aucun conseil n'est donné, l'algorithme est:

```
Lres={}
Si (hypothese_courante ?H)
  pour tout ?X de (F2_depart_application ?A)
    Si (F1_RUNF12_valeur_possible ?A ?X 1)
      Si (F1_RUNF12_valeur_affectee ?A ?X 0)
        pour tout ?Y de (F12_valeur_possible ?A ?X)
          ajouter [?X ?Y] à Lres
```

11.4.5. algorithmes et changement de paramétrage

Si l'on considère l'exemple ci-dessus, on constate que l'algorithme dépend de l'ordre dans lequel les expressions de la conjonction sont considérés. Si cet ordre est modifié, il faut réécrire l'algorithme. Plus généralement, tout changement de paramétrage se traduit par la regénération de certains algorithmes afin que les conseils du nouveau paramétrage soient pris en compte.

11.4.6. compilation des algorithmes en C

les algorithmes sont générés dans le langage procédural de Sepiar2 et sont interprétés par l'interpréteur de ce langage intégré au système. Ils peuvent également être compilés dynamiquement en C.

12. Paramétrage automatique

Le fonctionnement normal du système Sepiar2 est un fonctionnement autonome sans aucune intervention de l'utilisateur. Il peut fonctionner de cette manière selon deux modes : le mode résolution et le mode apprentissage.

En mode résolution, étant donnés une base de connaissances et l'ensemble des paramétrages actifs des règles et relations, le système résout de façon classique et sans apprentissage le problème de base. En particulier, les paramétrages actifs ne sont pas modifiés et aucune trace de session n'est analysée. Les paramétrages actifs sont donc ceux fixés en début de session.

Le mode apprentissage, dont nous parlons dans ce chapitre, a pour objectif de déterminer des paramétrages actifs pertinents permettant une utilisation plus efficace de la base de connaissances. Cet apprentissage est réalisé grâce à une méta-expertise dont nous décrivons le principe en 12.1. Cette méta-expertise permet en particulier de générer des expériences dont certaines permettent de créer de nouveaux paramétrages (12.2 et 12.3) et d'autres de comparer plusieurs paramétrages (12.4). Après avoir regardé un extrait de session d'apprentissage(12.6), nous montrons en 12.7 les résultats obtenus par le système Sepiar2 avec cette méta-expertise.

12.1. La méta-expertise

Alors qu'en mode résolution, seule l'expertise de base est utilisée, en mode apprentissage, deux expertises sont utilisées en alternance : l'expertise de base et la méta-expertise. La méta-expertise est exécutée à intervalle régulier grâce à l'utilisation d'interruptions système. Son rôle principal est de changer les paramétrages actifs pour permettre un fonctionnement plus efficace de l'expertise de base. Comme indiqué sur la figure 20, elle procède en détectant les problèmes puis en générant des expériences pour trouver des informations pertinentes sur l'expertise de base. Ces expériences sont réalisées lors des utilisations de l'expertise de base qui suivent. Lorsqu'une expérience est terminée, elle est analysée, des paramétrages sont éventuellement créés et des changements de paramétrages actifs peuvent être effectués.

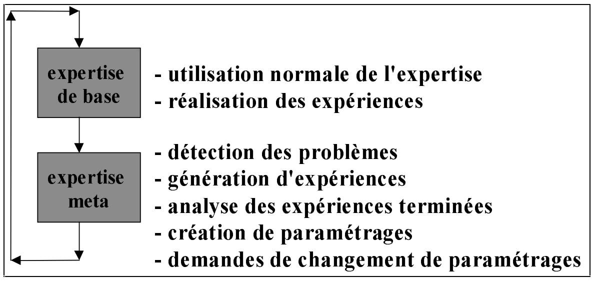


figure 20 : *alternance des expertises*

Afin que la méta-expertise ait le temps de faire ses expériences et de modifier en conséquence des paramétrages actifs, le problème de base peut être résolu plusieurs fois lors d'une même session d'apprentissage.

12.1.1. Détection des problèmes

L'objectif est de s'attaquer en priorité aux problèmes d'inefficacité les plus importants. En mode apprentissage, le temps de traitement de chaque relation est noté et la méta-expertise considère comme prioritaire les relations pour lesquelles ce temps est élevé. Plus précisément, on considère à un moment donné le temps T total d'utilisation de l'expertise de base. Si n est le nombre total de relations de cette expertise, les relations pour lesquelles le temps est supérieur à T/n sont considérées comme prioritaire. Ce critère, simple à mettre en œuvre, est satisfaisant car il évite à la méta-expertise de s'intéresser aux relations pour lesquelles le temps d'utilisation est très faible et il lui permet de se concentrer sur les relations où un gain important peut être espéré.

12.1.2. Nouveaux paramétrages envisagés

Les nouveaux paramétrages considérés par la méta-expertise sont obtenus en ajoutant de nouveaux conseils aux paramétrages actifs.

Les conseils envisagés actuellement sont les suivants:

- ne pas mémoriser la relation
- regrouper des expressions d'une conjonction

- ordonnancer différemment les expressions d'une conjonction
- retarder la propagation avec un seuil donné
- créer une relation intermédiaire avec plus de paramètres

12.1.3. Génération d'expériences

La trace par défaut produite lors de l'exécution d'une session est très pauvre: le système ne note que des informations pouvant être obtenues facilement comme le temps CPU utilisé pour chaque relation.

Ce type d'information n'est pas du tout suffisant pour pouvoir trouver et modifier pertinemment les paramétrages actifs. C'est pourquoi la méta-expertise génère des expériences pour obtenir des informations sur l'expertise de base adaptées à ses besoins. Ces expériences peuvent être classées en deux catégories.

L'objectif de la première catégorie d'expériences est d'obtenir des informations pertinentes pour pouvoir générer de nouveaux paramétrages. Il en est ainsi dans deux cas que nous étudions :

- trouver un nouvel ordonnancement des expressions d'une conjonction
- regrouper des expressions d'une conjonction par la création d'une relation intermédiaire

L'objectif des expériences de la seconde catégorie est de tester un paramétrage envisagé. Avant de changer de paramétrage actif, une expérience est générée pour comparer le paramétrage actif et le paramétrage envisagé. Si cette comparaison est favorable au paramétrage envisagé, la méta-expertise demande le changement de paramétrage qui prend alors effet immédiatement.

12.2. Expériences pour trouver un nouvel ordonnancement des expressions d'une conjonction

Pour améliorer l'efficacité de l'utilisation d'une conjonction, une possibilité est de modifier l'ordre d'utilisation des expressions de cette conjonction. Nous avons vu en 11.1 que les conseils d'ordonnancement permettent d'influer sur cet ordre et l'objectif de ce type d'expérience est donc de trouver de tels conseils.

Pour trouver de tels conseils, Sépiar, lorsqu'il utilise la définition de la conjonction pour déterminer des valeurs, va procéder différemment: au lieu d'utiliser un ordre figé pour les expressions, il va déterminer dynamiquement un ordre. Pour chacune des expressions candidates à un moment donné, le système note le nombre d'instanciations possibles et en particulier celle pour laquelle ce nombre est minimum. Il génère à ce moment un conseil indiquant la liste ordonnée des meilleurs choix (nombre de valeurs possibles proches du minimum).

Lorsque l'expérience est terminée, un certain nombre de conseils ont été créés. Cet ensemble de conseils est considéré comme intéressant s'il permet d'obtenir un ordonnancement des expressions différent de celui qui existait déjà. Dans ce cas, un nouveau paramétrage est créé incluant ces nouveaux conseils.

12.3. Expériences pour regrouper des expressions d'une conjonction

Considérons la conjonction R dont la représentation interne est schématisée en haut de la figure 21. C'est une conjonction de 5 expressions faisant intervenir respectivement les relations r1, r2, r3, r4 et r5. Une autre représentation interne possible est indiquée en bas de la figure: les expression 2, 4 et 5 ont été regroupées dans une nouvelle relation nr et la relation R est représentée comme une conjonction de 3 expressions.

Un tel regroupement peut être utile car il permet en mémorisant nr de mémoriser la jointure des expressions 2 4 et 5.

Toutefois, beaucoup de regroupement ne sont pas intéressants et peuvent même être néfaste.

Un cas où le regroupement peut être intéressant est celui où les relations des expressions regroupées sont des relations stables. Dans ce cas, la mémorisation de la jointure est intéressante car elle est peu souvent modifiée.

Lorsque le principe d'un regroupement est envisagé, la méta-expertise génère une expérience permettant d'estimer la stabilité des relations en entrée du nœud du réseau de la relation R.

Supposons par exemple que pendant un certain laps de temps, 30 modifications soient parvenues de r1, 1 de r2, 100 de r3, 4 de r4 et 1 de r5. Nous constatons que les arrivées en provenance de r2, r4 et r5 sont peu fréquentes et l'expérience permet dans un tel cas de préconiser le nouveau paramétrage utilisant nr dans la définition.

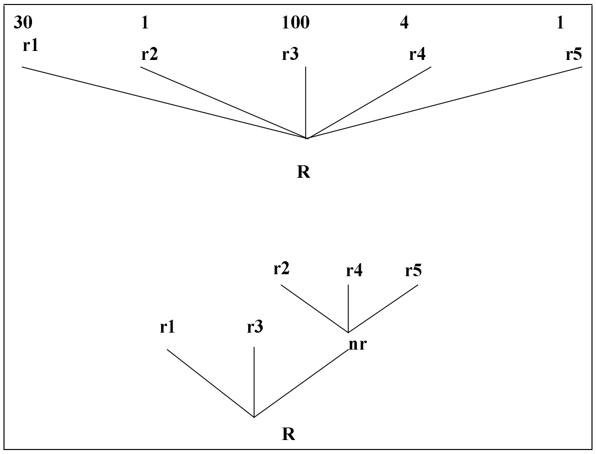


figure 21 : *regroupement des entrées stables*

Lorsque l'expérience est terminée, si des entrées stables sont détectées, alors une relation regroupant ces entrées est générée et un paramétrage considérant la nouvelle définition interne de R est créé.

12.4. Expériences de comparaisons de paramétrage

Nous venons de voir comment des nouveaux paramétrages préconisant des modifications de l'ordre des expressions d'une conjonction ou un regroupement d'expressions peuvent être générés. Dans certains cas, la création de paramétrages ne nécessitent même pas d'informations complémentaires. Il en est ainsi de paramétrages comme la non mémorisation d'une relation: il suffit alors de générer un paramétrage possédant comme conseil supplémentaire le conseil de non-mémorisation.
Dans tous les cas, quel que soit le paramétrage généré, il n'y a aucune assurance sur son efficacité: bien qu'il n'ait pas été créé au hasard, un vérification s'impose avant son activation et c'est là l'objectif des expériences de comparaisons de paramétrage.
Le principe de ces expériences est de tester simultanément le paramétrage actif et le paramétrage envisagé pour le remplacer.
Un cas particulier simple est celui où chacun des deux paramétrages utilise la même définition interne et conduisent à la mémorisation de la relation. Dans ce cas, la relation est dupliquée dans le réseau, l'original utilisant le paramétrage actif en vigueur et la duplication utilisant le paramétrage testé. Il suffit alors de noter le temps nécessaire pour l'utilisation des deux relations pour conclure quant à l'intérêt du paramétrage proposé.

La problème devient plus délicat lorsque la mémorisation n'est pas réalisée pour la relation et/ou sa duplication. En figure 22 est représenté à gauche une partie du réseau concernant une relation R. Elle est utilisée dans la définition interne de trois relations R1, R2 et R3. On suppose ici que certaines relations sont mémorisées (R,R1,R2,R5,R6,R7,R8) et d'autres non (R3 et R4). Si le nouveau paramétrage envisagé pour R conduit à ne plus mémoriser R, il faut faire attention car cela va affecter aussi les relation R1, R2 et R3: là où elles obtenaient directement les valeurs dont elles avaient besoin pour R, il va falloir avec ce nouveau paramétrage utiliser la définition de R. Cela affecte aussi des relations non directement liées à R mais liées à des relations non mémorisées affectées: ainsi R3 est une relation affectée par ce changement et, n'étant pas mémorisée, il s'en suit que les relations R6, R7 et R8 vont être affectées. Par contre, les relations R4 et R5 utilisent la relation R1 qui est mémorisée : elles ne sont dont pas affectées par cette modification.

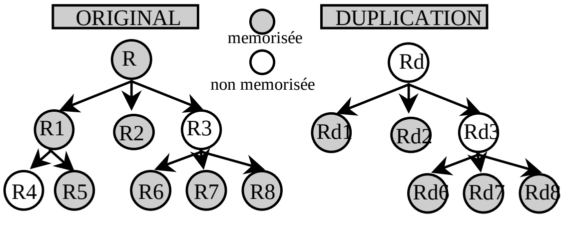


figure 22 : *comparaison par duplication*

La réalisation de l'expérience nécessite donc une duplication du sous-réseau composé de toutes les relations affectées, le paramétrage de la copie de R étant le paramétrage à tester.

Lors de l'utilisation de la base de connaissances, les temps CPU nécessaires pour les deux sous-réseaux sont comparés. Si le temps CPU nécessaire pour le sous-réseau dupliqué est moins élevé, le paramétrage testé devient alors le paramétrage actif.

12.5. Un exemple de méta-règle

A titre d'exemple, nous donnons en figure 23 une règle de la méta-expertise qui permet lorsqu'une expérience a permis de trouver un nouvel ordonnancement de générer l'expérience qui permet de créer l'expérience de comparaison de paramétrages.

```
 (DEF_REGLE "comparer_Ordres_De_Premisses" META
   // si ?r est une relation dérivée de paramétrage actif ?pa
   SI [?r relationderivee ^parametrageActif ?pa]
   // si ?expe est une expérience genre 1
   //  (i.e. pour déterminer des ordonnancement d'expressions)
   // pour la relation ?r , ?opt étant la liste des conseils d'ordonnancement
   // le bilan de l'expérience est positif (i.e. on a trouvé des conseils permet de déterminer des ordres
intéressants)
      [?expe experience ^genre 1 ^relation ?r ^option ?opt ^bilan "positif"]
   // on n'a pas encore fait d'experience de genre 2 (i.e. comparaison de parametrages)
   // pour la relation ?r avec ces méta-connaissances ?opt
      (NON[? experience ^genre 2 ^relation ?r ^option ?opt])
ALORS
     (CREER [[?e experience ^relation ?r
       ^parametrageInitial ?pa
       ^genre 2 ^etat "projet" ^option ?opt
      ^createur ?expe ^description "ordonnancement"]]
     (ECRIRE_EXPERIENCE " Etude d'un nouvel ordonnancement pour la relation " ?r)))
```

figure 23 : *une méta-règle générant une expérience de comparaison*

12.6. Un exemple de session d'apprentissage

```
// Ceci est une trace (très) partielle d'une expérimentation
// les commentaires ont été ajoutés à la main
// les affichages du programme sont en gras
// description des conditions de l'expérimentation
--------------------------------------
Probleme            :PBcryptaddition
Jeu de donnees       :SENDMORE
Nombre de repetition :6  // d'exécution de l'expertise de base
----------------------------
Paquet espionne : paquet_CRYPT
Experiences envisageables automatiquement {OrdreConjonction }
----------------------------
Execution des 6 sessions en mode apprentissage

Repetition n° 1/6 de (SESSION PBcryptaddition SENDMORE)

// la méta-expertise génère des expériences pour déterminer des conseils d'ordonnancement sur plusieurs
  relations dont trois que nous allons suivre ici

EXPERIENCE g8825 active 1 /* 1 pour ordonnancement*/RFK8125
EXPERIENCE g8901 active 1 RFK5398
EXPERIENCE g8923 active 1 RRelreg_UtiEqtAff
```

//L'expertise de base trouve la solution

:::: SOLUTION ::::R{8 }N{6 }O{0 }Y{2 }S{9 }E{5 }D{7 }M{1 }

Repetition n° 2/6 de (SESSION PBcryptaddition SENDMORE)

:::: SOLUTION ::::R{8 }N{6 }O{0 }Y{2 }S{9 }E{5 }D{7 }M{1 }

// la méta-expertise a été activée plusieurs fois pendant cette deuxième exécution de l'expertise de base mais les trois expériences qui nous intéressent ici ne sont pas encore terminées

----

Repetition n° 3/6 de (SESSION PBcryptaddition SENDMORE)

// l'expérience sur la relation RFK8125 se termine positivement : un ensemble de conseil d'ordonnancement intéressant a été trouvé

EXPERIENCE g8825 termine 1 RFK8125 P0 :positif option:%[P RFK8125 2]

// une expérience permettant de comparer le nouveau paramétrage et le paramétrage actif actuel est générée

EXPERIENCE g9486 active 2 /* comparaison*/ **RFK8125 P0 option:%[P RFK8125 2]**

//même séquence pour RRelreg_UtiEqtAff

EXPERIENCE g8923 termine 1 RRelreg_UtiEqtAff option:%[P RRelreg_UtiEqtAff 4]

EXPERIENCE g9990 active 2 RRelreg_UtiEqtAff P3 option:%[P RRelreg_UtiEqtAff 4]

// l'expérience sur la relation RFK5398 se termine négativement

EXPERIENCE g8901 termine 1 RFK5398 P3 :negatif

// Après comparaison, il s'avère que le nouveau paramétrage trouvé pour RFK8125 n'est pas intéressant

EXPERIENCE g9486 termine 2 RFK8125 P0 :Leca :negatif T:13631->15332 : {[!memo negatif!] }
option:%[P RFK8125 2]

Stat: temps:Total:54 #:14 [0..17]// 54/60$^{\text{ème}}$ avec l'actuel

Stat: temps_test:Total:70 #:14 [0..19] // 70/60$^{\text{ème}}$ pour le prétendant

// Après comparaison, il s'avère que le nouveau paramétrage trouvé pour RRelreg_UtiEqtAff est intéressant

EXPERIENCE g9990 termine 2 RRelreg_UtiEqtAff P3 :Leca :positif T:14489->15737 : {[!memo positif!] }
option:%[P RRelreg_UtiEqtAff 4]

Stat: temps:Total:135 #:14 [0..61]

Stat: temps_test:Total:43 #:20 [0..15]

// La méta expertise transmet au compilateur le changement de paramétrage actif de la relation RRelreg_UtiEqtAff. La modification est faite immédiatement.

(SET_PARAMETRAGE_ACTIF RRelreg_UtiEqtAff 4)

// et l'expérimentation se poursuit

12.7. Résultats

Les paramétrages obtenus lors d'une session d'apprentissage sont mémorisés et peuvent être réutilisés lors de sessions ultérieures soit en mode résolution, permettant ainsi des résolutions plus efficaces, soit en mode apprentissage pour trouver, en partant des paramétrages trouvés précédemment, un ensemble de paramétrage encore meilleur.

L'expérimentation consiste ici à enchaîner plusieurs sessions en mode apprentissage, une session prenant comme directives de paramétrages celles déduites de la session précédente. Chaque session réalise une catégorie d'expériences. Le tableau ci-dessous résume les résultats obtenus lors de deux expérimentations avec la base de connaissance de cryptaddition. La première expérimentation a été réalise avec la cryptaddition SEND+MORE=MONEY (colonne 3) et la seconde avec DONALD+GERALD=ROBERT (colonne 4). Le pourcentage indique la gain relativement au paramétrage par défaut.

Ces trois expériences montrent une amélioration sensible de l'efficacité de l'utilisation des connaissances.

session	Catégorie d’expériences	SENDMORE	DONALD
1:	Fusion de prémisses :	75.2 %	78,5%
2:	Ordre des prémisses:	59.0 %	78.4%
3:	Mémorisation:	45,5 %	77.5%
4:	Utilisation des seuils:	43,3 %	78.6%
5:	Ajout de paramètres aux relations:	46,1 %	50.8%
6:	Fusion de prémisses :	45,3 %	38.7%
7:	Ordre des prémisses:	41,2%	51.4%
8:	Mémorisation:	40,6 %	35.4%
9:	Utilisation des seuils:	40,5 %	40,1%
10:	Ajout de paramètres aux relations	40,7 %	40.6%

L’amélioration obtenu pour un jeu de données est aussi intéressante pour l’autre jeu de données ainsi que le montre le tableau suivant :

	amélioration pour SENDMORE	amélioration pour DONALD
Paramétrage appris avec SENDMORE	50,3%	57 ,3%
Paramétrage appris avec DONALD	52.7%	45.7%

13. Conclusion

L'utilisation efficace de connaissances déclaratives n'est pas seulement un problème d'algorithmique : il n'existe vraisemblablement pas d'algorithme qui donnerait satisfaction quelque soit la base de connaissances. L'utilisation efficace de connaissances déclaratives est un problème d'intelligence artificielle qui nécessite pour être résolu l'utilisation d'informations dynamiques ne pouvant être obtenue qu'au moment de l'utilisation des connaissances. Le système Sepiar2 est conçu pour pouvoir produire et utiliser de telles informations et pour pouvoir adapter en conséquence ses méthodes d'utilisation des connaissances. Il possède diverses méthodes pouvant être adaptées à chaque connaissance (relations et règles) de la base. Une méta-expertise permet en analysant le fonctionnement de décider des meilleures méthodes à utiliser et de les activer dynamiquement. Cette direction de recherche est une direction prometteuse comme le confirment les résultats déjà obtenus par ce système.

14. Références

[FAL 89] FALLER B. - *Des méthodes adaptatives pour l'efficacité des systèmes à règles de productions. Le système METRO* - Rapport interne L.R.I. Orsay, 1989

[FOR 82] FORGY C.L. - *RETE : a fast algorithm for the many pattern/many object pattern match problem* - Artificial intelligence Vol.19, 1982

[KOR 93] KORNMAN S., PARCHEMAL Y. *Opportunist Methods for Knowledge Processing* – EWAIC 93 p213-217, 1993

[LAU 86] LAURIERE J.L. - *SNARK: un langage déclaratif* - TSI, vol. 5 , n°3, 141-172, 1986

[MIR 87] MIRANKER D.P. - *A Better Match Algorithm for AI Production Systems* - AAAI 87, 42-47, 1987

[PAR 90] PARCHEMAL Y , MALAVAL G.- *SNOPS : un moteur d'inférence à propagation paramétrable* – Convention IA 90, 181-192 , Paris 1990

[PAR 92] PARCHEMAL Y - *Les moteurs à propagation paramétrable* – Revue de l'intelligence artificielle- Volume 6, pp313-343 , 1992

[PIT 90] PITRAT J. - *Métaconnaissances : futur de l'intelligence artificielle* - Editions Hermès, 1990

Gestion implicite de la concurrence dans un système à base de tableau noir

Tristan PANNEREC
Laboratoire d’Informatique de Paris VI

Résumé : Cet article présente un mécanisme d’adaptation et de combinaison de méthodes concurrentes de résolution dans un système à tableau noir mono-contextuel. Le but est d'isoler les différents types de difficultés qui apparaissent dans les problèmes à résoudre pour les traiter de façon indépendante. Le principe est de décomposer l'expertise de résolution en un ensemble de méthodes simples qui sont combinées par le système pour obtenir un raisonnement complexe. Cette organisation des connaissances conduit souvent à l'apparition de concurrence : plusieurs méthodes peuvent être utilisables au même moment pour résoudre une partie du problème, chacune en traitant un aspect particulier. Le système doit alors arbitrer les éventuels conflits pour obtenir une solution globale. Ce mécanisme à été implémenté dans le système MARECHAL[1] et validé sur des problèmes de construction de solutions pour un jeu de stratégie.

Mots clés : Stratégies concurrentes, combinaison, connaissances de contrôle, tableau noir, jeux de stratégie.

1. Introduction

Pour résoudre des problèmes, un système d’Intelligence Artificielle a en général besoin de nombreuses connaissances. Lorsque ces problèmes sont complexes, le nombre de connaissances devient tel que l’utilisation d’une seule source globale de connaissances est impossible. Les systèmes ont alors intérêt à adopter une architecture distribuée dans laquelle plusieurs sources de connaissances (SC) spécialisées coopèrent pour résoudre le problème [Davis 80]. Ces SC peuvent être construites de façon indépendante (principe de modularité) et être de nature différente (procédures, moteur d’inférence en chaînage avant ou arrière etc.). Ce besoin a donné lieu dans les années soixante-dix à l’introduction du modèle de tableau noir [Nii 86] [Corkill 91], qui connut ensuite de nombreux développements [Engelmore et al. 88] [Jagannathan et al. 89]. Dans ce modèle, les SC communiquent via une zone de données globale et partagée appelée tableau noir permettant d’enregistrer les éléments de solution appelés hypothèses.

L’un des aspects intéressants des modèles à tableau noir est de pouvoir utiliser des stratégies de résolution concurrentes qui permettent de représenter différentes façons de raisonner sur une situation. Chaque stratégie conduit à des hypothèses concurrentes dont il faut gérer les conflits. Cela permet à l'expert de fournir un ensemble de méthodes spécialisées au lieu d'une seule méthode universelle qui est en général impossible à exprimer. Les méthodes étant spécialisées, elles sont plus simples à définir. Mais, pour pouvoir fournir ces méthodes de façon indépendante, le système doit prend en charge la gestion des interactions entre elles. Or, dans la plupart des générateurs de systèmes à tableau noir, aucun mécanisme n’est fourni pour la gestion implicite de la concurrence entre les hypothèses. La gestion de cette concurrence est à la charge du cogniticien qui développe le système.

Dans le système MARECHAL, nous avons implémenté un tel mécanisme dans une optique mono-contextuelle. Le système raisonne en effet sur un seul ensemble d’hypothèses et utilise des techniques de « backtrack » et de résolution de conflit. L’objectif est d’avoir un système dont l’utilisation mémoire est linéairement proportionnelle à la taille du problème.

Après avoir décrit dans la partie suivante l'application test et l'architecture de base du système MARECHAL, nous présenterons le langage de description des connaissances de contrôle utilisé dans le système. La quatrième partie sera alors consacrée à l’utilisation et la gestion des stratégies concurrentes. La cinquième partie décrira les résultats obtenus.

---

[1] Monitoring-based Autonomous Reasoning Engine for Complex Hypothesis hAndLing

2. Présentation du système MARECHAL

Le mécanisme proposé a été implémenté dans le système MARECHAL, qui est un système expert de résolution de problèmes à solution complexe, c'est-à-dire pour lesquels le système n'a pas à effectuer un choix, mais un ensemble de choix coordonnés. Les jeux classiques comme les échecs ou le go sont des exemples de problèmes à solution simple. Les jeux de stratégie, dans lesquels les joueurs doivent faire dans le même temps des choix de déplacement pour chacune de leurs pièces, constituent des problèmes à solution complexe. Le système MARECHAL est appliqué à un jeu de cette famille dont la description est l'objet de la première sous-partie suivante. Ces problèmes ne peuvent être résolus comme les problèmes à solution simple. La résolution doit en effet être pensée en terme de construction et non de choix, ce qui implique une architecture de base spécifique qui sera présentée dans la seconde sous-partie.

2.1. Problème réel étudié

Le système MARECHAL est actuellement utilisé en tant qu'adversaire artificiel dans un jeu de stratégie. Celui-ci consiste à programmer le déplacement d'un ensemble de pièces sur un terrain pour conquérir des objectifs et détruire les pièces adverses. Chaque joueur possède au début de la partie un ensemble de pièces ayant chacune des caractéristiques invariables (bonus contre chaque type de pièce, vitesse de déplacement en fonction du terrain etc.) et des caractéristiques variables au cours de la partie ("force", position etc.). Des règles complexes permettent de régir les combats lorsque deux pièces ennemies se rencontrent. La difficulté du jeu provient du fait qu'il se joue de façon simultanée, contrairement aux jeux de stratégie classiques, comme les échecs ou le Go. Le jeu est en effet figé pendant que les deux joueurs programment leurs déplacements et ceux-ci sont ensuite exécutés automatiquement par le gestionnaire du jeu, ainsi que les combats qui peuvent apparaître. Aucun facteur aléatoire n'est utilisé pendant cette simulation, ce qui fait qu'en dehors des intentions adverses, l'évolution est totalement prévisible.

Ce genre de jeu demande une expertise à la fois volumineuse et complexe. Le système doit en effet gérer des buts contradictoires comme éviter de perdre des pièces, essayer de détruire les pièces de l'adversaire et occuper les objectifs. Au niveau stratégique, il doit savoir étudier le rapport des forces en présence et anticiper les possessions de chaque joueur pour définir son comportement. Il doit ensuite étudier le terrain pour choisir ses axes d'offensive, ses lignes de défense etc. Au niveau tactique, il doit maîtriser toutes les caractéristiques de ses pièces (bonus selon les terrains, les pièces adverses etc.), coordonner leurs déplacements et fournir des solutions qui soient correctes quelle que soit la solution adverse. L'espace des problèmes est énorme car chaque situation est définie par le terrain (un damier d'environ 400 à 4000 cases avec 5 types différents de terrain), un ensemble de 10 à 100 pièces pour chaque joueur, les caractéristiques de chaque type de pièce et les conditions de victoire. Dans un tel contexte, trouver un algorithme qui fonctionne quelle que soit la situation est rapidement apparu comme une tâche impossible, ce qui a motivé l’utilisation de stratégies de résolution spécialisées.

Afin d’éviter toute ambiguïté dans la suite de cet article, il nous semble important de préciser la différence entre une stratégie de jeu et une stratégie de résolution. Les stratégies de jeu réfèrent aux solutions possibles du problème, alors que les stratégies de résolution sont des méthodes ou « algorithmes » qui permettent au système de résoudre le problème, c’est-à-dire de trouver une bonne stratégie de jeu.

2.2. Architecture générale du système

L'architecture générale du système MARECHAL est organisée en trois couches (cf. Figure 24). La couche la plus basse est une couche de perception qui permet de dialoguer avec le monde extérieur (dans notre cas, le gestionnaire du jeu). Elle contient tout d'abord un ensemble de services utilisables par les couches supérieures pour interroger le monde extérieur. Pour un jeu de stratégie, il s'agit de pouvoir accéder au terrain, aux caractéristiques des pièces etc. Cette couche contient également des fonctions de pré-traitement de ces informations (calcul d'indices de menace grossiers pour un point, étude de la répartition géographique des éléments du jeu etc.) ainsi qu'un ensemble de procédures permettant d'agir sur le monde (écriture des ordres de mouvement pour chaque pièce etc.).

La seconde couche est la couche inférence qui permet d'effectuer les déductions et les décisions. Elle est principalement centrée sur un moteur d'inférence qui utilise des conseils écrits dans un langage proche de la logique des prédicats. Pour l'application au jeu de stratégie, le système dispose actuellement d'environ 300 conseils. Comme dans la plupart des problèmes, la chaîne d'inférences peut être très longue et conduire à une explosion combinatoire si ces conseils ne sont pas organisés, ce qui a motivé une vision distribuée de la base de conseils. Le raisonnement a donc été segmenté au maximum par un découpage en plusieurs étapes correspondant chacune à la résolution d'une partie du problème. Les conseils ont alors été regroupés en 50 bases qui, associées au moteur d'inférence, forment 50 SC du domaine différentes. Lors de l'utilisation d'une base, plusieurs paramètres doivent en général être fixés afin de focaliser l'exécution sur un ensemble de données. Par exemple, la base qui permet d'étudier les opportunités d'attaque d'une pièce possède comme paramètre le numéro de la pièce étudiée. Cette couche contient également quelques SC du domaine écrites directement en C++. Il s'agit de procédures qui implémentent des raisonnements spécialisés (recherche d'un chemin par descente de gradient, anticipations à court terme etc.) et qui ne peuvent être définies par une base de conseils.

La troisième couche est la couche de résolution/construction qui permet de contrôler la résolution du problème, c'est-à-dire de choisir en permanence les SC du domaine à exécuter. Elle utilise pour cela un ensemble de SC de contrôle écrites dans un langage, baptisé CKSL, qui sera présenté dans la partie suivante. Cette couche est constituée d'un interpréteur qui dispose de deux mécanismes principaux : le premier concerne la gestion de la concurrence entre SC et est détaillé dans la quatrième partie de ce papier, le second porte sur la détection, l'analyse et la correction d'erreur ; il permet au système de revenir sur des choix qui s'avèrent mauvais a posteriori. La description de ce second mécanisme ne sera pas abordée dans ce papier.

Le système dispose enfin d'une mémoire principale sous la forme d'une base de faits qui constitue la zone de données globale (tableau noir). Cette base permet d'enregistrer toutes les connaissances déduites et utilisées par les SC, comme par exemple : « la pièce p7 a pour ordre d'attaquer le point (7,5) » ou « au prochain tour, l'objectif o3 sera contrôlé par l'adversaire ». Elle permet également de répertorier les objets du monde et leurs propriétés comme par exemple : "p7 est une pièce" ou "p7 appartient au groupe g1". Ces faits sont très utiles pour l'instanciation des variables dans les règles : cela permet d'écrire, par exemple, « soit o un objectif » ou « soit p une pièce du groupe g1 ». Chaque fait possède une valeur de vérité comprise entre 0 (totalement faux) et 100 (totalement vrai) qui peut avoir des sémantiques différentes. L'interfaçage de la base de faits avec les autres composants du système se fait via un module de gestion mémoire.

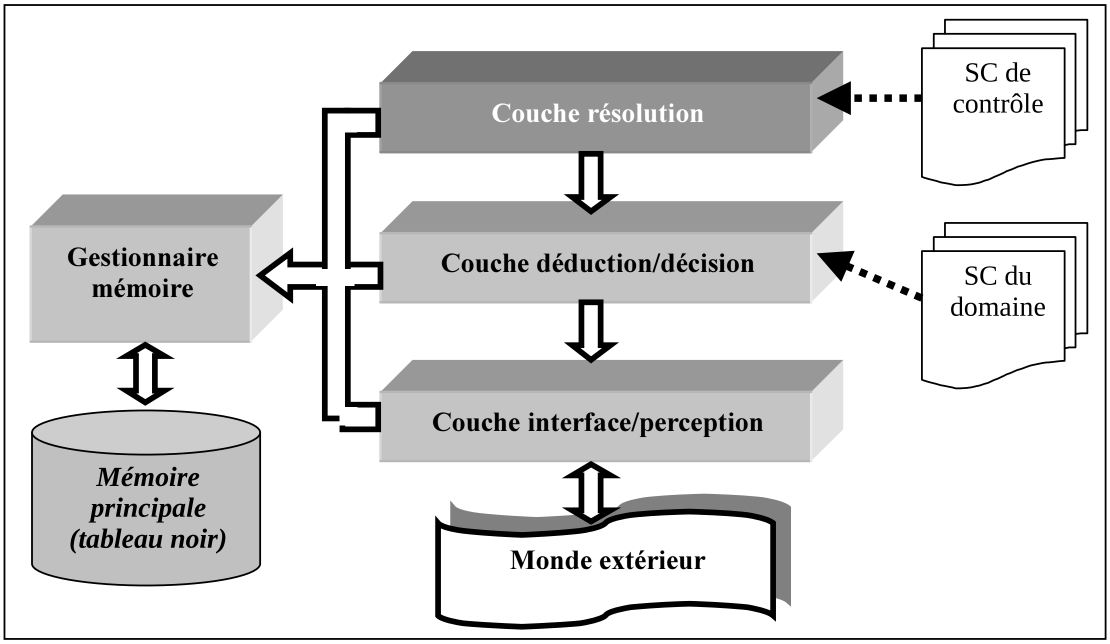


Figure 24: *Architecture générale du système*

3. Définition des connaissances de contrôle

Dans cette partie, nous faisons tout d'abord un rapide historique des diverses formes de contrôle utilisées dans les systèmes à tableau noir. Nous présentons ensuite le langage de description des connaissances de contrôle utilisé dans le système MARECHAL.

3.1. Le contrôle dans le modèle de tableau noir

Dès son apparition, le modèle du tableau noir posa le problème du contrôle des diverses sources de connaissances. Un système à base de tableau noir doit en effet déterminer en permanence quelle SC activer et sur quelles données. Le problème se complique lorsque les SC peuvent être à la fois coopératives ou concurrentes. Dans les premiers systèmes, comme HearSay-II [Erman et al. 80] [Lesser et al. 77], le contrôle était implémenté de façon procédurale. Ce type de contrôle était difficile à maintenir et insuffisant pour décrire des stratégies complexes. Il devint donc nécessaire de fournir aux systèmes à base de tableau noir des connaissances de contrôle explicites formant des SC de contrôle par opposition aux SC du domaine. Plusieurs possibilités ont été explorées. Le principe initial était l'opportunisme : chaque SC s'activait « d'elle-même » (par l'intermédiaire d'un programme de contrôle) lorsqu'elle pensait pouvoir entraîner une progression vers la solution. Si cette vision était très souple, elle était parfois assez inefficace et difficile à mettre en place lorsqu'on pouvait déterminer a priori des chaînes d'activations figées pour le problème. Le principe de contrôle hiérarchique (que nous appellerons d'une manière générale « contrôle a priori ») fut donc introduit dans des systèmes comme HASP/SIAP [Nii et al. 82] et CRYSALIS [Terry 83], où les SC sont organisées en une hiérarchie à plusieurs niveaux. Lorsqu'une SC de contrôle est activée, elle choisit une séquence d'activations de SC du niveau inférieur. Ce type de contrôle pose le problème inverse du précédent : il est inadapté lorsque le flux de contrôle ne peut être déterminé a priori. Pour pallier cela, des systèmes hybrides, comme le système ATOME [Lâasri et al. 88] sont apparus. Ils permettent d'allier à la fois la souplesse d'un contrôle opportuniste avec l'efficacité d'un contrôle hiérarchique. C'est dans cette logique que se positionne le système MARECHAL.

3.2. Description des connaissances de contrôle dans le système MARECHAL

Pour pouvoir décrire les connaissances de contrôle nécessaires au système MARECHAL, nous avons défini un langage spécialisé baptisé CKSL[1]. Toutes les SC de contrôle sont donc écrites dans ce langage. Contrairement à la plupart des systèmes de tableau noir existants, le contrôle n'est pas écrit sous forme de règles mais sous forme d'instructions impératives. Cela permet à l'utilisateur de définir les plans de résolution dans leur forme originale, sans avoir à les convertir en règles. Le langage permet l'utilisation de variables, l'appel à des SC de contrôle ou du domaine et la possibilité d'interventions directes sur la base de faits, bien que ces interventions soient rarement utiles car le système les prend normalement à sa charge.

Il est basé sur quatre opérateurs principaux ayant chacun un rôle particulier. Le premier est l'opérateur de séquence qui permet de gérer la coopération entre différentes SC dans la réalisation d'une tâche commune, par décomposition de celle-ci en une séquence d'étapes. Le type de contrôle est ici « a priori ».

Le second est l'opérateur de combinaison qui permet de gérer la concurrence entre plusieurs SC. Lorsque la situation peut être traitée selon plusieurs méthodes de raisonnement, ces méthodes sont concurrentes dans le sens où elles cherchent chacune à imposer leur solution. L'interpréteur doit alors résoudre les conflits qui apparaissent. Le contrôle est ici opportuniste car l'ordre dans lequel sont activées les stratégies n'est pas fixé a priori, mais est déterminé par le système en fonction de la situation, au moment de l'exécution. Plus une stratégie est adaptée à la situation, plus elle sera déclenchée tôt. L'utilisation de stratégies concurrentes et le mécanisme d'adaptation et de combinaison associé dans l'interpréteur sont décrits plus en détails dans la quatrième partie de ce papier.

Les deux opérateurs restants correspondent à des structures itératives. La boucle « Compteur » permet d'implémenter un compteur qui incrémente à chaque passage une variable en partant d'une valeur initiale jusqu'à une valeur finale. Ce type de boucle permet par exemple de parcourir un ensemble d'objets (les objectifs, les pièces, les cases etc.). La fonction est coopérative et le contrôle est « a priori » puisque l'ordre et le nombre de passages sont définis au moment de l'écriture de la boucle.

La boucle « ChoixMultiple » permet de faire une succession de choix interdépendants sur des données pour effectuer une focalisation locale. Cette boucle sert en général à prendre des décisions concernant la solution du problème. Pour cela, une SC du domaine calcule au préalable l'ensemble des possibilités avec, pour chacune, une valeur d'intérêt heuristique. La boucle détermine alors les possibilités qui seront effectivement retenues. La concurrence est ici double. Un choix peut tout d'abord empêcher plusieurs autres choix ultérieurs. Ainsi, si on décide d'envoyer la pièce p7 au point (6,3), cela pourra empêcher les autres choix portant sur la même pièce ou le même point. Un choix peut également avoir pour conséquence l'appropriation de ressources partagées. Dans l'exemple précédent, la pièce p7 va s'approprier un chemin sur la carte pour atteindre le point (6,3). Dans certains cas, cela peut empêcher d'autres mouvements, par faute de place.

---

[1] Control Knowledges Specification Language

Le tableau de la Figure 25 résume les caractéristiques des quatre opérateurs utilisés. On remarquera que chaque opérateur correspond à un couple (fonction, objet) donné et que le type de contrôle dépend uniquement de la fonction de l'opérateur.

Opérateur	Mot clé	Fonction	Objet	Type de contrôle
séquence	-	coopération	SC	a priori
combinaison	« Combiner »	concurrence	SC	opportuniste
itération 1	« QQSoitEntier »	coopération	données	a priori
itération 2	« ChoixMultiple »	concurrence	données	opportuniste

Figure 25 : *Liste des opérateurs de CKSL*

Bien que la plupart du temps, l'utilisation de plan de résolution remplace avantageusement celle de règles, il existe des cas où les règles peuvent être intéressantes, notamment pour implémenter des opérations réflexes. L'appel systématique d'une vérification de contraintes après une étape d'anticipation en est un exemple. Pour cela, le langage permet la définition de bases de déclenchement, qui associent des actions à des étapes figurant dans les plans.

4. Les stratégies de résolution spécialisées

Lorsque plusieurs méthodes de résolution différentes existent pour un même problème, il est en général intéressant de donner au système toutes ces méthodes. Ainsi, pour gérer les différentes stratégies de résolution dont il dispose, le système MARECHAL utilise un mécanisme basé sur deux étapes principales : l'adaptation et la combinaison. Cette partie présente une discussion sur l'utilisation de stratégies spécialisées puis une vue générale de leur gestion dans le système MARECHAL et enfin le détail des deux étapes principales du mécanisme.

4.1. Utilisation de stratégies de résolution spécialisées

La difficulté de l'acquisition d'une méthode de résolution vient de la multiplicité des formes que peut prendre cette méthode selon le problème posé. Il semble que lorsque le problème fait intervenir des difficultés d'ordres différents, l'expert humain dispose de méthodes de résolution spécifiques adaptées à chacune d'elles. Par exemple, lorsque des pièces ennemies sont proches, l'expert humain applique un raisonnement tactique où le combat est central. Au contraire, si les pièces ennemies sont éloignées, les combats sont à court terme très improbables et l'expert se concentre alors sur le déploiement de ces pièces grâce à une réflexion plus stratégique.

Ces constatations nous ont conduit à construire un système capable, lui aussi, d'adapter et de combiner des stratégies de résolution simples pour obtenir un raisonnement complexe. Le principe est d'avoir une méthode de résolution pour chaque type de difficulté. La première étape fut d'identifier les différents types de difficultés présents dans le problème. Dans notre application, le problème contient à la fois une difficulté d'ordre stratégique et une difficulté d'ordre tactique. Pour chacune de ces difficultés, nous avons alors considéré les cas extrêmes de l'espace du problème dans lesquels cette difficulté est unique et déduit les contraintes qui permettaient de les définir. Par exemple, pour la difficulté d'ordre stratégique, les cas extrêmes associés sont les cas de début de partie et la contrainte porte sur l'absence de pièces ennemies proches. Cette contrainte est très forte et permet de s'affranchir d'un grand nombre de cas. La mise au point d'une méthode de résolution sur cet ensemble de cas extrêmes ne pose alors plus de problème. L'expert qui transmet sa stratégie n'est plus obligé de penser à tous les cas particuliers possibles. Ces contraintes lui permettent de se délimiter un champ d'étude et de se concentrer sur la difficulté visée. Dans le cas général, lorsque les contraintes ne sont pas vérifiées, l'adaptation de la méthode est à la charge du système. L'association des contraintes et de la méthode de résolution est appelée un schéma de résolution.

Le schéma principal est celui qui est exécuté initialement. Pour résoudre le problème, il fait appel à un ensemble de deux sous-schémas comme le montre la Figure 26. Chacun de ces sous-schémas permet de résoudre le problème dans des conditions différentes. Le schéma stratégique s'occupe du déploiement stratégique lorsque les pièces sont loin de celles de l'adversaire. Le schéma tactique s'occupe des combats lorsque les pièces sont en contact. Il utilise lui-même deux sous-schémas (STO et STD) selon que la situation est offensive ou défensive. On remarquera que les schémas s'opposent dans la définition de leur cadre d'application. Tous ces schémas sont implémentés dans le système actuel, mais de nombreux autres pourraient être utilisés, selon la nature du terrain, la composition des armées etc.

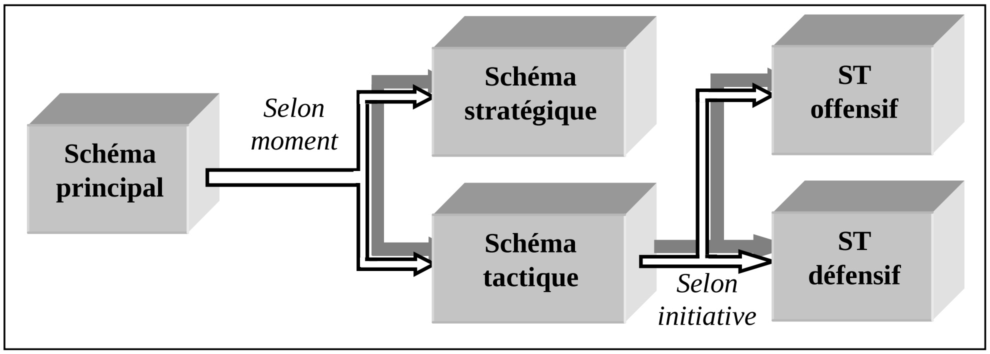


Figure 26: *Hiérarchie des schémas de résolution implémentés*

L'utilisation de sous-schémas dans les plans se fait grâce à l'opérateur « Combiner » présenté dans la troisième partie. Malgré les apparences, cette instruction est différente d'un sélecteur (comme le « switch » du C ou le « case of » du Pascal). Il ne s'agit pas de sélectionner un schéma mais de les exécuter tous en les adaptant et les combinant. La Figure 27 présente de façon simplifiée le contenu de deux méthodes de résolution.

Schéma principal	**Schéma Tactique**
Analyser la mission	Analyser l'ennemi
Analyser les moyens	Analyser le terrain
Combiner	Combiner
Cas ennemi lointain	Cas offensif
⇨ Schéma Strat.	⇨ Schéma STO
Cas ennemi proche	Cas défensif
⇨ Schéma Tactique	⇨ Schéma STD
FinCombiner	FinCombiner
Envoyer la solution	

Figure 27: *Exemple de plans de résolution*

Le but étant donc d'associer à des cas des méthodes de résolution, cette approche peut faire penser au raisonnement à partir de cas (CBR). Il existe en fait deux différences importantes. La première porte sur le choix des cas. Dans le raisonnement à partir de cas, on prend un grand nombre de cas au hasard pour couvrir l'espace des problèmes. Chaque cas doit être vu comme un point de cet espace. Dans l'approche proposée ici, les cas sont peu nombreux et choisis avec beaucoup de soin car ils représentent plutôt des composantes de l'espace. Plus précisément, deux ou plusieurs cas définissent un vecteur de base de cet espace. Chaque cas correspond à une valeur extrême de la composante. Par exemple, la proximité de l’ennemi peut être considérée comme une composante dans l'espace des situations et les deux cas extrêmes correspondent aux valeurs "proche" et "loin". La seconde différence est que, dans le CBR, on associe à chaque cas sa solution et non la méthode de résolution. On adapte alors la solution, alors qu'ici on adapte la méthode de résolution. Celle-ci est ensuite exécutée pour obtenir une solution. Cette démarche est plus difficile à mettre en oeuvre, car il est facile d'obtenir de nombreux exemples de solution alors qu'il est pénible de fournir même un petit nombre de méthodes. En outre, l'espace des problèmes doit être analysé pour déterminer les cas extrêmes intéressants. Néanmoins, cette démarche permet de travailler sur des problèmes dans lesquels il n'existe pas de liens directs entre les caractéristiques des situations et leur solution associée, c'est-à-dire lorsque la fonction qui à un cas associe une solution, est fortement irrégulière. Dans le cadre d'un jeu de stratégie, deux situations très proches peuvent en effet conduire à des solutions très différentes, et le raisonnement à partir de cas n'est pas utilisable.

4.2. Vue générale du mécanisme de gestion des stratégies spécialisées

Lorsque plusieurs stratégies spécialisées sont disponibles, le raisonnement de l'expert ne consiste pas simplement à sélectionner l’une d’elles pour l'appliquer. Dans notre exemple, il est extrêmement rare d’avoir uniquement des pièces ennemies proches ou des pièces ennemies éloignées. On est donc dans aucun des deux cas possibles (début ou milieu de partie). L’expert est alors capable d’adapter ses méthodes et de les localement : certaines pièces obtiennent leurs mouvements d'après une réflexion stratégique et d'autres d'après une réflexion tactique, selon leur environnement. Nous avons donc souhaité donner au système les mêmes capacités. La gestion des différentes stratégies est effectuée par le système et non par l'expert qui fournt les connaissances. Ce dernier n'a ainsi plus à faire l'effort de généralisation pour obtenir une méthode universelle.

Dans le système ATOME, un tel mécanisme est fourni, mais il repose sur le paradigme multi-monde [Lâasri et al. 88] de l'ATMS [De Kleer 86] et son extension implémentée dans ART [Clayton 85]. Dans ce paradigme, plusieurs mondes coexistent dans le système, chacun représentant un ensemble cohérent d'hypothèses. Ces mondes sont organisés selon une hiérarchie et on associe à chaque morceau de connaissance le monde le plus général auquel il appartient. Cette approche pose deux problèmes. Le premier concerne la quantité d'espace mémoire utilisé. Lorsque le nombre de mondes à envisager est très grand et que chaque monde engendre des données nombreuses et différentes, on obtient des bases de connaissances énormes. Dans l'application que nous avons étudiée, le système envisage souvent plusieurs milliers de mondes et doit engendrer pour chaque ensemble d'hypothèses plusieurs milliers (voir dizaines de milliers) de faits. Le deuxième problème est celui de la détermination des dépendances entre les connaissances. Lorsqu'une SC produit une hypothèse, il faut déterminer le monde le plus général auquel appartient cette nouvelle hypothèse en fonction des hypothèses qui ont conduit à sa création. Or, dans le cas de SC sous forme de boîte noire, cette tâche est impossible automatiquement et requiert donc l'intervention de l'utilisateur. A cause de ces problèmes, et parce que nous pensons que le cerveau humain ne fonctionne pas de cette façon, nous avons cherché à implémenter dans le système MARECHAL les mêmes possibilités de gestion de concurrence à partir d'une vision à monde unique (mono-contextuelle).

La gestion des stratégies de résolution permet de résoudre un cas général pour lequel le système n'est normalement pas prévu puisque ces connaissances ne portent que sur quelques cas extrêmes. Schématiquement, cela revient à se donner une base de méthodes $\{M_i\}$ et d'engendrer l'espace des méthodes $\Sigma(\alpha_i M_i)$. La pondération représente le fait que chaque méthode M_i doit être appliquée de façon focalisée sur la partie du problème qu'elle sait traiter. Par exemple, la méthode qui permet de déployer les pièces (schéma de début de partie) ne doit s'appliquer qu'aux pièces qui ne sont pas trop proches des pièces adverses parmi l'ensemble des pièces. Cette focalisation globale est effectuée grâce à la première étape du mécanisme : l'étape d'adaptation d'une méthode à la situation courante. Elle est différente et complémentaire de la focalisation locale que l'on trouve dans le modèle à tableau noir et qui existe également dans le système MARECHAL. La focalisation globale est en effet plus générale car une donnée peut plus ou moins appartenir à la focalisation mais elle est en contrepartie plus lourde à gérer. La sommation dans l'expression «$\Sigma(\alpha_i M_i)$» représente la seconde étape du mécanisme, à savoir la combinaison dynamique des applications focalisées des méthodes M_i. Ce processus global qui est représenté sur la Figure 28 comprend donc deux étapes principales.

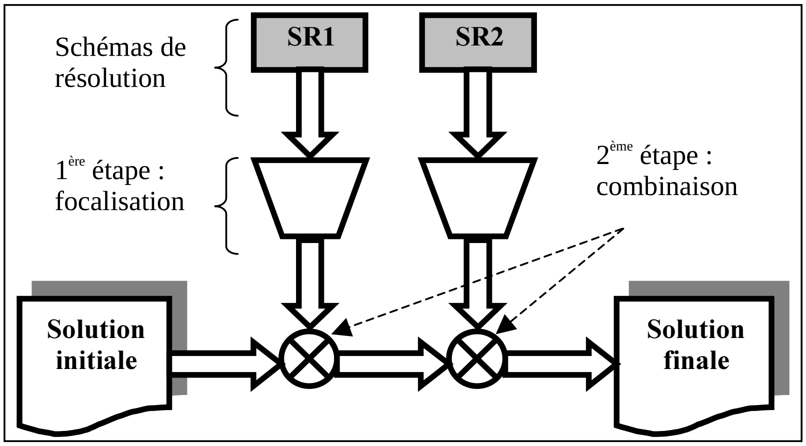


Figure 28: *Mécanisme de gestion des stratégies de résolution*

4.3. Application focalisée d'un schéma de résolution

Pour adapter l'exécution d'un schéma à la situation courante, l'interpréteur se base sur les conditions du schéma qui définissent les caractéristiques spécifiques des cas extrêmes associés. Ces conditions sont fournies sous la forme de contraintes sur des objets du monde. Pour le schéma stratégique, il s'agit de contraintes de distances portant sur l'ensemble des pièces amies pour représenter le fait que ces pièces doivent être hors de portée des pièces adverses. Le système calcule tout d'abord une répartition d'appartenance sur l'ensemble d'objets visé. Plus la contrainte est vérifiée (cette évaluation est en effet souvent floue), plus l'appartenance au schéma sera élevée. Cette appartenance est stockée comme la valeur de vérité du fait répertoriant l'objet dans la base de faits. On obtiendra par exemple que p7 est une pièce à 57%. Le système fait alors une moyenne de ces appartenances microscopiques pour déterminer l'adéquation globale de la situation courante avec les conditions d'exécution du schéma. Cette adéquation globale est utilisée pour déterminer l'ordre dans lequel les schémas concurrents sont exécutés. Les schémas les plus en rapport sont en effet exécutés en priorité et ceux qui sont totalement inadaptés sont ignorés. Par exemple, on peut avoir pour une situation les valeurs suivantes : stratégie = 20%, tactique = 80%. Le système va alors exécuter le schéma de milieu de partie en premier puis modifier la situation obtenue par l'exécution du schéma de début de partie.

Avant d’exécuter une stratégie de résolution, le système doit propager les degrés d'appartenance microscopique obtenus sur l'ensemble des objets directement liés aux contraintes. En effet, de nombreux objets ont des liens entre-eux et doivent donc "hériter" des degrés d'appartenance. Dans le jeu de stratégie étudié, les pièces sont organisées en hiérarchie avec la présence de pièces de commandement. Un groupe de pièces est donc un objet abstrait du monde et son appartenance au schéma est la moyenne des degrés d'appartenance de chaque pièce qui le compose. La propagation doit également se faire sur les faits qui représentent des propriétés d'un ou plusieurs objets. Par exemple, le fait "p5 est une pièce du groupe g1" est une propriété pour la pièce p5. Son degré d'appartenance au schéma sera défini comme celui de la pièce p5. Là encore, l'appartenance est enregistrée comme la valeur de vérité du fait représentant la propriété. Le système propage automatiquement toutes ces données grâce à une base de définitions d'objet et de propriété.

Une fois ces degrés d'appartenance calculés, le système les utilise pour modifier l’évaluation des conseils. Il lui suffit de prendre en compte dans la valeur finale d'un conseil les valeurs d'instanciation. Considérons par exemple le conseil fictif qui est présenté dans la Figure 29 tel qu'il serait fourni au système. Ce conseil signifie que pour toute pièce p, plus la force de p est élevée par rapport aux autres pièces, plus il faut envoyer la pièce p sur le point (7,5). Supposons que, comparée aux forces des autres pièces, la force de la pièce p7 soit élevée à 70%. Sans focalisation, le système obtient alors qu'il faut envoyer p7 en (7,5) avec un intérêt de 70% car p7 est considérée comme étant une pièce à 100%. Supposons maintenant que la pièce p7 appartienne à 20% au schéma de résolution courant, la même évaluation donnera cette fois un intérêt très inférieur à 70% car le système va effectuer une agrégation conjonctive entre les valeurs 70% et 20%. Ce résultat traduit le fait que, la pièce n'appartenant que très faiblement au schéma, on accorde beaucoup moins d'importance aux décisions qui la concernent.

```
(       Quelquesoit[pièce($p)]
        Elevé[force($p)]
) bon <100> Faire Ajouter[ordre($p aller_en(7,5))]
```

Figure 29: *Exemple de conseil*

Ce mécanisme permet véritablement d'adapter l'exécution du schéma à la situation courante. Dans l'exemple de la Figure 30, l'occupation de l'objectif O est capitale pour les noirs. Si l'on applique le schéma stratégique sans focalisation, le système décide d'y envoyer la pièce A car elle est la plus proche (mouvements noirs). Avec la focalisation, le système préférera envoyer la pièce B (mouvements blancs) car l'appartenance de la pièce A au schéma est très faible puisqu'elle est proche de pièces ennemies. Cette solution est bien plus sûre et la pièce A pourra être utilisée pour les combats tactiques qui risquent d'avoir lieu sur la ligne de front.

Figure 30: *Exemple des conséquences de la focalisation*

4.4. Combinaison de schémas

Grâce à l’étape de focalisation, le système peut exécuter un schéma de résolution en l'adaptant à la situation. Mais lorsque plusieurs schémas concurrents doivent être exécutés, le système doit également combiner les résultats obtenus. L'objectif est que chaque schéma apporte sa contribution dans la solution finale, selon ses capacités. Pour réaliser cette combinaison, il y a deux possibilités : soit on applique tous les schémas et on combine ensuite les solutions obtenues, soit on applique les schémas de manière séquentielle et on combine en temps réel les résultats. Dans le dernier cas, chaque nouvelle solution est combinée pendant sa construction avec la solution courante et le résultat devient la nouvelle solution courante. La première méthode a l'avantage d'être indépendante de l'ordre d'exécution des schémas mais pose de gros problèmes pour le calcul de la solution finale, notamment à cause du fait que les conséquences de deux choix ne sont souvent pas la somme des conséquences de chaque choix. Supposons par exemple que dans la solution 1, la pièce x1 cherche à se déplacer vers le point y1 et que dans la solution 2, la pièce x2 (x2 différent de x1) cherche à se déplacer vers le point y2. Dans chaque solution, les conséquences sont que chaque pièce atteindra son objectif. Si y1 est différent de y2, il n'y a pas de problème pour la solution globale. Mais si y1 et y2 sont identiques, les conséquences sont modifiées dans la situation globale car l'une des deux pièces ne pourra pas atteindre son objectif. Nous avons donc choisi la seconde possibilité qui consiste à combiner au fur et à mesure des exécutions des schémas. Cette solution a, en outre, l'avantage de faciliter les remises en question et de construire progressivement la solution. Pour cela, le système part d'une solution vide puis il exécute de manière séquentielle les schémas concurrents. Chaque exécution modifie la solution en lui apportant de nouveaux éléments.

Comme les champs d'action des schémas se recoupent la plupart du temps, le système doit gérer les conflits qui apparaissent entre les schémas. Par exemple, il est rare qu'une pièce appartienne à un seul schéma. Deux schémas peuvent donc donner des mouvements différents à la même pièce, ce qui est impossible. Il faut

Figure 31: Exemple de combinaison

5. Résultats

L'utilisation du mécanisme de gestion des stratégies de résolution améliore nettement le niveau de jeu de l'ordinateur par rapport à une simple sélection des schémas concurrents. Des tests effectués sur plusieurs dizaines de parties ont révélé une amélioration de l'ordre de 50%, mais ce nombre n'est qu'indicatif car le niveau actuel du système est surtout limité par son manque de connaissances, ce qui peut biaiser les résultats. La Figure 32 présente quelques exemples de confrontations entre le système jouant avec gestion des stratégies spécialisées et le système jouant par simple sélection. Pour chaque partie, le tableau indique les scores obtenus. Ces parties n'étant pas forcément équilibrées, elles sont jouées deux fois avec échange des camps.

Partie	**#1**		**#2**		**#3**		**#4**	
Avec gestion	17	20	14	14	15	23	22	11
Sans gestion	11	12	10	6	14	12	7	15

Figure 32: *Quelques exemples de confrontations "avec gestion" contre "sans gestion"*

6. Conclusion

Grâce à son mécanisme d'adaptation et de combinaison, le système MARECHAL est capable d'utiliser des exemples de résolution sur des cas simples pour résoudre des cas généraux complexes. La complexité émergente de son raisonnement est ainsi obtenue par la confrontation avec la situation, mais des améliorations dans la résolution des conflits sont encore nécessaires. De part sa vision mono-contextuelle, le système ne gère en permanence qu'un seul ensemble d'hypothèses, ce qui lui permet de limiter fortement ses besoins en espace mémoire et donc de traiter des problèmes volumineux.

Le but du mécanisme proposé est avant tout de simplifier l'acquisition des méthodes de résolution. Au lieu de chercher une méthode très complexe qui marche dans tous les cas anormaux, l'expert se contente de définir des méthodes simples dont il délimite ensuite les conditions d'application. Malgré cela il faut toujours des méthodes pour chaque extrémité d'une composante de l'espace des problèmes. Or, bien souvent, l'une des extrémités correspond à une situation normale et les autres à des cas particuliers et il n'est pas possible de fournir des méthodes pour tous les cas particuliers. Les perspectives d'amélioration à long terme du système seraient donc de le rendre capable de se suffire des cas normaux. Le système devrait alors analyser la situation pour détecter les anormalités et y adapter ses connaissances au moment du raisonnement.

7. Bibliographie

[Clayton 85] Clayton B. D. : *A First Look at Viewpoints*. Inference Corporation, 1985.

[Corkill 91] Corkill D. D. : *Blackboard Systems* – AI Expert 6 (9), p. 40-47, 1991.

[Davis 80] Davis R. : *Report on the Workshop on Distributed Artificial Intelligence* **-** SIGART Newsletter, n°73, p. 42-52, 1980.

[De Kleer 86] De Kleer J. : *An Assumbtion-based TMS*. Artificial Intelligence 28, p. 127-162, 1986.

[Engelmore et al. 88] Engelmore R. et Morgan T. : *Blackboard Systems* – Addisson-Wesley, Reading, Massachussets, 1988.

[Erman et al. 80] Erman L. D., Hayes-Roth F., Lesser V. R. et Reddy D. R. : *The HearSay-II Speech-Understanding System : Integrating Knowledge to Resolve Uncertainty* **-** ACM computing Surveys 12 (2), p. 213-53, 1980.

[Jagannathan et al. 89] Jagannathan V., Dodhiawala R. et Baum L. S. : *Blackboard Architectures and Applications* – Academic Press, New-York, 1989.

[Lâasri et al. 88] Lâasri H., Maître B., Mondot T., Chapillet F. et Haton J.P. : *ATOME : A Blackboard Architecture with Temporal and Hypothetical Reasoning* – Proceedings of the 8th ECAI, Munich, p. 1-5, 1988.

[Lesser et al. 77] Lesser V. R. et Erman L. D. : *A Retrospective view of the HearSay-II Architecture*. In Proceedings of IJCAI-77, p790-800, 1977.

[Nii et al. 82] Nii H. P., Feigenbaum E. A. Anton J. J. et Rockmore A. J. : *Signal-to-Symbol Transformation : HASP/SIAP Case Study* – AI Magazine 3 (2), p. 23-35, AAAI Press, 1982.

[Nii 86] Nii H. P. : *Blackboard Systems : The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures* – AI Magazine 7 (2), AAAI press, 1986.

[Terry 83] Terry A. : *The CRYSALIS Project : Hierarchical Control of Production Systems* – Tech. Report HPP-83-19, Université de Standford, Califonie, 1983.

Le nouveau MUSCADET et la TPTP Problem Library

Dominique PASTRE
Crip5 - Université René Descartes

Résumé :
Après avoir rappelé les principales caractéristiques du démonstrateur MUSCADET et avoir présenté les améliorations de la version 2 de ce système, cet article présente la TPTP Problem Library (Thousands of Problems for Theorem Provers) et les compétitions de démonstrateurs organisées dans le cadre des conférences CADE (Conferences on Automated DEduction) puis présente et discute le travail récent effectué autour de la TPTP Problem Library : analyse des problèmes, enrichissement de MUSCADET, enrichissement de la base TPTP, compétition 1999.

Mots-clés :
Raisonnement mathématique, Démonstration automatique de théorèmes, Systèmes à base de connaissances, Déduction naturelle

1. Introduction

Le démonstrateur MUSCADET [Pastre 1989 et 1993] est un système à base de connaissances qui utilise des méthodes issues de la déduction naturelle, proches de celles utilisées par le mathématicien. Dans la section 2, on compare ces méthodes avec le Principe de Résolution utilisé par la plupart des démonstrateurs.

Des connaissances et savoir-faire mathématiques, élémentaires ou plus spécialisées, sont donnés au système sous forme de règles et métarègles, écrites, interprétées par un moteur d'inférence. Des actions complexes sont décrites par des paquets de règles.

Dans la première version de MUSCADET[1], un langage d'expression des règles et métarègles a été défini et un moteur d'inférence a été écrit en Pascal. La version 2 de MUSCADET, appelée aussi PUSCADET[2] est une réécriture de l'ensemble du système en Prolog (connaissances et moteur d'inférences, ce dernier étant réduit à quelques prédicats Prolog complétant l'interpréteur Prolog). Cette traduction s'est accompagnée de nombreuses améliorations qui sont décrites en sections 3 puis 5, ainsi que les raisons du choix de ce langage unique pour exprimer toutes les connaissances, qu'elles soient déclaratives pures, procédurales ou intermédiaires.

La section 4 présente la TPTP Problem Library (Thousands of Problems for Theorem Provers) et les compétitions de démonstrateurs organisées dans le cadre des conférences CADE (Conferences on Automated DEduction).

La section 5 présente et discute le travail récent effectué autour de la TPTP Problem Library : analyse des problèmes, enrichissement de MUSCADET, enrichissement de la base TPTP, préparation de la compétition 1999 dont les problèmes et les résultats sont donnés et commentés en section 6.

2. La démonstration automatique de théorèmes (DAT)

Rappelons que le premier programme d'IA est un programme de DAT, c'est le LOGIC THEORIST de Newell, Shaw et Simon, écrit en 1956 pour résoudre des sous-problèmes posés par la réalisation de leur premier programme d'échecs.

Depuis les démonstrateurs utilisent deux grandes familles de méthodes : le Principe de résolution ou les méthodes issues de la Déduction naturelle.

---

[1] Moteur Utilisant un Système de Connaissances Adapté à la DEmonstration de Théorèmes.

[2] Prolog Utilisant un Système de Connaissances Adapté à la DEmonstration de Théorèmes *ou*
Modélisation Utilisant un Système de Connaissances Adapté à la DEmonstration de Théorèmes version 2.

2.1. Principe de résolution et Déduction naturelle

Parmi les programmes anciens, on peut encore citer le programme de Wang (1960) qui commence à avoir de bons résultats et est basé sur le calcul des séquents.

Puis Robinson (1965) révolutionne le domaine avec son "Principe de Résolution" et les méthodes précédentes (calcul des séquents, déduction naturelle) sont abandonnées jusqu'à l'article de Bledsoe (1971) qui montre une meilleure efficacité de la Déduction naturelle, combinée avec le Principe de Résolution. Par la suite, Bledsoe n'utilisera plus que la Déduction naturelle dans son UT theorem prover (1978). Il écrit à cette époque "We have *talked* a lot and proved very *few* hard theorems (by computer) during the last several years. It is time to *do*, to show that our concepts are good. It is time to get a lot more *experience* with our provers. … . One thing that would help push this field ahead, would be for authors to follow the practice of publishing the proof of at least one *hard theorem* in each new methods paper. We do not believe this field will remain vital unless we developp truly powerful provers, and not just theories." (Bledsoe 1977, page 29)

Néanmoins, la plupart des auteurs continuent à travailler avec le Principe de Résolution, en raison de son efficacité théorique supérieure. De nombreuses stratégies ont été écrites pour améliorer la méthode sur le plan pratique, mais pendant de longues années, ce sont surtout des résultats théoriques qui sont publiés, ainsi que quelques démonstrations trouvées "à la main" ou semi-automatiquement.

Pour beaucoup, DAT est alors quelquefois devenu synonyme de Principe de Résolution. Les clauses ne sont plus seulement des énoncés intermédiaires pour le démonstrateur mais un langage d'expression des problèmes. Les nombreux exemples, difficiles, dans plusieurs domaines mathématiques, suggérés par Wos (1988), sont exprimés en anglais, puis directement dans le "langage de clauses". Il n'y a pas l'intermédiaire du langage du premier ordre. Wos (1988) écrit, à la fin de son livre, "We are […] interested in receiving any proofs of the more difficult test problems […]. If those proofs are expressed in the clause language, we would be delighted." La TPTP Problem Library, dont il sera question plus tard, n'a, pendant les premières années, contenu que des données de problèmes sous forme de clauses.

Les avantages et inconvénients du Principe de résolution et de la Déduction naturelle sont résumés dans la Fig. 1

	Principe de résolution	*Déduction naturelle*
avantages	- simplicité (une seule règle d'inférence) - résultats théoriques	- méthodes et preuves proches de celles de l'être humain - preuves faciles à lire - possibilité d'imiter les heuristiques utilisées par l'homme - mise au point plus facile (raison des échecs, localisation) - limitation des explosions combinatoires
inconvénients	- explosion combinatoire - preuves difficiles à lire - tous les concepts sont sur le même plan - signification peu évidente des fonctions de Skolem - éloignement des quantificateurs de leur portée	- non complètes (sauf Gentzen pure) - peu de possibilité d'études théoriques

Fig. 1. *Avantages et inconvénients du Principe de résolution et de la Déduction naturelle*

2.2. Exemples de démonstration

On trouvera en Fig. 2, à titre d'exemples, les démonstrations, dans les deux styles, d'un théorème très simple, la transitivité de l'inclusion

$$\forall A\ \forall B\ \forall C\ (A \subset B \wedge B \subset C \Rightarrow A \subset C)$$

avec la définition de l'inclusion

$$\forall A\ \forall B\ (A \subset B \Leftrightarrow \forall X\ (X \in A \Rightarrow X \in B))$$

On constatera que la démonstration par la déduction naturelle est facile à suivre par un lecteur humain. Au contraire, la signification de la fonction de Skolem, utilisée dans les clauses pour le Principe de Résolution n'est pas immédiate.[1]

Par le Principe de résolution	*Par la déduction naturelle*
Les clauses issues de la définition et de la négation du théorème à démontrer sont les suivantes (1) $\neg A \subset B \vee \neg X \in A \vee X \in B$ (2) $A \subset B \vee f(A,B) \in A$ (3) $A \subset B \vee \neg f(A,B) \in A$ (4) $A_0 \subset B_0$ (5) $B_0 \subset C_0$ (6) $\neg A_0 \subset C_0$ où A_0, B_0 et C_0 sont des constantes de Skolem et f une fonction de Skolem une preuve[2] : (7) : (1) et (4) $\neg X \in A_0 \vee X \in B_0$ (8) : (1) et (5) $\neg X \in B_0 \vee X \in C_0$ (9) : (3) et (6) $\neg f(A_0,C_0) \in C_0$ (10) : (2) et (6) $f(A_0,C_0) \in A_0$ (11) : (7) et (10) $f(A_0,C_0) \in B_0$ (12) : (8) et (11) $f(A_0,C_0) \in C_0$ (13) : (9) et (12) $\square$ (clause vide)	hypothèses : $A \subset B$, $B \subset C$ objets : A, B, C conclusion : $A \subset C$ remplacement de la conclusion par sa définition : $\forall X\ (X \in A \Rightarrow X \in C)$ nouvel objet : X nouvelle hypothèse (ajout) : $X \in A$ nouvelle conclusion (remplacement) : $X \in C$ nouvelle hypothèse (ajout) : $X \in B$ (car $A \subset B$ et $X \in A$) nouvelle hypothèse (ajout) : $X \in C$ (car $B \subset C$ et $X \in B$) théorème démontré

Fig. 2. *Démonstration de la transitivité de l'inclusion par le Principe de résolution et par la Déduction naturelle*

On trouvera par contre dans le paragraphe 5.2.1. un exemple de théorème pour lequel la mise sous forme de clauses est triviale et la démonstration par le Principe de Résolution immédiate, alors que la déduction naturelle demande des stratégies plus élaborées.

3. Démonstrateurs écrits par l'auteur

Les principales idées ayant conduit à l'écriture de la première puis de la deuxième version de MUSCADET étaient déjà présentes dans un ancien système appelé DATTE[3] [Pastre 1978]. Les progrès effectués depuis sont dûs en grande partie à l'analyse des nombreuses expérimentations et à des améliorations techniques.

3.1. DATTE

C'était un programme écrit en ... Fortran

---

[1] La signification de f est la suivante : si A n'est pas inclus dans B, f(A,B) est un élément appartenant à A et non à B (on peut en choisir un si on admet l'axiome du choix); si A est inclus dans B, f(A,B) peut être n'importe quoi.

[2] Avec la stratégie de l'ensemble support, la plus fréquemment utilisée dans le domaine mathématique

[3] Démonstration Automatique de Théorèmes en Théorie des Ensembles

Ses points forts :
- quelques réécritures et des règles
- une construction automatique de règles (avec les mêmes réécritures que pour les démonstrations)
- un traitement spécifique et efficace des hypothèses existentielles et des hypothèses disjonctives
- un graphe pour représenter les relations binaires
- une stratégie de démonstration (sous forme d'un plan programmé)

Ces idées sont fondamentales et sont toujours mises en oeuvre dans les démonstrateurs écrits depuis. Les progrès accomplis ont été des généralisations, des compléments et beaucoup d'améliorations techniques.

Ses points faibles :
- la syntaxe : pas de symboles fonctionels, notation polonaise préfixe, symboles sur 3 caractères !
- des programmes (on commençait à parler de règles, mais on ne parlait pas encore vraiment de bases de connaissances à l'époque ...)

Résultats

Le programme a eu de bons résultats dans les domaines suivants de théorie des ensembles : opérations ensemblistes, applications, ensembles images et images réciproques, relations d'équivalence et relations d'ordre, ordinaux.

3.2. MUSCADET

C'est un système à base de connaissances comportant :
- des bases et sous-bases de faits (théorèmes et sous-théorèmes) construites dynamiquement
- des règles et métarègles
- un moteur d'inférence écrit en en PL1 puis réécrit en Pascal
- la définition d'un langage de règles

Ses points forts :

Un langage de règles, unique, se voulant déclaratif, est utilisé pour exprimer :
- des connaissances mathématiques générales
- des connaissances spécifiques à des domaines particuliers
- des "savoir-faire"
- des stratégies de démonstration et de gestion de la démonstration
- des "super-actions" définies par des paquets de règles interprétées par le même moteur d'inférence
- des manipulations formelles
- la construction de règles à partir des définitions et des lemmes par des métarègles
- la gestion de sous-bases (sous-théorèmes issus de découpages) et le transfert d'information d'une (sous-)base à une autre

De plus :
- l'ordre des règles est en principe quelconque
- on a une structuration analogue des faits et des règles
- on peut utiliser des symboles fonctionnels et une "mise à plat" automatique donne des noms à tous les objets définis par des symboles fonctionnels et les manipulations sont ensuite faites comme s'ils étaient définis par des prédicats. En particulier un quantificateur "pour le seul ... tel que ..." permet de gérer correctement des transformations en étant traité, suivant le contexte comme un quantificateur existentiel ou comme un quantificateur universel.

Ses points faibles :
- syntaxe : c'est mieux mais on a encore une notation de listes préfixées
- peu de manipulations formelles (le système ne connaît même pas les nombres, il est seulement capable de faire des comparaisons lexicographiques et possède quelques règles de réécritures adhoc)
- certains paquets de règles sont assez illisibles car le langage n'est pas adapté à des connaissances et savoir-faire procéduraux
- les utilisateurs ne peuvent (ne veulent) pas modifier le moteur
- la gestion des sous-bases (sous-théorèmes) est assez lourde
- les découpages sont couteux en temps
- le ramasse-miettes, qui a été écrit trop tard, se fait quelque fois à un mauvais moment
- il est quelquefois indispensable de savoir comment le moteur fonctionne pour comprendre le comportement du système
- on n'a pas résisté à la tentation d'utiliser quelques trucs pour forcer le système à faire ce qu'on veut
- quelques bogues sont quelquefois impossibles à trouver

Utilisation :
MUSCADET a été conçu pour donner des connaissances et savoir faire spécifiques. Il a été utilisé par l'auteur sur les Espaces vectoriels topologiques et en Géométrie discrète. Il a démontré quelques théorèmes difficiles. Mais on peut considérer que le système a été aidé par la donnée de connaissances qui, bien que générales, étaient adaptées aux théorèmes traités. MUSCADET a d'autre part été utilisé par des stagiaires de DEA, par des chercheurs étrangers, ainsi que par deux thésards (Bazin(1993) et Spagnol (1999)).

3.3. MUSCADET version 2 (alias PUSCADET)

Il s'agit d'une traduction en Prolog, accompagnée de nombreuses améliorations, de MUSCADET

Les **motivations** qui ont conduit à cette traduction sont les suivantes :

- pouvoir exprimer dans le même langage du déclaratif pur, du prodédural pur, et tous les intermédiaires;
- utiliser les possibilités syntaxiques (notations infixes) et calculatoires (nombres) de Prolog;
- donner aux utilisateurs la possibilité d'avoir accès à tout le système, de rajouter des modules (par exemple de calcul formel, traitement du `ou` variaire)
- tout est alors explicite ou fait par l'interpréteur Prolog (relativement) standard.

On a mis l'accent sur l'expression la plus naturelle (humaine ?) des connaissances plutôt que sur l'aspect déclaratif.

Ces facilités constituent néanmoins un **piège**, car, en tant qu'utilisateurs déformés par des années de programmation classique, la tentation est très forte de programmer. Tout est alors encore explicite, mais peut devenir illisible.

MUSCADET 2.0 (version 1997 de PUSCADET) a facilité le travail en géométrie discrète, mais il était encore nécessaire de donner des règles adhoc de manipulations formelles. Un module de calcul formel, écrit par un chercheur invité, a résolu certains des problèmes.

Puis PUSCADET a été complété (la version 1999 a été officiellement appelée **MUSCADET 2.1**) pour travailler sur la base TPTP et participer à la compétition de démonstrateurs CASC-16.

3.4. Exemples d'énoncés

3.4.1. Transitivité de l'inclusion

Les expressions dans les trois systèmes du théorème exprimant la transitivité de l'inclusion

$$A \subset B \wedge B \subset C \;\Rightarrow\; A \subset C$$

et de la définition de l'inclusion

$$A \subset B \;\Leftrightarrow\; \forall X\,(X \in A \Rightarrow X \in B)$$

se trouvent en Fig. 3.

DATTE

Théorème : `TH IMP ET . INC A0 B0 . INC B0 C0 . INC A0 C0`

Définition : `DEF . INC A B QQS  X IMP . APP X A . APP X B`

MUSCADET

Théorème : `(IMP (ET (INC A B) (INC B C)) (INC A C))`

Définition : `(<=> (INC A B)(QQS X (=> (APP X A)(APP X))))`

PUSCADET

Théorème : `a inc b  et  b inc c  =>  a inc c`

ou : `qqs(A, qqs(B, A inc B et B inc C  => A inc C)`

ou (syntaxe de la TPTP library) :

`![A,B]: (subset(A,B) & subset(B,C) => subset(A,C))`

Définition : `A inc B <=> qqs(X,(X app A => X app B)))`

ou : `qqs(A,qqs(B, A inc B <=> qqs(X,(X app A => X app B))))`

ou (syntaxe de la TPTP Library) :

`![A,B]:(subset(A,B) <=> !X: (member(X,A) => member(X,B)))`

Fig. 3. *Expressions dans les trois systèmes du théorème exprimant la transitivité de l'inclusion*

On peut aussi exprimer ce théorème en utilisant des énoncés du deuxième ordre. La Fig. 4 montre un tel énoncé. Mais Prolog n'accepte pas une telle notation avec des prédicats variables. Cet énoncé doit être traduit, par un simple traitement de texte dans l'énoncé de la Fig. 5.

```
Théorème :  transitive(inc)
Définition : transitive(R)<=>
            qqs(X,qqs(Y,qqs(Z,R(X,Y) et R(Y,Z) => R(X,Z))))
```

Fig. 4. *Enoncé du deuxième ordre*

```
Théorème :  transitive(inc)
Définition : transitive(R)<=>
   qqs(X,qqs(Y,qqs(Z,..[R,X,Y]et..[R,Y,Z]=>..[R,X,Z]))))
```

Fig. 5 *Enoncé du "deuxième ordre" pour Prolog*

On voudrait alors pouvoir unifier `r(a,b)` et `..[R,X,Y]`. Mais l'unification standard ne le fait pas. Un traitement automatique du symbole "`..`" permet d'y remédier. Par exemple, les métarègles ne génèrent pas des conditions de la forme `hyp(N,..[R,X,Y])` mais à la place la condition `hyp(N,H),H =.. [R,X,Y]`[1]

3.4.2. Autre exemple

DATTE

```
DEF INT C A B QQS X  EQ . APP X C ET . APP X A . APP X B
TH IMP ET . INC A0 A1 ET . INC B0 B1
         ET INTER C0 A0 B0 INTER C1 A1 B1
       . INC C0 C1
```

MUSCADET

```
Définition :  (= (INTER A B) < X ∧(ET (APP X A)(APP X)) >)[2]
Théorème :  IMP (ET (INC A A1) (INC B B1))
   (INC (INTER A A1) (INTER B B1))
hypothèses :   (:(INTER A A1)$1)          conclusion : (INC $1 $2)
          (:(INTER B B1)$2)
```

PUSCADET

```
Définition :  A inter B = [X,X app A et X app B] [3]
ou  X app A inter B <=> X app A et X app B
ou  ![A,B,X]: (member(X,intersection(A,B))
                      <=> member(X,A) & member(X,B))

Théorème :   a inc a1 et b inc b1  =>  a inter a1  inc  b inter b1
hypothèses :   a inter a1 : a_inter_a1
   b inter b1 : b_inter_b1
conclusion :   a_inter_a1  inc  b_inter_b1

ou (usage pour la TPTP library)

Théorème :  ![A,B,A1,B1]: (subset(A,A1) & subset(B,B1)
            =>  subset(intersection(A,A1),intersection(B,B1)))
hypothèses :   x inter x1 : x_inter_x1
   x2 inter x3 : x2_inter_x3
conclusion :   x_inter_x1  inc  x_inter_x1
```

Fig. 6. *Expressions dans les trois systèmes d'un théorème de théorie des ensembles*

---

[1] Il s'agit du symbole `=..` prédéfini en Prolog. On a `r(a,b) =.. [r,a,b]`. On n'a pas `r(a,b) = ..[r,a,b]`, mais `r(a,b) =.. H, H = ..[r,a,b]`. En fait, le choix du symbole `..`, fait pour que les notations se ressemblent, est plutôt source de difficulté !

[2] < > au lieu de { } à cause des claviers des années 80 …

[3] [ ], notation de liste au lieu de { } pour faciliter la lecture et le traitement

La Fig. 6 contient les énoncés pour le théorème suivant de théorie des ensembles :

Théorème $A \subset A' \wedge B \subset B' \;\Rightarrow\; A \cap B \subset A' \cap B'$

Définition $A \cap B = \{X \mid X \in A \wedge X \in B)$

L'inconvénient des énoncés universels clos, pour la lisibilité de la trace, est que tous les objets créés, à partir de variables quantifiées, s'appellent x, x1, x2, ... C'est pourquoi, en dehors de la TPTP Library, on préfère donner des énoncés avec des constantes (ce qui correspond à la première étape de la skolemisation de la négation du théorème pour le Principe de résolution). MUSCADET peut d'autre part recevoir des énoncé universels non clos pour les définitions et les lemmes mais pas pour les théorèmes à démontrer.

3.5. Exemples de démonstrations

3.5.1. Exemple 1

La Fig. 7 donne un résumé de la démonstration par MUSCADET du théorème précédent :

$$A \subset A' \wedge B \subset B' \;\Rightarrow\; A \cap B \subset A' \cap B'$$

<table>
<tr><th>**hypothèses**
(ajoutées par des déductions successives)</th><th>**conclusion**
(chaque conclusion remplace la précédente)</th></tr>
<tr><td></td><td>$A \subset A' \wedge B \subset B' \Rightarrow A \cap B \subset A' \cap B'$</td></tr>
<tr><td colspan="2">*Règle "élim_fonc" (donne des noms aux termes apparaissant dans l'énoncé)*</td></tr>
<tr><td>C: $A \cap B$
C': $A' \cap B'$</td><td>$A \subset A' \wedge B \subset B' \Rightarrow C \subset C'$</td></tr>
<tr><td colspan="2">*règle "$\Rightarrow$" (traite les conclusions de la forme $H \Rightarrow C$)*</td></tr>
<tr><td>$A \subset A'$
$B \subset B'$</td><td>$C \subset C'$</td></tr>
<tr><td colspan="2">*règle "def_concl_1" (définition de la conclusion)*</td></tr>
<tr><td></td><td>$\forall x (x \in C \Rightarrow x \in C')$</td></tr>
<tr><td colspan="2">*règles "$\forall$" et "$\Rightarrow$" (la règle "$\forall$" traite les conclusions universelles)*</td></tr>
<tr><td>$x1 \in C$</td><td>$x1 \in C'$</td></tr>
<tr><td colspan="2">*règles "$\cap$11" et "$\cap$12" (règles construites à partir de la définition de l'intersection)*</td></tr>
<tr><td>$x1 \in A$
$x1 \in B$</td><td></td></tr>
<tr><td colspan="2">*règle "$\subset$" (règle construite à partir de la définition de l'inclusion)*</td></tr>
<tr><td>$x1 \in A'$
$x1 \in B'$</td><td></td></tr>
<tr><td colspan="2">*règle "$\cap$2" (règle construite à partir de la définition de l'intersection)*</td></tr>
<tr><td>$x1 \in C'$</td><td></td></tr>
<tr><td colspan="2">*règle "stop" (règle d'arrêt, on vient d'obtenir en hypothèse la conclusion que l'on voulait démontrer)*</td></tr>
</table>

Fig. 7. *Résumé d'une démonstration de MUSCADET*

3.5.2. Exemple 2

Soient f, g, h trois applications de A dans B, B dans C, C dans A. Si, parmi les trois applications $h_{o}g_{o}f$, $g_{o}f_{o}h$, $f_{o}h_{o}g$; *deux sont injectives (resp. surjectives) et la troisième surjective (resp. injective), alors f, g et h sont bijectives.*

Il y a en fait six théorèmes indépendants à démontrer : la bijectivité de chacune des trois applications f, g et h dans chacun des deux cas (deux composées injectives et une surjective ou l'inverse)

MUSCADET démontre ces théorèmes en créant des images et des antécédants jusqu'à obtenir tous ceux qui sont nécessaires à une démonstration directe[1]. Il crée à peu près autant d'éléments inutiles que d'éléments nécessaires. On trouvera en Fig. 8. les créations nécessaires pour la démontration de la bijectivité de h dans le cas où $h_o g_o f$ est injective et les deux autres composées surjectives.

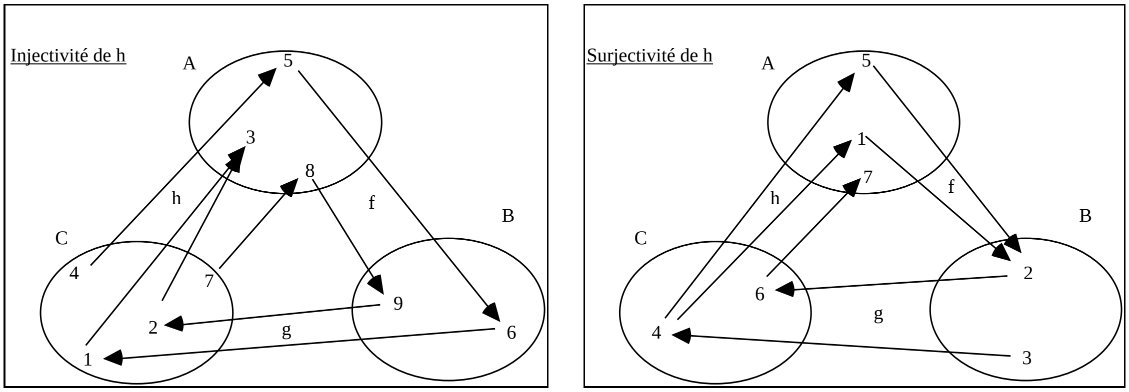


Fig. 8. *Objets créés dans la démonstration d'un théorème sur les applications*

3.6. Exemples de règles et de super-actions

On trouvera en Fig. 9. et 10. des exemples de règles et super-actions en MUSCADET et en PUSCADET.

Le paramètre N, qui n'existait pas dans la première version de MUSCADET sert à reprérer la sous-base (sous-théorème) sur lequel on est en train de travailler.

On notera l'utilisation importante du "si ... alors ... sinon ..." (en Prolog (... -> ... ; ...)), moins déclarative mais plus naturelle dans certains cas (par exemple `ajhyp`) que les règles en vrac.

Le traitement des hypothèses disjonctives est un exemple montrant également comment un peu moins de déclarativité donne plus de naturel.

La règle de MUSCADET

```
REGLE OU SI CONCL C HYP H EGAL(H(OU A B))
         ALORS SUPHYP H NOUVCONCL(ET(=>A C)(=>B C))
```

qui donne, après découpage par la règle `ET` et l'application de la règle `=>` deux sous-théorèmes ayant comme hypothèse supplémentaire, respectivement `A` et `B`, a d'abord été traduite en PUSCADET par la règle suivante faisant le même travail :

```
regle(N, ou) :- hyp(N, A ou B), not hyp_traite(N, A ou B), [2]
                concl(N, C),
                ecrire1([N, traitement,de,l,hypothese,disjonctive,A ou B]),
                nouvconcl(N, (A => C) et (B => C)),
                ajhyp_traite(N, A ou B).
```

---

[1] DATTE démontrait déjà ces théorèmes, avec cependant une petite facilité supplémentaire : il recevait la définition de la composée de *trois* fonctions, ce qui évitait de créer quelques éléments inutiles. On trouvera dans (Pastre 78) les démontrations détaillées de deux des théorèmes (un des plus faciles et un des plus difficiles)

[2] Le prédicat `hyp_traite` permet une mémorisation qui a dû être ajoutée dans de nombreux cas pour éviter que l'hypothèse soit traitée à nouveau. Ceci était en général inutile dans MUSCADET, car le moteur d'inférence de MUSCADET n'appliquait jamais une règle deux fois pour les mêmes instanciations. D'une part, cela avait, dans quelques cas, posé des problèmes qui ont été résolus en ajoutant des conditions ne servant qu'à provoquer une instanciation différente. D'autre part, l'interpréteur Prolog ne donne pas les instanciations ayant permis les diverses unifications qu'il fait, et garder ce test aurait nécessité de réécrire l'unification. Il a été jugé préférable de mémoriser les éléments traités et de tester dans la règle que l'hypothèse n'a pas encore été traitée. (Cette condition pourrait être ajoutée automatiquement.)

MUSCADET

```
REGLE STOP1 SI CONCL C HYP C ALORS NOUVCONCL VRAI

REGLE => SI CONCL(=>A B)ALORS AJHYP A NOUVCONCL B

POUR NOUVCONCL C REGLE NOUVCONCL1 ELIFON C AFFECTER(CONCL C)
                                  message(nouvelle conclusion C)

POUR AJHYP H REGLE AJHYP1 SI PREMIER(H ET) ELTLISTE(A H) NOMDIF(A ET)
                          ALORS AJHYP A
             REGLE AJHYP4 SI EGAL(H(=X Y))NOMDIF(X Y) NONHYP H
                          ALORS AJOUTER(HYP H) message(ajouter hypothese H)
                                ajobj H STOP
             REGLE AJHYP5 SI PREMIER(H P) NOMDIF(P ET = QQS SEUL) NONHYP H
                          ALORS AJOUTER(HYP H) message(ajouter hypothese H)
                                ajobj H STOP
             REGLE AJHYP6 SI ATOME H NONHYP H
                          ALORS AJOUTER(HYP H) message(ajouter hypothese H) STOP
             REGLE AJHYP7a SI EGAL(H(SEUL (:A X)Y)) HYP(:A X1)
                           ALORS LIGNE message(remplacer X par X1 dans Y)
                                 REMPLACER(Y X X1) AJHYP Y SUPHYP H STOP
             REGLE AJHYP7b SI EGAL(H(SEUL (:A X)Y)) NONHYP(:A X1) NONOBJ X
                           ALORS AJHYP (:A X) AJHYP Y SUPHYP H
                                 AJOUTER(OBJ X) message(ajouter objet X) STOP
             REGLE AJHYP7c SI EGAL(H(SEUL (:A X)Y)) NONHYP(:A X1) OBJ X
                           ALORS CREER X AJHYP (:A X) AJHYP Y SUPHYP H
                                 AJOUTER(OBJ X) message(ajouter objet X) STOP
             REGLE AJHYP8 SI EGAL(H(QQS(APP X A)C))
                          ALORS CREER REGHYP
                                CONSREG((REGHYP) H REGLACTIV)
                                ORDOREG STOP
             REGLE AJHYP9 SI EGAL(H(QQS X A)) ATOME X
                          ALORS CREER REGHYP
                                CONSREG((REGHYP) A REGLACTIV)
                                ORDOREG STOP

REGLE ET SI CONCL C PREMIER(C ET)ELTLISTE(A C)NOMDIF(A ET)ALORS DEM A

POUR DEM A REGLE DEM1 SI NUMERO N
            ALORS CREER N AJOUTER(SOUSTH((NUMERO N)(CONCL A)))
                message(ajouter le sous theoreme de numero N et de conclusion A)
                 COPITEM DEMSOUSTH RETOURDEM A
                 SUPPRIMER SOUSTH message(supprimer le sous theoreme N)
```

Fig. 9. *Exemples de règles et super-actions en MUSCADET*

Puis un théorème du type suivant a soulevé un problème :
On a les deux hypothèses disjonctives suivantes : `a1 ou a2 ou a3` et `b1 ou b2 ou b3 .`
Un premier découpage (première hypothèse disjonctive) donne les sous-théorèmes suivants dont on a indiqué seulement les hypothèses non encore traitées :

1 2	
b1 ou b2 ou b3	b1 ou b2 ou b3
a1 a2 ou a3	

puis le découpage de la deuxième hypothèse disjonctive donne

11	12	21	22
a1	a1	a2 ou a3	a2 ou a3
b1	b2 ou b3	b1	b2 ou b3

etc.

PUSCADET

```
regle(N,stop1):- concl(N,C), hyp(N,C), nouvconcl(N,vrai).

regle(N,=>) :- concl(N, A => B), ajhyp(N, A), nouvconcl(N,B).

nouvconcl(N, C) :- not concl(N, C), retractall(concl(N, _)),
           assert(concl(N, C)),
           ecrire1([N, nouvelle,conclusion, C]).

ajhyp(N, H) :- ( H = A et B -> ajhyp(N, A), ajhyp(N,B)
       ; hyp(N, H) ->  true
       ; H = (X = X) -> true
       ; H = ..[R, X, Y]  -> H1 =..[R, X, Y], ajhyp(N, H1)
       ; H = ..[F, X]:Y  -> Y1 =..[F, X], ajhyp(N, Y1:Y)
       ; H = seul(A:X,Y) -> (hyp(N,A:X1) -> ecrire1([remplacer,X,par,X1,dans,Y])
                            ; creer_objet(N,x,X1),ajhyp(N,A:X1)
                            ),
                            remplacer(Y,X,X1,Y1),ajhyp(N,Y1)
       ; H = non seul(FX:Y,P) -> ajhyp(N,seul(FX:Y,non P ))
       ; H = qqs(_, _) -> creer_nom_regle(reghyp,Nom),
        consreg( H, _, N, Nom, [])
       ; H = A => B -> creer_nom_regle(reghyp,Nom),
        consreg( H, _, N, Nom, [])
       ; assert(hyp(N, H)),
         ecrire1([N, ajouter,hypothese,H])
       ).

regle(N, et) :- concl(N, A et B), demconj(N, 0, A et B), concl(N, vrai).

demconj(N, I, A) :- (A = B et C -> dem(N, I, B), I1 is I+1, demconj(N, I1,C)
                    ; dem(N, I, A)
                    ).
dem(N, I, A) :-  I1 is I+1,  N1 is 10*N+I1,
        nl,ecrire1([************************************************]),
        ecrire([sous-theoreme, N1,*****]),
        nouvconcl(N1,A), copitem(N, N1),
        assert(sousth(N, N1)),
        nl,ecrire([-----------------------------------------------]),
        ecrire([creation,du,sous-theoreme,N1]),
        appliregactiv(N1),
        (concl(N1, vrai) -> concl(N, C), retirer(A, C, C1),
                            nouvconcl(N, C1)
        ; true).
```

Fig. 10. Exemples de règles et super-actions en PUSCADET

Finalement, on aura :

11	121	122	211	212	2211	2212	2221	2222
a1	a1	a1	b1	b1	a2	a2	a3	a3
b1	b2	b3	a2	a3	b2	b3	a2	b3

Le mélange des cas est encore pire si on a une troisième hypothèse disjonctive.
On préfère avoir le découpage suivant, plus naturel :

1 2	3	
b1 ou b2 ou b3	b1 ou b2 ou b3	b1 ou b2 ou b3
a1 a2	a3	

puis

11	12	13	21	22	23	31	32	33
a1	a1	a1	a2	a2	a2	a3	a3	a3
b1	b2	b3	b1	b2	b3	b1	b2	b3

Ce découpage sera obtenu par la règle suivante :

```
regle(N, ou) :- hyp(N, A ou B), not hyp_traite(N, A ou B),
                concl(N, C),
                ecrire1([N, traitement,de,l,hypothese,disjonctive,A ou B]),
                hypou(A ou B => C , T),  % decoupage de tous les ou
                nouvconcl(N,T),
                ajhyp_traite(N, A ou B).
```

utilisant le prédicat Prolog `hypou`, procédural et récursif [1]

```
hypou(A ou B => C, (A => C) et BC) :- hypou(B => C, BC),!.
hypou(T,T).
```

qui construit directement l'énoncé `T = (A1 => C ) et (A2 => C) et (A3 => C) et ...` à partir d'une hypothèse `A1 ou A2 ou A3 ou ... .`

Une amélioration (mais ce n'est pas la plus urgente) consisterait à ce que MUSCADET comprenne les "…" dans une notation de la forme `A ou B ou` … et construise automatiquement ce prédicat Prolog. Les notations avec des "…" sont en effet très courantes en mathématiques.

4. The TPTP Problem Library

La TPTP Problem Library (Thousands of Problems for Theorem Provers) est une base de problèmes conçue et maintenue par Geoff Sutcliffe (Australie) et Christian Suttner (Allemagne) depuis 1993 pour tester et évaluer des systèmes de démonstration automatique de théorèmes. Elle se trouve sur le Web à l'adresse http://www.cs.jcu.edu.au/~tptp

4.1. La base et les problèmes

Jusqu'en 1996, la base ne comportait que des énoncés sous forme de clauses (format CNF : Clause Normal Form). Depuis 1997, elle comporte également des énoncés du calcul des prédicats du premier ordre (format FOF : First Order Formula).

En	1998, il y avait	3275 problèmes en format	CNF, dont	2296 abstraits et	86 génériques,
		347	FOF	332	1
	1999,	3334	CNF	2352	86
		670	FOF	632	1

Un même problème abstrait peut être exprimé sous diverses formulations et donner ainsi plusieurs problèmes. Un problème générique est un problème général (qui n'est pas du premier ordre) dont quelques exemples (d'ordre 0 ou 1) correspondant à différentes tailles du problème figurent dans la base, par exemple, le théorème générique
(p1 <=> (p2 <=>...(pN <=> (p1 <=> (p2 <=>...<=> pN)...)
a deux instances dans la base, correspondant à N = 5 et N=14 :

```
        p1<=>(p2<=>(p3<=>(p4<=>(p5<=>(p1<=>(p2<=>(p3<=>(p4<=>p5))))))))
```

et

```
p1<=>(p2<=>(p3<=>(p4<=>(p5<=>(p6<=>(p7<=>(p8<=>(p9<=>(p10<=>(p11<=>(p12<=>(p13<=
>(p14<=>(p1<=>(p2<=>(p3<=>(p4<=>(p5<=>(p6<=>(p7<=>(p8<=>(p9<=>(p10<=>(p11<=>(p12
                   <=>(p13<=>p14)))))))))))))))))))))))))))
```

Les problèmes relèvent de 28 domaines correspondant à 5 grands thèmes présentés dans la Fig. 11, ainsi que le nombre de problèmes, pour chaque domaine, en 1999.

n1 (n2, n3) signifie qu'il y a n1 problèmes dans la base correspondant à n2 problèmes abstraits et n3 problèmes génériques. On n'a indiqué ces nombres uniquement pour les domaines où il existe des problèmes en format FOF.

On trouvera en annexes 1 à 4 des exemples de théorèmes dans les domaines SYN, MGT et COM. Le domaine SET sera étudié en section 5.2.1.

---

[1] immédiat à écrire pour toute personne ayant une petite habitude de la programmation Prolog

	domaines	*format FOF*	*format CNF*
Logique	Logique combinatoire	2	279 (257)
	Calcul logique		
	Modèles de Henkin		
Mathématiques	**Théorie des ensembles (SET)**	**321** (300) [**5** en 98]	696 (561) [695 (562) en 98]
	Théorie des graphes		
	Algèbre : Groupes	1	374 (203, 50) [363 (193) en 98]
	+ (Boole, Robbins, Distr, Treillis, Anneaux, Algèbre générale)		
	Théorie des nombres		
	Topologie		
	Analyse		
	Géométrie		
	Théorie des corps		
	Théorie des catégories		
Informatique	**Informatique théorique (COM)**	3 (1)	6 (4)
	Représentation des connaissances		
	Planning		
	Vérification de programmes		
Engineering	Conception de circuits		
	Vérification de circuits		
Sciences sociales	**Management (MGT)**	**53** (41)	0
Autre	**Syntaxique (SYN)** [280 (279, 1) en 98]	**286** (283, 1)	479 (469, 27)
	Puzzles [2 en 98]	3	57 (36, 4)
	Miscellaneous	1	13 (9, 3)

Fig. 11. *Nombre des problèmes dans les différents thèmes et domaines de la TPTP Problem Library en 1999*

4.2. Les compétitions

Elles sont organisées depuis 1996 par Geoff Sutcliffe (Australie) et Christian Suttner (Allemagne) dans le cadre des conférences CADE (Conference on Automated DEduction).
Elles comportent quatre divisions dont certaines comportent plusieurs catégories (Fig. 12).

Divisions	*Catégories*	*clauses de Horn*	*Egalité*
MIX : format CNF, sans égalité unitaire	HEQ	oui	oui
	HNE	oui	non
	NEQ	non	oui
	NNE	non	non
	PEQ		oui, pure
UEQ : format CNF, avec égalité unitaire			
SAT : format CNF, non théorèmes (cad ensembles de clauses satisfaisables)			
FOF : format FOF	FEQ		oui
	FNE		non

Fig. 12. *Divisions et catégories des compétitions*

Les logiciels doivent être installés sur les machines du site de la compétition une à deux semaines avant, puis passer des tests effectués par les organisateurs, c'est-à-dire être capables de démontrer des théorèmes faciles et être valides (ne pas démontrer des théorèmes qui n'en sont pas). Des tests de validité sont encore effectués après la compétition pour les vainqueurs de chaque catégorie.

Le classement se fait en fonction du nombre de problèmes résolus. En cas d'égalité, en fonction des temps d'exécution.

En 1998, 18 démonstrateurs ont participé à la compétition, dont 3 dans la division FOF. La Fig. 13 donne les résultats, pour ces 3 concurrents, pour toutes les divisions où ils ont participé.

	Concurrents	SPASS	Bliksem	Otter
MIX	9	3ème		6ème
HEQ		2ème		**1er**
HNE		3ème		5ème
NEQ		3ème		4ème
NNE		**1er**		2ème
PEQ		6ème		7ème
UEQ	8	6ème		2ème
SAT	4	**1er**		
FOF	3 **40** problèmes	**1er** **39** résolus	2ème **21** résolus	3ème **2** résolus
FEQ	3 **20** problèmes	2ème **19** résolus	**1er** **19** résolus	3ème **2 résolus**
FNE	3 **20** problèmes	**1er** **20** résolus	2ème **2** résolus	3ème **0** résolu

Fig. 13. *Résultats de la compétition 1998, de SPASS, Bliksem et Otter*

On remarquera les bons résultats d'Otter en format CNF, mais non en format FOF, ce qui montre, si cela était encore nécessaire, que la mise automatique sous forme d'un bon ensemble de clauses n'est pas facile.

Les 40 théorèmes de la division FOF comprenaient 13 MGT, 25 SYN, 1 COM et 1 PUZ. PUSCADET en a démontré 8 (de la catégoire FEQ, avec égalité).

D'autre part, Bliksem a par la suite été disqualifié, car ayant été trouvé invalide au cours des tests post-compétition.

5. PUSCADET et la base TPTP

Plusieurs phases de travail ont été nécessaires pour permettre à PUSCADET de participer raisonnablement à la compétition 1999. Il a d'abord fallu poursuivre la traduction de MUSCADET (et même de DATTE) en Prolog, puis analyser la base TPTP dans l'optique de l'utilisation de MUSCADET, expérimenter et ajouter ou modifier des connaissances pour traiter des théorèmes de la base qui n'étaient pas démontrés, proposer de nouveaux théorèmes (dans la syntaxe et l'esprit de la base) pour enrichir la base, enfin mettre au point un exécutable respectant les contraintes de la compétition.

5.1. Poursuite de la traduction en Prolog

La traduction ayant été commencée dans le cadre du travail en géométrie discrète, de nombreuses spécificités de MUSCADET n'avaient pas été incorporées, ou bien seulement dans un forme simplifiée. Certaines possibilités de DATTE, même, n'avaient jamais été traduites en MUSCADET ni en PUSCADET.

Actuellement, tout le "programme" DATTE est réalisé en PUSCADET, et tous les théorèmes démontrés par DATTE le sont par PUSCADET, sauf ceux concernant les ordinaux qui nécessitent des lemmes et qui n'ont pas été tous testés. Le démonstrateur général MUSCADET est entièrement traduit sauf le traitement des conclusions existentielles dans le cas le plus complexe. Les connaissances spécifiques à certains domaines (par exemple Espaces vectoriels topologiques) n'ont pas été traduits. Ces connaissances étaient données directement dans le langage de MUSCADET sous forme opérationnelle. L'orientation future sera plutôt de les donner uniquement par des énoncés du premier ordre et d'écrire des métarègles les traduisant dans les formes opérationnelles. L'aide donnée au démonstrateur sera ainsi plus claire et de tels problèmes pourront rentrer dans la base TPTP.

5.2. Analyse de la base et expérimentations de PUSCADET

Les essais ont été faits sur les seuls domaines ayant des énoncés dans le format FOF. L'analyse a porté sur les domaines SET, MGT, SYN et COM.

5.2.1. Théorie des ensembles

Le domaine SET ne comportait en 1998 que cinq théorèmes dont quatre tournent autour des paradoxes de la théorie des ensembles et le cinquième est un théorème facile sur l'égalité d'ensemble.

1 - *il n'existe pas d'ensemble formé des éléments qui n'appartiennent pas à eux-mêmes*

$\neg \exists X\ \forall Y\ (Y \in X \Leftrightarrow Y \notin Y)$ (SET043+1)

2 - *s'il existe un ensemble formé des éléments qui appartiennent à eux-mêmes, alors il n'est pas vrai que tout ensemble a un complément*

$\exists Y\ \forall X\ (X \in Y \Leftrightarrow X \in X) \Rightarrow \neg \forall X1\ \exists Y1\ \forall Z\ (Z \in Y1 \Leftrightarrow Z \notin X1)$ (SET044+1)

3 - si on a l'axiome *pour tout Z il existe un ensemble formé des éléments qui appartiennent à Z et n'appartiennent pas à eux-mêmes*, alors on a le théorème *il n'existe pas d'ensemble universel* (SET045+1)

axiome : $\forall Z\ \exists Y\ \forall X\ (X \in Y \Leftrightarrow (X \in Z \wedge X \notin X))$ théorème : $\neg \exists Z\ \forall X (X \in Z)$

4 - *il n'existe pas d'ensemble formé des éléments X tels qu'il n'existe pas de chaîne* $X \in Z \in X$

$\neg \exists Y\ \forall X\ (X \in Y \Leftrightarrow \neg \exists Z\ (X \in Z \wedge Z \in X))$ (SET046+1)

5 - *l'égalité d'ensemble est une relation symétrique*

axiome : $\forall X\ \forall Y\ (X =_{ens} Y \Leftrightarrow \forall Z\ (Z\in X \Leftrightarrow Z\in Y))$
théorème : $\forall X\ \forall Y\ (X =_{ens} Y \Leftrightarrow Y =_{ens} X)$ (SET047+1)

Or, dans tout le travail que j'avais fait précédemment, les définitions ensemblistes utilisaient des propriétés relatives à des ensembles (ce qui est un moyen, en théorie naïve, de ne pas avoir de paradoxes). Ainsi PUSCADET savait traiter des énoncés de la forme

$\forall X\ (X\in A \Rightarrow p(X))$

ou $\forall X\in A\ p(X)$ avec des quantificateurs relativisés,

ainsi que $\forall X\ (q(X) \Rightarrow p(X))$

qui donnaient des règles de la forme

si hyp $X\in A$ *alors ajhyp* $p(X)$
si hyp $q(X)$ *alors ajhyp* $p(X)$

instanciant X en partie <conditions>.

Mais PUSCADET ne savait pas traiter des énoncés de la forme

$\forall X\ p(X)$

Ce problème a été résolu en utilisant, dans la construction des règles, le fait d'être un objet mathématique (déjà présent pour mémoriser tous les objets mathématiques créés en cours de démonstration), uniquement s'il n'y a pas d'autre condition instanciant X (car s'il y a d'autres conditions, elles sont certainement plus sélectives). On a ainsi la règle

si objet X alors ajhyp p(X)

pour le premier cas, et pas de changement pour les règles construites dans les autres cas.

Quatre des cinq théorèmes ont alors été démontrés par la version standard, le deuxième (SET044+1) ne l'est que par une variante obtenue en modifiant l'ordre de traitement des hypothèses existentielles et disjonctives.

Le premier théorème (SET043+1) est un exemple de théorème trivial pour le Principe de Résolution, dont la démonstration par MUSCADET est plus complexe. Néanmoins, les méthodes utilisées par MUSCADET sont les mêmes que celles utilisées dans des théorèmes plus complexes où les démonstrateurs utilisant le Principe de Résolution échouent.

Pour le Principe de Résolution, la négation du théorème donne les deux clauses suivantes : $Y\in Y \vee Y\in f(X)$

$Y\notin Y \vee Y\notin f(X)$

qui donnent immédiatement la clause vide avec l'unification $Y = f(X)$.

La démonstration de MUSCADET se trouve en Fig. 14.

objet	hypothèses	règles	conclusion
			$\neg \exists X\ \forall Y\ (Y\in X \Leftrightarrow Y\notin Y)$
	$\exists X\ \forall Y\ (Y\in X \Leftrightarrow Y\notin Y)$		faux
a			
		r0 : si $Y\in a$ et $Y\in Y$ alors Faux r1 : si $Y\notin Y$ alors $Y\in a$ r2 : si objet Y alors $Y\in Y \vee Y\in a$	
	$a\in a \vee a\in a$ (règle r2) $a\in a$ (règle ou) faux (règle r0)		*démontré*

Fig. 14. *Démonstration par PUSCADET du théorème SET043+1*

Plusieurs centaines de problèmes classiques existaient dans le format CNF, mais, déchiffrer un ensemble de clauses CNF pour reconstruire l'énoncé du premier ordre est un travail encore plus long et fastidieux que construire les clauses à partir de l'énoncé du premier ordre. Je n'ai donc pas travaillé sur ces problèmes à partir de ces énoncés, mais sur des problèmes classiques similaires que j'ai (ou j'avais) déjà moi-même formulés (voir section 5.3).

5.2.2. Logique

Le domaine SYN comporte des énoncés "syntaxiques" n'ayant pas de sémantique évidente. Il s'agit de logique abstraite, d'ordre 0 ou 1. Des exemples d'énoncés et de démonstrations par PUSCADET sont donnés en annexes 1 et 2.

Certains énoncé sont triviaux. Il ne faut pas se fier au commentaire donné, par exemple, pour le premier exemple de l'annexe 1 (SYN040+1) où il est dit qu'il s'agit du théorème le plus difficile démontré par le Logic Theorist [1]. Le commentaire est seulement historique et valable pour des démonstrateurs uniquement syntaxiques qui ne connaîtraient pas la sémantique des connecteurs logiques $\Rightarrow$ et $\neg$, ce qui n'est pas le cas avec le Principe de Résolution ni la Déduction naturelle.

Certains théorèmes n'étaient pas démontrés par PUSCADET car ils contenaient des sous-formules que l'on ne rencontre jamais dans un énoncé traduisant un problème réel, et que MUSCADET ne générait jamais comme formule intermédiaire, par exemple $\neg\neg p$ (SYN041+1) ou $\neg (p \Rightarrow q)$ (SYN001+1). Il a été facile de rajouter des réécritures élémentaires.

D'autres théorèmes sont constitués de grosses formules, sans définition de prédicats intermédiaires et MUSCADET serait sans doute capable de les démontrer mais dans un temps déraisonnable que je ne lui ai jamais laissé. C'est le cas du théorème SYN007+1.14 qui est une instance, pour N=14 du théorème générique (voir section 4.1 et annexe 1)

p1 <=> (p2 <=>...(pN <=> (p1 <=> (p2 <=>...<=> pN)…)

et qui nécessite un nombre trop important de découpages. Le théorème SYN007+1.005, pour N=5, de taille moindre, est démontré en 90 secondes[2] (SPARC10 - 150 MHz - SWI-Prolog 2.1.3) et après 159 découpages.

Ces théorèmes sont des théorèmes artificiels pour lesquels les méthodes naturelles de MUSCADET ne sont pas du tout adaptées. En ce qui me concerne, pour démontrer ces théorèmes, ma première idée a été d'utiliser la traduction de la logique vers l'arithmétique (vrai / 1, faux / 0, $\wedge$ / *, $\neg$ / +1, $\neg \Leftrightarrow$ / +) et ils deviennent très faciles[3]. Cela confirme qu'il n'y a pas de méthode universelle pour démontrer tous les types de théorèmes. Il doit être possible de donner cette méthode au système ainsi que des caractéristiques des théorèmes pour lesquelles elle est conseillée.[4]

Certains théorèmes ne sont pas démontrés et les raisons n'en ont pas encore été étudiées.

Enfin, certains théorèmes correspondent à des traductions dans le calcul des prédicats du premier ordre de problèmes exprimés en logique K multi-modale et les énoncés sont énormes (jusqu'à 26 pages !). Ces énoncés ont été proposés par l'auteur de SPASS qui semble particulièrement adapté à ce type de problèmes (dits ALC problèmes). Ils existent aussi sous forme de clauses dans d'autres divisions que la division FOF. Dans les discussions animées qui ont précédé la compétition 1999 (voir section 6.2), certains ont suggéré que ces problèmes ne puissent pas être éligibles (voir section 6.1).

5.2.3. Management

Pour le domaine MGT, les difficultés ont d'abord été dues au fait qu'on n'énonce pas des problèmes d'économie comme des problèmes de mathématiques et que certaines formules ne pouvaient être traitées par PUSCADET. Dans certains cas, comme précédemment, il a suffi de rajouter des règles simples.

Deux cas ont été plus difficilement résolus.

Le premier concerne des axiomes ou des hypothèses de la forme $\forall X\, (p(X) \Rightarrow q(f(X)))$ (exemple MGT021+1) à partir desquels MUSCADET construit des règles de la forme

si <conditions à partir de p(X)> et si on a l'hypothèse Y:f(X) alors ajouter l'hypothèse q(Y)

Dans le contexte MGT, il faut aussi construire des règles de la forme

si <conditions à partir de p(X)> alors créer Y:f(X) et ajouter l'hypothèse q(Y)

qui crée tout de suite un objet *Y*, ou

---

[1] Et encore, Jacques Pitrat m'a signalé que le Logic Theorist n'avait montré l'implication que dans un sens.

[2] Les temps moyens sont plutôt de l'ordre de quelques seceondes

[3] p $\Leftrightarrow$ q étant traduit par 1 + p + q, la formule devient 1+p1+1+p2+…+1+pN+1+p1+1+p2+…+1+p(N-1) +pN qui est impair donc égale à 1, c'est-à-dire vrai.

[4] On peut aussi faire un raisonnement par récurrence et utilisant la sémantique. Supposons que le théorème soit vrai pour N-1 variables propositionnelles. Alors si p1 est vrai la formule est équivalente à (p2 <=>...(pN <=> (p2 <=>...<=> pN)…) qui est vraie par hypothèse de récurrence. Si p1 est faux, la formule est équivalente à $\neg$ (p2 <=>...(pN <=> $\neg$ (p2 <=>...<=> pN)…) qui est aussi équivalente à (p2 <=>...(pN <=> (p2 <=>...<=> pN)…). Un tel raisonnement est hors de portée des démonstrateurs actuels.

si <conditions à partir de $p(X)$*> alors ajouter l'hypothèse* $\exists Y\ (Y{:}f(X) \wedge q(Y))$
qui créera l'objet Y si et quand l'hypothèse existentielle sera traitée.

Ces règles peuvent être très gênantes dans d'autres contextes, surtout sous la première forme.

En mathématiques, s'il y a l'idée de création dans cet énoncé, il sera plutôt exprimé sous la forme $\forall X\ (p(X) \Rightarrow \exists Y\ q(Y)))$ et conduira à l'écriture d'une règle ajoutant des hypothèses existentielles qui créeront de nouveaux objets quand elles seront traitées. Le faire trop vite peut conduire à la création d'un chaîne infinie de nouveaux objets.

Des énoncés de la forme $p(X,...) \Rightarrow q(f(X,...))$ se rencontrent également, issus d'une définition $\forall X...(q(f(X,...)) \Leftrightarrow p(X,...))$ du symbole fonctionnel *f* (c'est, avec deux variables l'exemple de la définition de l'intersection vue en section 3.4.2.). Dans ce cas, il ne faut pas créer ce nouvel objet. MUSCADET dispose du mot-clef `definition` pour donner les énoncés qui sont des définitions. Pour la TPTP Library, il a fallu reconnaître automatiquement si un énoncé est une définition ou une propriété.

Voici quelques exemples d'axiomes, lemmes ou sous-formules issues de définitions, illustrant tous ces cas :

<conditions sur a et b> $\Rightarrow$ *taux_de_croissance(a) < taux_de_croissance(b)*

(axiome) il faut introduire les taux de croissance

$A \subset B \Rightarrow \complement B \subset \complement A$

(lemme) ne considérer que les complémentaires qui existent déjà

$convexe(E) \wedge X \in E \wedge Y \in E \Rightarrow segment(X,Y) \subset E$

(issu de la définition de convexe) introduire les segments avec prudence

$X \in A \wedge Y \in B \Rightarrow X \in A \cap B$

(issu de la définition de l'intersection) ne considérer que les intersections qui existent déjà

$ouvert(U) \wedge \ldots \Rightarrow \exists U\ (V \subset U \wedge ouvert(V) \wedge \ldots)$

(propriété exprimée sous forme de savoir-faire) l'idée de création est explicite

Le deuxième cas difficile concerne des axiomes (lemmes) comportant des hypothèses universelles (exemple MGT031+1) de la forme

$$P(X) \wedge \forall Y\ Q(X,Y) \Rightarrow R(X)$$

Je n'en avais jamais rencontré dans les exemples que j'ai traités en mathématiques, sans doute parce que le mathématicien aurait alors défini un nouveau concept

$$S(X) \Leftrightarrow \forall Y\ Q(X,Y)$$

et l'axiome précédent serait exprimé par

$$P(X) \wedge S(X) \Rightarrow R(X)$$

Il a donc fallu rajouter des métarègles construisant, à partir du lemme

$$P(X) \wedge \forall Y\ Q(X,Y) \Rightarrow R(X)$$

des règles du type suivant

si< conditions construites à partir de P >
conclusion C
< conditions de ressemblance entre C et les hypothèses actuelles >
alorsnouvelle conclusion $\forall Y\ Q(X,Y) \wedge (R(X) \Rightarrow C)$

5.2.4. Informatique théorique

La base TPTP comporte, en format FOF, trois modélisations du théorème affirmant l'indécidabilité de la terminaison des programmes (annexe 4). PUSCADET réussissait à en démontrer une fin 1998. La version actuelle de PUSCADET ne le démontre plus. Ce cas devra être analysé.

5.2.5. Résultats

Cette phase de tests sur la base TPTP a été provisoirement stoppée pour se consacrer aux suivantes (enrichissement de la base, mise au point d'un exécutable dans les conditions de la compétition).

Fin 1998, PUSCADET démontrait 4 théorèmes SET sur 5 (et le 5ème par une variante), 1 théorème COM sur 3. Il avait progressé, sans trop de difficulté dans le domaine SYN, plus laborieusement dans le domaine MGT.

Il réussissait à démontrer 8 des 40 théorèmes proposés dans la division FOF à la compétition 1998 (6 des 13 MGT, 2 des 25 SYN), ce qui était mieux que les performances d'Otter.

5.3. Enrichissement de la base TPTP

Il est prévu que la base soit enrichie par les utilisateurs, j'ai donc souhaité proposer tous les théorèmes de théorie des ensembles démontrés par MUSCADET (et même pour la plupart par DATTE). Pour certains, parmi les plus compliqués, j'étais persuadée que les démonstrateurs basés sur le Principe de Résolution, ne pourraient pas les démontrer (par exemple ceux du type de l'exemple de la section 3.5.2).

Outre la traduction dans la syntaxe TPTP, j'ai essayé d'utiliser des notations, des conventions et des définitions proches de celles utilisées pour les théorèmes en format CNF. Après déchiffrage laborieux des clauses de ces problèmes CNF, cela n'a été fait partiellement, car les choix faits par les précédents auteurs étaient assez axiomatiques alors que ma base personnelle reposait plutôt, au moins pour les notions élémentaires, sur la théorie naïve des ensembles et une pratique usuelle des mathématiques courantes.

Plusieurs difficultés plus sérieuses sont apparues.

La base de connaissances de MUSCADET, bien que générale, connaissait le symbole d'appartenance à un ensemble, utilisé dans plusieurs règles et métarègles, en particulier pour le traitement des quantificateurs relativisés ainsi que dans le traitement de plusieurs cas particuliers de conclusions existentielles. Il a fallu réécrire ces règles sans cette facilité, car l'écriture des énoncé dans la base TPTP ne doit comporter aucun symbole non défini autre que logique.

Ensuite, la base de connaissances de MUSCADET utilisait quelques méta-symboles qui ne sont pas du premier ordre.

D'abord des prédicats `definition` et `lemme` permettaient de déclarer que des énoncés étaient soit des définitions, soit des lemmes, et ces énoncés n'étaient pas traités exactement de la même façon. En effet, on ne considère pas de la même façon la définition de l'inclusion

$$A \subset B \;\Leftrightarrow\; \forall X\,(X\in A \Rightarrow X\in B)$$

et le lemme exprimant la transitivité de l'inclusion

$$A \subset B \wedge B \subset C \;\Rightarrow\; A \subset C$$

En effet, une conclusion $a \subset b$ pourra être remplacée par sa définition $\forall X\,(X\in a \Rightarrow X\in b)$ mais pas par $a \subset B \wedge B \subset c$. Outre que ce n'est pas ainsi que l'on traite l'utilisation de conditions suffisantes, l'existence de la variable B, non instanciée, serait actuellement catastrophique.

Or dans la base TPTP, les énoncés sont donnés avec les mot-clefs `axiom`, `hypothesis` ou `conjecture` (pour le théorème à démontrer). Il m'a donc fallu donner mes définitions avec le mot-clef `axiom` et écrire des règles permettant de reconnaître si un "axiome" est une définition ou bien un lemme.

Un cas important où il est nécessaire de savoir qu'un énoncé est une définition est le suivant. MUSCADET est conçu pour travailler avec des propriétés *positives* plutôt que *négatives*. Pour cette raison, j'avais défini la propriété `non_disjoint` plutôt que `disjoint`

`non_disjoint`(A,B) $\Leftrightarrow \exists X\,(X\in A \wedge X\in B)$

et `disjoint`(A,B) $\Leftrightarrow \neg$ `non_disjoint`(A,B)

PUSCADET reçoit maintenant la définition de `disjoint`,

`disjoint`(A,B) $\Leftrightarrow \neg \exists X\,(X\in A \wedge X\in B)$

ce qui est plus conforme au vocabulaire usuel (sinon à l'image mentale usuelle), mais pour chaque définition de la forme p(…) $\Leftrightarrow \neg$ q(…), il crée un nouveau concept de nom `non_p` et de définition `non_p` (…) $\Leftrightarrow$ q(…)

et remplace la définition de p par la définition

p(…) $\Leftrightarrow \neg$ `non_p` (…).

La définition de `non_disjoint` est ainsi générée automatiquement à partir de celle de `disjoint` et on est ainsi ramené à la situation antérieure que MUSCADET savait traiter.

Un autre symbole utilisé par MUSCADET est le symbole "`:`" : "`y:f(x,…)`" signifie que `y` est le nom de `f(x,…)`. Il est introduit par le système dans la mise à plat des formules (par exemple, à partir de la sous-formule `p(g(f(x)))`, il crée les sous-formules `f_x:f(x)`, `g_f_x:g(f_x)` et `p(g_f_x)` où `f_x` et `g_f_x` sont des constantes. Ce symbole apparaît aussi dans les règles de manipulation. Cette transformation ne posait pas de problème, mais ce symbole était aussi utilisé pour la donnée de quelques définitions dans le domaine des applications (composition et inverses), par exemple

```
comp(G,F,A,B,C) : H <=> qqs(X app A, qqs(Z app C,
             ..[H,X] : Z <=> ilexiste(Y app B, ..[F,X] :Y et ..[G,Y] : Z)))
```

Le symbole fonctionnel `".."` ne posait pas de problème car il est du premier ordre et a été traduit par le prédicat `apply` déjà utilisé dans les énoncés CNF. (`apply(F,X,Y)` pour `..[F,X,]:Y`). Par contre, le symbole `":"` doit être compris des démonstrateurs, il ne pouvait donc pas subsister. Je l'ai d'abord remplacé par "=" dans la définition et ai ajouté dans PUSCADET une règle qui le traduisait immédiatement par `":"` dans ce type d'énoncés, ce qui ne changeait donc rien pour PUSCADET. Mais l'énoncé initial devenait contradictoire avec d'autres. Cette contradiction a été détectée par Geoff Sutcliffe testant la cohérence de ma base avec d'autres démonstrateurs (Otter, puis SPASS, plus performant). La raison profonde [1] est due à l'existence d'un énoncé définissant l'ensemble vide ϕ (ou à quelques autres énoncés moins simples). Considérant une application f dont le domaine est l'ensemble vide, que l'on veut composer avec g, on a

```
comp(g,f,ϕ,B,C) = H <=> qqs(X app ϕ, … )
```

le deuxième membre `qqs(X app ϕ, … )` est toujours vrai puisque ϕ n'a pas d'élément, le premier membre l'est donc également, ce qui signifie que n'importe quelle variable H est égale à `comp(g,f,ϕ,B,C)` donc que n'importe quoi est égal à n'importe quoi, ce qui conduit inévitablement à une contradiction.

Après quelques essais, le nouvel énoncé correct est alors le suivant :

```
qqs(X app A, qqs(Z app C, ..[comp(G,F,A,B,C),X]:Z <=>
                 ilexiste(Y app B, ..[F,X]:Y) et ..[G,Y]:Z))))
```

soit, dans la syntaxe TPTP

```
![G,F,A,B,C,X,Z]: ( (member(X,A) & member(Z,C))
 => (apply(compose_function(G,F,A,B,C),X,Z)
                <=> ?[Y]: (    member(Y,B) & apply(F,X,Y)
                    & apply(G,Y,Z)) ) )
```

L'inconvénient de ce nouvel énoncé est qu'il n'est pas reconnu comme définition, il est donc interprété comme lemme, ce qui n'est pas gênant pour lui, mais n'est pas très harmonieux ….

La version finale de cette base de nouveaux théorèmes TPTP a donc été mise au point à la suite de nombreux échanges avec Geoff Sutcliffe. Le plus crucial a été le problème des définitions de la composition et de l'inverse, mais il faisait suite à de nombreux échanges sur le vocabulaire utilisé (ainsi `complement` est devenu `difference`[2], …). D'autres modifications ont été faites directement par Geoff Sutcliffe pour harmoniser avec l'ensemble de la base (présentation des formules, `element` devenu `member`, …)[3], ou pour respecter les contraintes de la base TPTP, plus fortes que celles imposées à PUSCADET (le symbole `comp`, utilisé par PUSCADET à la fois comme prédicat ou comme symbole fonctionnel, qui est utilisé après que l'on ait démontré l'unicité de la composée, a été réécrit soit `compose_predicat` soit `compose_function`)[4]. Il a également rajouté, pour les démonstrateurs qui n'ont pas un traitement "built-in" de l'égalité, non seulement les axiomes standard de l'égalité, mais tous les axiomes de la forme $X = Y \wedge p(X,\ldots) \Rightarrow p(Y,\ldots)$ pour tous les prédicats p utilisés (et analogues pour les symboles fonctionnels). Je tiens à souligner ainsi que Geoff Sutcliffe ne se contente pas de rassembler des problèmes, mais construit la base de la manière la plus harmonieuse possible.

Nous avons eu également des discussions au sujet du symbole utilisé pour l'égalité d'ensembles (`=` ou `equal_set`) et de la définition du produit (annexe 5)[5] .

La nouvelle version 1999 contenait dans le domaine SET et en format FOF les 121 théorèmes que j'avais proposé, mais aussi 195 autres nouveaux théorèmes proposés par deux autres auteurs. Certains ont été démontrés sans problème par PUSCADET, d'autres ont été démontrés après l'ajout de règles, d'autres enfin ont conduits à des créations infinies de nouveaux éléments. Je n'ai pas encore eu le temps de les analyser.

---

[1] Cette raison simple n'a pas été trouvée tout de suite, car les réfutations transmises par Geoff Sutcliffe utilisait d'autres énoncés plus compliqués et aboutissaient à la clause vide après quelques détours …

[2] Les anglo-saxons appellent *différence de A et B* ce que nous appellons *complémentaire de B dans A.*

[3] Ceci a d'ailleurs failli conduire à une catastrophe car , recevant la nouvelle version de la base TPTP, censée être celle de la compétition, je me suis aperçue, après des essais infructueux d'abord incompréhensibles, que la modification d'`element` en `member` n'avait pas été faite pour la dernière version de la définiton de la composition. Cela rendait indémontrable la plupart des théorèmes les plus difficiles … Heureusement, une dernière mise au point a été faite avant la compétition.

[4] Là encore, un oubli de cette nouvelle notation dans un théorème le rendait indémontrable.

[5] Doit-on imposer à l'ensemble dont on prend le produit de ne pas être vide, ce qui est indispendable en théorie axiomatique, mais ne semble pas contradictoire avec les énoncés modélisant la théorie naïve que j'ai donnés. En tout cas, SPASS n'a plus trouvé la définition contradictoire, après que la définition de `comp` ait été corrigée et je pense qu'elle ne l'est pas.

5.4. Mise au point d'un exécutable respectant les contraintes de la compétition

Cela a été un travail beaucoup plus long que prévu, et certainement le plus stressant !

Chaque concurrent doit installer son système sur les machines du site de la compétition, une à deux semaines avant la compétition, puis n'y a plus accès, toutes les manipulations étant faites par l'organisateur. Le système doit être appelé par un nom d'exécutable suivi d'un nom de fichier contenant un problème. Or la pratique habituelle en Prolog est d'appeler Prolog et de lancer une commande Prolog. Cela n'était pas possible.

5.4.1. Premier essai

La première idée a été d'écrire un script shell faisant rentrer dans Prolog et indiquant les commandes à effectuer, selon la syntaxe suivante[1] :

```
pl <<\! > /dev/null 2> /dev/null
['~pastre/puscadet/muscadet.pl'].
tptp('$1').
halt.
!
```

qui recevrait en argument (`$1`) le nom du fichier dans lequel se trouvent les données (`input_formula` de natures `axiom`, `hypothesis` et `conjecture`). `pl` est l'appel à SWI-Prolog, `muscadet.pl` est le fichier contenant toutes les données Prolog (démonstrateur et connaissances), `tptp` est le prédicat Prolog lançant la lecture d'un fichier et la démonstration.

La redirection (`2>`) de la sortie erreur standard vers le fichier Unix poubelle (`/dev/null`) évite que les `warning` ne soient affichés sur la sortie standard.

La difficulté est que le `$` est un caractère spécial en Prolog et que les paramètres d'un script shell doivent s'écrire `$1, $2,` … . Tous les essais de guillemets, quotes, anti-slashes ont été infructueux.

5.4.2. Première solution

La solution utilise trois scripts shell. Le premier script appelé modifie un deuxième script pour en générer un troisième qu'il exécute : `muscadet0` remplace dans `muscadet1` la chaîne `fichier` par le nom donné en argument `$1` dans `muscadet0` et le sauve en `muscadet2`, puis exécute `muscadet2.`

`muscadet0`

```
ed muscadet1 <<% > /dev/null
g/fichier/s//$1/
w muscadet2
q
%
muscadet2
```

`muscadet1`

```
pl <<\! > /dev/null 2> /dev/null
['~pastre/puscadet/muscadet.pl'].
tptp('fichier').
halt.
!
```

`muscadet2`

```
pl <<\! > /dev/null 2> /dev/null
['~pastre/puscadet/muscadet.pl'].
tptp('MGT025+1.p').
halt.
!
```

Ceci manquait d'élégance et surtout, l'inconvénient majeur était de devoir créer un fichier en cours d'exécution, les conditions de la compétition autoriseraient-elles la génération de ce script intermédiaire ?

5.4.3. Deuxième et meilleure solution

La deuxième idée a été d'écrire un programme C appelant Prolog en lui transmettant les paramètres. Divers essais infructueux, après consultation de la documentation SWI-Prolog[2], incompréhensible car ne donnant pratiquement pas d'exemples, m'ont forcée à abandonner.

Enfin, la solution a été d'écrire un petit programme C qui, au moyen des fonctions C `strcpy` et `strcat`[3], construit puis appelle le script sous forme d'une unique chaîne de caractères comportant le nom du fichier donné en entrée. `cmd` pointe vers la chaîne construite en plusieurs étapes, `tmp` vers l'argument, `\n` est le passage à la ligne. C'est tout à fait illisible, mais ça marche.

---

[1] Merci à Yannick Parchemal qui m'a indiqué cette syntaxe.

[2] Merci à Bruno Bouzy pour l'aide dans ces essais.

[3] Merci encore Bruno Bouzy qui m'a indiqué cette possibilité.

```
main(argc,argv)
int argc;
char *argv[];
{  char tmp[256];
   char tmp1[256];
   char cmd[256];
   strcpy(tmp, argv[1]);
   strcpy(cmd,"pl -f muscadet.pl <<! 2> /dev/null \n tptp('");
   strcat(cmd,tmp);
   strcpy(tmp1, "'). \n halt. \n ! ");
   strcat(cmd,tmp1);
   system(cmd);
}
```

L'option `-f` de Prolog charge le fichier `muscadet.pl` à l'appel de Prolog.

Mes efforts n'étaient pas encore terminés. Je voulais que le système n'écrive que le résultat demandé, et non la bannière Prolog, les messages de compilation, etc. C'est ce que fait l'option `-g true`[1]. Soit finalement, pour le début de la chaîne :

```
   strcpy(cmd,"pl -f muscadet.pl -g true <<! 2> /dev/null \n tptp('");
```

Il ne restait plus qu'à installer le système sur les machines du site de la compétition, rectifier quelques détails dus au changement de machine, et espérer que le système passe tous les tests … . En fait, seuls quelques cas de plantage à la lecture des données sont restés inexpliqués, sans doute dus au fait que la version de SWI-Prolog du site de la compétition n'était pas la même que la mienne.

6. La compétition 1999 (CASC-16)

Elle eut lieu à Trento (Italie) en juillet 1999. Il y avait 17 concurrents dont 6 dans la division FOF.

6.1. Choix des problèmes

Les problèmes proposés pour la compétition sont choisis parmi les problèmes les plus difficiles de la base. Leurs numéros et les noms de symboles sont remplacés par des noms sans signification, l'ordre des formules et sous-formules est aléatoirement modifié.

Les vainqueurs sont censés rendre leur source public.

Le but des organisateurs, et ils font pour cela un gros travail, est de permettre une évaluation la plus objective possible des démonstrateurs (tout en rappelant régulièrement à ceux qui le prennent trop au sérieux que "it is for fun").

La difficulté de tous les problèmes de la base est évaluée en fonction des performances des participants (chacun a dû donner, quelques semaines avant, les performances de son système sur la totalité de la base, avec les temps d'exécution). Certaines considérations sont prises en compte, comme le fait que des problèmes se ressemblent ou que certains problèmes semblent faits pour certains démonstrateurs (par exemple si un système démontre un théorème qu'aucun autre ne démontre et ne démontre pas de nombreux autres théorèmes).

Un ensemble de problèmes *éligibles* est alors constitué, et le dernier choix se fait aléatoirement le matin de la compétition.

Un jury, composé de trois chercheurs, A.Bundy, C.Kirchner et F.Plenning, non impliqués dans la compétition, en contrôle le bon déroulement, avant, pendant et après.

6.2. Public ou secret ?

Un message de Geoff Sutcliffe, un mois avant la compétition, demandant à tous les inscrits leur avis sur une mise à jour des critères d'évaluation a provoqué, parmi les habitués de cette compétition, des échanges dont certains ont été assez violents, un vainqueur de 1998, ayant même été accusé de tricherie car ayant refusé de rendre public la totalité de son système. La vraie question est de savoir si un système a le droit de tourner pendant toute l'année sur la base de manière à faire des ajustements de certains paramètres, au moyen de tables dans le cas cité, et de garder ces tables secrètes. Le débat a ensuite porté sur la question de savoir si la compétition doit permettre de dégager le "meilleur", comme une course de Formule 1 (mais dans ce cas, est-il légitime d'utiliser des deniers publics ?) ou doit permettre de faire progresser la démonstration

---

[1] Merci à Miguel Pintado pour sa connaissance infinie des profondeurs de Prolog inconnues de tous les autres utilisateurs de Prolog consultés.

automatique (et dans ce cas, chacun doit pouvoir profiter du travail des autres). La plupart de auteurs ont d'ailleurs plusieurs versions de leur système, la version CASC étant une version particulière.

Le ton s'est calmé durant la conférence. En décrivant son système, le vainqueur précité a parlé d'apprentissage (learning) dans la mise au point de son système qu'un de ses opposants a qualifié d'ajustement (tuning).

Epilogue : immédiatement après la compétition, Geoff Sutcliffe a envoyé un message disant qu'il avait l'intention d'installer sur le Web les sources et les exécutables de tous les participants, et que si quelqu'un y voyait une objection qu'il le fasse savoir (pourquoi). Il n'y eut pas d'objection.

6.3. A propos des tests de validité

Quelques mois après la compétition, un des vainqueurs (qui ne concourait pas dans la division FOF) a été disqualifié car son système a été trouvé invalide au cours des tests post-compétition sur la totalité de la base. Il s'en est suivi un échange de courriers assez vifs, ses défenseurs argumentant qu'il n'avait pas été trouvé invalide sur les problèmes de la compétition, donc que cela n'avait pas influencé les résultats et qu'il n'avait pas à être disqualifié. Ils avancent également que même les logiciels professionnels ont des bogues et que personne ne peut garantir que son logiciel n'en a pas. Le jury a donné raison aux organisateurs et a confirmé la disqualification, considérant qu'il n'était pas sérieux de déclarer vainqueur un système non-valide ! D'abord, les problèmes de la compétition sont représentatifs d'un ensemble plus large de problèmes, sinon elle n'aurait pas d'intérêt pour la communauté ATP (Automated Theorem Proving). Ensuite, on ne peut pas avoir confiance sur la validité des démonstrations d'un système invalide, même sur les problèmes de la compétition[1]. Enfin, à titre de signal pour une communauté plus large, il est important de montrer que les organisateurs et le jury sont intransigeants, considèrent que le problème de la non-validité est sérieux et font tout ce qu'ils peuvent pour le réduire.

6.4. Les résultats de la CASC-16

Les résultats complets peuvent être consultés aux adresses
http://www.cs.jcu.edu.au/~tptp/CASC-16/WWWResults/Results.html
et http://www.cs.jcu.edu.au/~tptp/CASC-16/WWWResults/ResultsSummary.html
Les résultats pour la division FOF (FEQ et FNE) sont en Fig. 15, 16 et 17.
SPASS est de nouveau le meilleur globalement pour la division FOF. MUSCADET a été le meilleur dans la catégorie FEQ, mais le dernier (nul !) dans la catégorie FNE.

FOF Problems

	SPASS 1.00T	E-SETHEO 99csp	SPASS 0.95T	Bliksem 1.10	MUSCADET 2.1	OtterMACE 437
Attempted	30	30	30	30	30	30
Solved	22	22	19	12	9	9
Av. Time	25.46	32.30	28.36	1.58	1.48	7.54

Fig. 15. *Résultats dans la division FOF*

Dans la catégorie FEQ, MUSCADET a démontré 9 théorèmes sur 15 et en démontre maintenant 2 autres.

---

[1] En effet, le jour de la compétition, les systèmes sont évalués uniquement sur leur réponse : l'énoncé est, ou n'est pas, un théorème

FEQ Problems

	MUSCADET 1.00T	**SPASS 99csp**	**E-SETHEO 1.10**	**Bliksem 0.95T**	**SPASS 437**	**OtterMACE 2.1**
MGT021+1	0.70	0.10	9.40	0.00	0.00	0.30
SET055+1	unknown	timeout	3.10	timeout	timeout	0.70
SET065+1	unknown	0.30	3.00	0.10	0.10	0.70
SET071+1	1.50	3.40	40.80	timeout	timeout	timeout
SET084+1	unknown	0.40	114.30	0.70	9.40	65.40
SET094+1	unknown	1.10	18.90	0.20	1.10	timeout
SET352+4	1.00	0.40	timeout	timeout	timeout	timeout
SET697+4	no_soln	0.30	timeout	timeout	timeout	timeout
SET723+4	1.60	timeout	timeout	timeout	timeout	timeout
SET726+4	1.80	timeout	timeout	timeout	timeout	timeout
SET746+4	1.70	timeout	timeout	timeout	timeout	timeout
SET752+4	1.60	timeout	timeout	timeout	timeout	timeout
SET757+4	1.90	timeout	timeout	timeout	timeout	timeout
SET769+4	1.50	timeout	timeout	timeout	timeout	timeout
SYN072+1	no_soln	0.10	12.90	0.00	0.00	unknown
Attempted	**15**	**15**	**15**	**15**	**15**	**15**
Solved	**9**	**8**	**7**	**5**	**5**	**4**
Av. Time	**1.48**	**0.76**	**28.91**	**0.20**	**2.12**	**16.78**

Fig. 16. *Résultats dans la catégorie FEQ de la division FOF*

Dans la catégorie FNE, MUSCADET n'a démontré aucun des 15 théorèmes. Depuis,

- 3 autres théorèmes ont été démontrés après ajout de quelques transformations ou stratégies simples dans la base de connaissances de MUSCADET. (Elles n'y figuraient pas car correspondent à des façons de s'exprimer que l'on ne rencontre pas en mathématiques et je n'avais pas eu le temps d'étudier la base SYN dans sa totalité).
- un théorème est démontré mais uniquement par la variante consistant à traiter les hypothèses disjonctives avant les hypothèse existentielles.
- 3 théorèmes présentent des difficultés qui ne semblent pas insurmontables et devront être étudiés.
- enfin les 8 autres ont des énoncés de plus de 10 pages et n'ont pas été examinés en détail. MUSCADET n'est certainement pas adapté à ce genre de problèmes (ALC, voir section 5.2.2) !

MUSCADET démontre donc aujourd'hui 14 des 30 théorèmes (11+3). Les autres concurrents peuvent aussi avoir amélioré leur performance depuis.[1]

---

[1] Citons par exemple une anecdote. Comme les organisateurs s'étonnaient qu'un démonstrateur, dans une autre division, ayant de très bons résultats les années précédentes, n'avait que des résultats médiocres, son auteur a indiqué qu'il s'était trompé de version en installant son système …

FNE Problems

	E-SETHEO 99csp	**SPASS 0.95T**	**SPASS 1.00T**	**Bliksem 1.10**	**OtterMACE 437**	**MUSCADET 2.1**
SYN036+1	8.80	0.00	0.30	18.00	timeout	unknown
SYN052+1	1.40	0.00	0.10	0.00	0.20	no_soln
SYN332+1	8.90	0.00	0.10	0.00	unknown	unknown
SYN362+1	1.40	0.00	0.10	0.00	0.20	unknown
SYN378+1	1.40	0.00	0.10	0.00	0.10	no_soln
SYN381+1	1.20	0.00	0.10	0.00	0.10	unknown
SYN411+1	1.50	0.00	0.10	0.00	0.20	unknown
SYN436+1	19.70	91.20	90.20	timeout	timeout	timeout
SYN439+1	136.20	timeout	timeout	timeout	timeout	timeout
SYN448+1	27.60	37.40	37.20	timeout	timeout	timeout
SYN466+1	121.40	99.10	131.20	timeout	timeout	timeout
SYN469+1	35.40	69.90	70.60	timeout	timeout	timeout
SYN484+1	56.50	65.20	64.80	timeout	timeout	timeout
SYN487+1	30.20	114.10	106.60	timeout	timeout	timeout
SYN508+1	56.60	51.40	52.50	timeout	timeout	timeout
Attempted	**15**	**15**	**15**	**15**	**15**	**15**
Solved	**15**	**14**	**14**	**7**	**5**	**0**
Av. Time	**33.88**	**37.74**	**39.57**	**2.57**	**0.16**	**-**

Fig. 17. *Résultats dans la catégorie FNE de la division FOF*

6.5. Les problèmes de la division FOF de la CASC-16

6.5.1. catégorie FEQ

SET 13 problèmes, dont :

- 8 proposés par moi-même, MUSCADET a échoué pour 1 (pour une raison inconnue, il réussit à Paris 5), SPASS en a démontré 2, les autres démonstrateurs aucun
  - *2 sur les opérations ensemblistes élémentaires :*
    - $\sigma(\{A,B\}) = A \cup B$ (SET352+4)
    - $A \subset B$ <=> $A \cap C(B,E) = \phi$ (SET697+4, non démontré par MUSCADET le jour de la compétition)
  - *3 sur les applications :*
    - si f,g,h sont des applications de A dans A, et si f est injective, alors $f_o g = f_o h$ => $g=h$ (SET723+4)
    - si f est une application de A dans B, g et h deux applications de B dans A telle que $g_o f = id_A$ et $f_o h = id_B$, alors $g = f^{-1}$ (SET726+4)
    - soient f et g deux applications resp. de A dans B et de B dans C, si f et g sont croissantes alors $g_o f$ est croissante (SET746+4)
  - *2 sur les ensembles images/images réciproques :*
    - $f(X \cup Y) = f(X) \cup f(Y)$ (SET752+4)
    - $f^{-1}(X \cap Y) = f^{-1}(X) \cap f^{-1}(Y)$ (SET757+4)
  - *1 sur les relations d'équivalence :*
    - si on a une relation d'équivalence sur E, alors deux classes d'équivalence sont égales si et seulement si elles ne sont pas disjointes (SET769+4)
- 5 issus de Quaife (1992) (SET...+1.p). PUSCADET en a démontré un.

Ces théorèmes sont basés sur une axiomatique qui devra être examinée en détails.

MGT 1 problème, démontré par tous les démonstrateurs (MGT021+1, voir annexe 3).

SYN 1 problème, non démontré le jour de la compétition, démontré depuis.

6.5.2. catégorie FNE

MUSCADET n'a démontré aucun des théorèmes de cette catégorie qui étaient tous des problèmes du domaine SYN.

8 théorèmes (ALC, voir section 5.2.2) ont des énoncés de plus de 10 pages … . Seuls E et SPASS ont réussi.

Les 7 autres sont les suivants :

SYN411+1 $\forall X\ \forall Y\ \forall Z\ p(X,Y,Z) \Leftrightarrow \neg\ \exists U\ \exists V\ \exists W\ \neg\ p(U,V,W)$

théorème trivial de logique pure que tous les démonstrateurs ont démontré sauf MUSCADET car il ne savait pas réécrire une hypothèse $\neg\ \exists U\ F$ en $\forall U\ \neg\ F$. Il savait traiter $\neg\ \exists U\ F$ en tant que conclusion et savait construire des règles à partir d'hypothèses universelles et pouvait ainsi démontrer l'implication dans le premier sens (Fig. 18).

Pour que MUSCADET démontre l'implication dans l'autre sens, d'une manière analogue, il a suffi d'ajouter la prise en compte des hypothèses de la forme $\neg\ \exists U\ F$ et de la forme $\neg\neg F$ pour la construction des règles à partir des hypothèses.

Les réécritures $\neg\ \exists U\ F$ en $\forall U\ \neg\ F$ et $\neg\neg F$ en F étant de celles effectuées pour mettre sous forme de clause, ce théorème est trivial pour le Principe de Résolution. On pourrait envisager de donner aussi directement ces réécritures pour démontrer immédiatement ce genre de théorèmes. Les méthodes employées sont plus générales.

objets	hypothèses	règles	conclusion
			$\forall X\forall Y\forall Z p(X,Y,Z)\Leftrightarrow\neg\exists U\exists V\exists W\neg p(U,V,W)$
	$\forall X\ \forall Y\ \forall Z\ p(X,Y,Z)$		$\neg\ \exists U\ \exists V\ \exists W\ \neg\ p(U,V,W)$
		r1 : si objets X,Y,Z alors p(X,Y,Z)	
	$\exists U\ \exists V\ \exists W\ \neg\ p(U,V,W)$		faux
u, v, w	$\neg\ p(u,v,w)$		
	p(u,v,w) *(règle r1)*		
	faux		*démontré*

Fig. 18. *Démonstration du premier sous-théorème du théorème SYN411+1*

SYN036+1 $[\exists X\ \forall Y\ (p(X) \Leftrightarrow p(Y)) \Leftrightarrow (\exists U\ q(U) \Leftrightarrow \forall W\ q(W))] \Leftrightarrow$
$[\exists X1\ \forall Y1\ (q(X1) \Leftrightarrow q(Y1)) \Leftrightarrow (\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1))]$

Cet énoncé est assez artificiel. Sa sémantique n'est pas évidente et une fois comprise, il est artificiellement compliqué. En effet les énoncés $\exists X\ \forall Y(p(X)\Leftrightarrow p(Y))$ et $\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1)$ ont la même interprétation : "le prédicat *p* prend toujours la même valeur". Une fois cette remarque faite, il suffit, en désignant par $a(p)$ la propriété $\exists X\ \forall Y\ (p(X) \Leftrightarrow p(Y))$ et par $b(p)$ la propriété $\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1)$ de démontrer que $\forall P(a(P)\Leftrightarrow b(P)) \Rightarrow \forall P\ \forall Q([a(P)\Leftrightarrow b(Q)] \Leftrightarrow [a(Q)\Leftrightarrow a(P)]$ (qui est un énoncé du deuxième ordre, si l'on se souvient que P et Q sont des propriétés).

PUSCADET n'est cependant pas capable de démontrer que

$$\exists X\ \forall Y\ [p(X) \Leftrightarrow p(Y)] \ \Leftrightarrow\ [\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1)]$$

mais démontre le théorème (facile)

$$\forall P\ [a(P) \Leftrightarrow b(P)] \ \Rightarrow\ \forall P\ \forall Q\ ([a(P)\Leftrightarrow b(Q)]\Leftrightarrow[a(Q) \Leftrightarrow a(P)])$$

SYN378+1 $\forall X\ p(X) \Rightarrow (\neg\ \exists Y\ q(Y) \vee \exists Z\ [p(Z)\Rightarrow q(Z)])$

PUSCADET ajoute l'hypothèse $\exists Y\ q(Y)$ et cherche à démontrer $\exists Z\ [p(Z)\Rightarrow q(Z)]$. Il crée un objet *y* tel que $q(y)$. Le jour de la compétition, il ne pouvait plus rien faire. Il démontre maintenant ce théorème après que le traitement des conclusions existentielles simples ait été complété.

SYN362+1 $[\forall Y\ \exists W\ r(Y,W) \wedge \exists Z\ \forall X\ (p(X) \Rightarrow \neg\ r(Z,X))] \ \Rightarrow\ \exists X\ \neg\ p(X)$

est également maintenant démontré par MUSCADET (traitement de la conclusion existentielle par l'absurde : on ajoute l'hypothèse universelle$\forall X\ p(X)$

SYN052+1 $\forall X\ [p \Leftrightarrow f(X)] \ \Rightarrow\ [p \Leftrightarrow \forall X1\ f(X1)]$

PUSCADET sait démontrer dans un sens $p \Rightarrow \forall X1\ f(X1)$ mais pas dans l'autre $\forall X1\ f(X1) \Rightarrow p$ car aucun objet n'est créé.

SYN381+1 $\forall X\ [\exists Y\ q(X,Y) \Rightarrow p(X)]$
$\wedge\ \forall V\ \exists U\ q(U,V)$
$\wedge\ \forall W\ \forall Z\ [q(W,Z) \Rightarrow (Z,W) \vee q(Z,Z)]$
$\Rightarrow \forall Z\ p(Z)$

Ce théorème est démontré avec la variante consistant à traiter les hypothèses disjonctives avant les hypothèses existentielles (de même que le théorème SET044+1 cité en 5.2.1). Sinon, une chaîne infinie de nouveaux objets est créée.

SYN332+1 $\exists X\ \exists Y\ \forall Z \Big(\ [f(X,Y) \wedge f(Y,X) \Leftrightarrow f(X,Z)]$
$\Rightarrow \Big[[f(X,Z) \Leftrightarrow f(Z,X)]$
$\Rightarrow \Big(\ [f(X,Z) \Leftrightarrow f(Y,Z)]$
$\Rightarrow \Big[\big([f(Y,X) \Rightarrow f(X,Y)] \Leftrightarrow f(Z,Z)\big)$
$\Rightarrow \big(\ [f(X,Y) \Leftrightarrow f(Y,X)] \Leftrightarrow f(Z,Y)]\big)\Big]\Big]\Big)$

PUSCADET n'est pas actuellement capable de démontrer ce théorème. J'ai moi-même mis assez longtemps pour le démontrer à la main. On trouvera en annexe 6 plusieurs démonstrations et le monitoring de leur recherche. On remarquera l'utilité de combiner plusieurs approches.

7. Conclusion

Les nombreuses améliorations du système MUSCADET, incluant sa réécriture complète en Prolog l'ont rendu plus performant. Il a ainsi pu démontrer des théorèmes énoncés dans un style différent de ceux sur lesquels les expérimentations avaient été auparavant faites, dans les domaines de l' économie, la logique pure, et les paradoxes de la théorie des ensembles.

MUSCADET a obtenu à la compétition 1999 organisée dans le cadre de la conférence CADE des résultats qui ont montré la supériorité des méthodes utilisées sur le Principe de résolution pour certains types de problèmes. En effet, il a eu les meilleurs résultats dans la catégorie "Premier ordre avec égalité". Il a au contraire eu de très mauvais résultats (bien qu'améliorés depuis) dans la catégorie "Premier ordre sans égalité".

La différence des résultats ne provient pas uniquement de la présence ou non de l'égalité mais il se trouve que les théorèmes comportant les difficultés pour lesquelles MUSCADET est efficace comportent aussi l'égalité. Ce sont des théorèmes manipulant de nombreux concepts définis par d'autres énoncés à partir desquels MUSCADET construit des règles, par opposition à des énoncés uniques pouvant être énormes et/ou dénués de sémantique. Ce sont aussi des théorèmes traitant de propriétés susceptibles de créer de nouveaux objets ou éléments. Le traitement des propriétés existentielles incorporé dans MUSCADET lui permet de procéder à des créations dans toutes les directions nécessaires, sans toutefois s'enfermer dans des chaînes infinies d'objets créés.

Ces résultats montrent également, si cela était encore nécessaire, l'intérêt de combiner plusieurs types de méthodes. Il est en effet possible d'analyser les familles d'énoncés pour lesquels un ou un autre type de méthode sera le plus efficace.

8. REFERENCES

[Bazin 1993] Bazin, J.M. : *GEOMUS: un résolveur de problèmes de géométrie qui mobilise ses connaissances en fonction du problème posé,* thèse université Paris 6, 1993.

[Bledsoe 1971] Bledsoe, W.W. : *Splitting and reduction heuristics in automatic theorem proving,* Journal of Artificial Intelligence 2, 55-77, 1971.

[Bledsoe 1977] Bledsoe, W.W. : *Non-resolution theorem proving,* Journal of Artificial Intelligence 9, 1-35, 1977.

[Bledsoe 1978] Bledsoe, W.W., Tyson, M. : *The UT interactive prover*, University of Texas, math. dept Memo ATP 17A, 1978.

[Lenat 1982] Lenat, D.B. : *AM : discovery in mathematics as heuristic search*, in Davis, R., Lenat, D.B, Knowledge-based systems in artificial intelligence, Mc Graw Hill, 1982.

[Pastre 1978] Pastre, D. : *Automatic Theorem Proving in Set Theory*, Journal of Artificial Intelligence 10, 1-27, 1978.

[Pastre 1989] Pastre, D. : *MUSCADET: an automatic theorem proving system using knowledge and metaknowledge in mathematics*, Journal of Artificial Intelligence 38, 257-318, 1989.

[Pastre 1993] Pastre, D. : *Automated Theorem Proving in Mathematics*, Annals of Mathematics and Artificial Intelligence, Volume 8, No. 3-4, 425-447, 1993.

[Pitrat 1990] Pitrat, J. : *Métaconnaissances, futur de l'intelligence artificielle*, Hermès, 1990.

[Robinson 1965] Robinson, J.A. : *A machine oriented logic based on the resolution principle*, J.ACM 12, 23-41, 1965.

[Spagnol 1999] Spagnol, J.P. : *Automatisation du raisonnement et de la rédaction des preuves en géométrie des configurations*, dans cette publication, 1999.

[Wang 1960] Wang H, : *Towards mechanical mathematics*, IBM J. Res. Dev. 4, 2-22, 1960.

[Wos 1988] Wos, L. : *Automated Reasoning : 33 Basic Research Problems*, Prentice Hall, 1988.

```
%--------------------------------------------------------------------------
% File     : SYN040+1 : TPTP v2.2.0. Released v2.0.0.
% Domain   : Syntactic
% Problem  : Pelletier Problem 1
% Version  : Especial.
% English  : A biconditional version of the 'most difficult' theorem
%            proved by the original Logic Theorist.
% Refs     : [NSS63] Newell et al. (1963), Empirical Explorations with the
%          : [Pel86] Pelletier (1986), Seventy-five Problems for Testing Au
%          : [Hah94] Haehnle (1994), Email to G. Sutcliffe
% Source   : [Hah94]
% Names    : Pelletier 1 [Pel86]
% Status   : theorem
% Rating   : 0.00 v2.1.0
% Syntax   : Number of formulae    :    1 (   0 unit)
%            Number of atoms       :    4 (   0 equality)
%            Maximal formula depth :    4 (   4 average)
%            Number of connectives :    5 (   2 ~  ;   0  |;   0  &)
%                                         (   1 <=>;   2 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :    2 (   2 propositional; 0-0 arity)
%            Number of functors    :    0 (   0 constant; --- arity)
%            Number of variables   :    0 (   0 singleton;   0 !;   0 ?)
%            Maximal term depth    :    0 (   0 average)
% Comments : [NSS63] first appeared in 1957, as cited in [Pel86]. The 1963
%            version is a reprint.
%--------------------------------------------------------------------------
input_formula(pel1,conjecture,(
    ( p
   => q )
<=> ( ~ q
   => ~ p )   )).
%--------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% File     : SYN001+1 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(pel2,conjecture,(
    ~ ~ p
<=> p   )).
%--------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% File     : SYN007+1.005 : TPTP v2.2.0. Released v2.0.0.
% Domain   : Syntactic
% Problem  : Pelletier Problem 71
% Version  : Especial.
%            Theorem formulation : For N = SIZE.
% English  : Clausal forms of statements of the form :
%            (p1 <-> (p2 <->...(pN <-> (p1 <-> (p2 <->...<-> pN)...)
% Refs     : [Pel86] Pelletier (1986), Seventy-five Problems for Testing Au
%          : [Urq87] Urquart (1987), Hard Problems for Resolution
% Source   : [Pel86]
% Names    : Pelletier 71 [Pel86]
% Status   : theorem
% Rating   : ? v2.0.0
% Syntax   : Number of formulae    :    1 (   0 unit)
%            Number of atoms       :   10 (   0 equality)
%            Maximal formula depth :   10 (  10 average)
%            Number of connectives :    9 (   0 ~  ;   0  |;   0  &)
%                                         (   9 <=>;   0 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :    5 (   5 propositional; 0-0 arity)
%            Number of functors    :    0 (   0 constant; --- arity)
%            Number of variables   :    0 (   0 singleton;   0 !;   0 ?)
%            Maximal term depth    :    0 (   0 average)
```

```
% Comments : The number of distinct letters in U-N is N. The number of
%            occurrences of sentence letters in 2N. The number of clauses
%            goes up dramatically as N increases, but I don't think it
%            shows that the problems are dramatically more difficult as N
%            increases. Rather, it's that the awkward clause form
%            representation comes to the fore most dramatically with
%            embedded biconditionals. On all other measures of complexity,
%            one should say that the problems increase linearly in
%            difficulty. Urquhart says that the proof size of any resolution
%            system increases exponentially with increase in N.
%          : This problem can also be done in terms of graphs, as described
%            in [Pel86] Problem 74.
%          : tptp2X: -ftptp -s5 SYN007+1.g
%--------------------------------------------------------------------------
input_formula(prove_this,conjecture,(
    p_1
<=> ( p_2
  <=> ( p_3
    <=> ( p_4
      <=> ( p_5
        <=> ( p_1
          <=> ( p_2
            <=> ( p_3
              <=> ( p_4
                <=> p_5 ) ) ) ) ) ) ) )   )).
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% File     : SYN007+1.014 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(prove_this,conjecture,(
    p_1
<=> ( p_2
  <=> ( p_3
    <=> ( p_4
      <=> ( p_5
        <=> ( p_6
          <=> ( p_7
            <=> ( p_8
              <=> ( p_9
                <=> ( p_10
                  <=> ( p_11
                    <=> ( p_12
                      <=> ( p_13
                        <=> ( p_14
                          <=> ( p_1
                            <=> ( p_2
                              <=> ( p_3
                                <=> ( p_4
                                  <=> ( p_5
                                    <=> ( p_6
                                      <=> ( p_7
                                        <=> ( p_8
                                          <=> ( p_9
                                            <=> ( p_10
                                              <=> ( p_11
                                                <=> ( p_12
                                                  <=> ( p_13
                                                    <=> p_14 ) ) ) ) ) ) ) ) ) ) ) )
) ) ) ) ) ) ) ) ) ) ) ) ) ) )   )).
%--------------------------------------------------------------------------



%--------------------------------------------------------------------------
% File     : SYN041+1 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(pel3,conjecture,(
    ~ ( p
     => q )
```

```
 => ( q
   => p )   )).
%--------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% File     : SYN062+1 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(pel32_1,axiom,(
    ! [X] :
      ( ( big_f(X)
        & ( big_g(X)
          | big_h(X) ) )
     => big_i(X) )   )).

input_formula(pel32_2,axiom,(
    ! [X] :
      ( ( big_i(X)
        & big_h(X) )
     => big_j(X) )   )).

input_formula(pel32_3,axiom,(
    ! [X] :
      ( big_k(X)
     => big_h(X) )   )).

input_formula(pel32,conjecture,(
    ! [X] :
      ( ( big_f(X)
        & big_k(X) )
     => big_j(X) )   )).
%--------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% File     : SYN056+1 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(pel26_1,axiom,(
    ? [X] : big_p(X)
<=> ? [Y] : big_q(Y)   )).

input_formula(pel26_2,axiom,(
    ! [X,Y] :
      ( ( big_p(X)
        & big_q(Y) )
     => ( big_r(X)
      <=> big_s(Y) ) )   )).

input_formula(pel26,conjecture,(
    ! [X] :
      ( big_p(X)
     => big_r(X) )
<=> ! [Y] :
      ( big_q(Y)
     => big_s(Y) )   )).
%--------------------------------------------------------------------------
```

res_SYN040+1.p

```
pel1
****************************************************
theoreme a demontrer
(p => q) <=> (non q => non p)
0 nouvelle conclusion (p => q) <=> (non q => non p)

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2 seul
.. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ reflexivity_ et defconcl1 defconcl1a defconcl1b
defconcl1bb ilexiste ou defconcl2 defconcl2app defconcl2elt defconcl2a defconcl2b
defconcl3

0 nouvelle conclusion ((p => q) => (non q => non p)) et ((non q => non p) => (p =>
q))
---------------------------------------------------------- regle <=>

************************************************ sous - theoreme 1 *****
1 nouvelle conclusion (p => q) => (non q => non p)
----------------------------------------------- creation du sous - theoreme 1
1 ajouter la regle active locale

- (regle(A, reghyp) :- hyp(A, p), not hyp(A, q), ajhyp(A, q)).

a la fin des regles universelles

1 nouvelle conclusion non q => non p
---------------------------------------------------------- regle =>
1 ajouter hypothese non q
1 nouvelle conclusion non p
---------------------------------------------------------- regle =>
1 ajouter hypothese p
1 nouvelle conclusion faux
---------------------------------------------------------- regle non
supprimer hypothese non q
1 nouvelle conclusion q
---------------------------------------------------------- regle hypnon
1 ajouter hypothese q
---------------------------------------------------------- regle reghyp
1 nouvelle conclusion vrai
---------------------------------------------------------- regle stop1
theoreme 1 demontre
===================
0 nouvelle conclusion (non q => non p) => (p => q)

************************************************ sous - theoreme 2 *****
2 nouvelle conclusion (non q => non p) => (p => q)
----------------------------------------------- creation du sous - theoreme 2
2 ajouter la regle active locale

- (regle(A, reghyp1) :- hyp(A, non q), hyp(A, p), not hyp(A, faux), ajhyp(A, faux)).

a la fin des regles universelles

2 ajouter la regle active locale

- (regle(A, reghyp2) :- not hyp(A, q ou non p), ajhyp(A, q ou non p)).

a la fin des regles universelles

2 nouvelle conclusion p => q
---------------------------------------------------------- regle =>
2 ajouter hypothese p
2 nouvelle conclusion q
---------------------------------------------------------- regle =>
2 ajouter hypothese q ou non p
---------------------------------------------------------- regle reghyp2
```

```
2 traitement de l hypothese disjonctive q ou non p
2 nouvelle conclusion (q => q) et (non p => q)
2 ajouter hypothese-traitee q ou non p
--------------------------------------------------------- regle ou

************************************************ sous - theoreme 21 *****
21 nouvelle conclusion q => q
----------------------------------------------- creation du sous - theoreme 21
21 ajouter hypothese q
21 nouvelle conclusion q
--------------------------------------------------------- regle =>
21 nouvelle conclusion vrai
--------------------------------------------------------- regle stop1
theoreme 21 demontre
===================
2 nouvelle conclusion non p => q

************************************************ sous - theoreme 22 *****
22 nouvelle conclusion non p => q
----------------------------------------------- creation du sous - theoreme 22
22 ajouter hypothese non p
22 nouvelle conclusion q
--------------------------------------------------------- regle =>
22 nouvelle conclusion vrai
--------------------------------------------------------- regle stop4
theoreme 22 demontre
===================
2 nouvelle conclusion vrai
--------------------------------------------------------- regle et
theoreme 2 demontre
===================
0 nouvelle conclusion vrai
--------------------------------------------------------- regle et
theoreme 0 demontre
===================
```

---

res_SYN001+1.p

```
pel2
****************************************************
theoreme a demontrer
non non p <=> p
0 nouvelle conclusion non non p <=> p

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2 seul
.. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ reflexivity_ et defconcl1 defconcl1a defconcl1b
defconcl1bb ilexiste ou defconcl2 defconcl2app defconcl2elt defconcl2a defconcl2b
defconcl3

0 nouvelle conclusion (non non p => p) et (p => non non p)
--------------------------------------------------------- regle <=>

************************************************ sous - theoreme 1 *****
1 nouvelle conclusion non non p => p
----------------------------------------------- creation du sous - theoreme 1
1 ajouter hypothese non non p
1 nouvelle conclusion p
--------------------------------------------------------- regle =>
1 ajouter hypothese p
--------------------------------------------------------- regle hypnonnon
1 nouvelle conclusion vrai
--------------------------------------------------------- regle stop1
theoreme 1 demontre
===================
0 nouvelle conclusion p => non non p

************************************************ sous - theoreme 2 *****
2 nouvelle conclusion p => non non p
----------------------------------------------- creation du sous - theoreme 2
```

```
2 ajouter hypothese p
2 nouvelle conclusion non non p
--------------------------------------------------------- regle =>
2 ajouter hypothese non p
2 nouvelle conclusion faux
--------------------------------------------------------- regle non
2 nouvelle conclusion vrai
--------------------------------------------------------- regle stop4
theoreme 2 demontre
===================
0 nouvelle conclusion vrai
--------------------------------------------------------- regle et
theoreme 0 demontre
===================
```

res_SYN041+1.p

```
pel3
****************************************************
theoreme a demontrer
non (p => q) => (q => p)
0 nouvelle conclusion non (p => q) => (q => p)

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2 seul
.. .. 1 ilec1 ilec2 ilec3 ilec4 ilec5 ilec1a ilec1b et_trivial_1 et_trivial_2
stop_trivial stop_trivial_ou conclet bidon_ et defconcl1 defconcl1a defconcl1b
defconcl1bb ilexiste ou defconcl2 defconcl2app defconcl2elt defconcl2a defconcl2b
defconcl3

0 ajouter hypothese non (p => q)
0 nouvelle conclusion q => p
--------------------------------------------------------- regle =>
0 ajouter hypothese p
0 ajouter hypothese non q
--------------------------------------------------------- regle hypnonimp
0 ajouter hypothese q
0 nouvelle conclusion p
--------------------------------------------------------- regle =>
0 nouvelle conclusion vrai
--------------------------------------------------------- regle stop1
theoreme 0 demontre ( pel3 ) en 1.233333 seconds
===================
```

res_SYN062+1.p

```
pel32
****************************************************
theoreme a demontrer

- qqs(A, big_f(A) et big_k(A) => big_j(A)).

0 nouvelle conclusion
- qqs(A, big_f(A) et big_k(A) => big_j(A)).

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2 seul
.. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ pel32_1_ pel32_1_1 pel32_2_ pel32_3_ reflexivity_ et
defconcl1 defconcl1a defconcl1b defconcl1bb ilexiste ou defconcl2 defconcl2app
defconcl2elt defconcl2a defconcl2b defconcl3

0 ajouter objet x
0 nouvelle conclusion big_f(x) et big_k(x) => big_j(x)
--------------------------------------------------------- regle qqs1
0 ajouter hypothese big_f(x)
0 ajouter hypothese big_k(x)
0 nouvelle conclusion big_j(x)
--------------------------------------------------------- regle =>
0 ajouter hypothese big_h(x)
```

```
--------------------------------------------------------- regle pel32_3_
0 ajouter hypothese big_i(x)
--------------------------------------------------------- regle pel32_1_1
0 ajouter hypothese big_j(x)
--------------------------------------------------------- regle pel32_2_
0 nouvelle conclusion vrai
--------------------------------------------------------- regle stop1
theoreme 0 demontre
===================
```

---

res_SYN056+1.p

```
pel26
****************************************************
theoreme a demontrer

- (qqs(A, big_p(A) => big_r(A)) <=> qqs(B, big_q(B) => big_s(B))).

0 nouvelle conclusion
- (qqs(A, big_p(A) => big_r(A)) <=> qqs(B, big_q(B) => big_s(B))).

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2 seul
.. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ pel26_1_ pel26_1_1 pel26_2_ pel26_2_1 reflexivity_ et
defconcl1 defconcl1a defconcl1b defconcl1bb ilexiste ou defconcl2 defconcl2app
defconcl2elt defconcl2a defconcl2b defconcl3

0 nouvelle conclusion
- (qqs(A, big_p(A) => big_r(A)) => qqs(B, big_q(B) => big_s(B))) et (qqs(B, big_q(B)
=> big_s(B)) => qqs(A, big_p(A) => big_r(A))).

--------------------------------------------------------- regle <=>

************************************************ sous - theoreme 1 *****
1 nouvelle conclusion
- (qqs(A, big_p(A) => big_r(A)) => qqs(B, big_q(B) => big_s(B))).

----------------------------------------------- creation du sous - theoreme 1
1 ajouter la regle active locale

- (regle(A, reghyp) :- hyp(A, big_p(B)), not hyp(A, big_r(B)), ajhyp(A, big_r(B))).

a la fin des regles universelles

1 nouvelle conclusion
- qqs(A, big_q(A) => big_s(A)).

--------------------------------------------------------- regle =>
1 ajouter objet x
1 nouvelle conclusion big_q(x) => big_s(x)
--------------------------------------------------------- regle qqs1
1 ajouter hypothese big_q(x)
1 nouvelle conclusion big_s(x)
--------------------------------------------------------- regle =>
1 ajouter hypothese
- ilexiste(A, big_p(A)).
--------------------------------------------------------- regle pel26_1_1
1 ajouter objet x1
1 ajouter hypothese big_p(x1)
1 ajouter hypothese-traitee
- ilexiste(A, big_p(A)).
--------------------------------------------------------- regle ilexiste
1 ajouter hypothese big_r(x1)
--------------------------------------------------------- regle reghyp
1 ajouter hypothese
- ilexiste(A, big_q(A)).
--------------------------------------------------------- regle pel26_1_
1 ajouter hypothese big_s(x)
--------------------------------------------------------- regle pel26_2_
1 nouvelle conclusion vrai
```

```
--------------------------------------------------------- regle stop1
theoreme 1 demontre
===================
0 nouvelle conclusion
- (qqs(A, big_q(A) => big_s(A)) => qqs(B, big_p(B) => big_r(B))).


************************************************ sous - theoreme 2 *****
2 nouvelle conclusion
- (qqs(A, big_q(A) => big_s(A)) => qqs(B, big_p(B) => big_r(B))).

----------------------------------------------- creation du sous - theoreme 2
2 ajouter la regle active locale

- (regle(A, reghyp1) :- hyp(A, big_q(B)), not hyp(A, big_s(B)), ajhyp(A, big_s(B))).

a la fin des regles universelles

2 nouvelle conclusion
- qqs(A, big_p(A) => big_r(A)).

--------------------------------------------------------- regle =>
2 ajouter objet x2
2 nouvelle conclusion big_p(x2) => big_r(x2)
--------------------------------------------------------- regle qqs1
2 ajouter hypothese big_p(x2)
2 nouvelle conclusion big_r(x2)
--------------------------------------------------------- regle =>
2 ajouter hypothese
- ilexiste(A, big_q(A)).
--------------------------------------------------------- regle pel26_1_
2 ajouter objet x3
2 ajouter hypothese big_q(x3)
2 ajouter hypothese-traitee
- ilexiste(A, big_q(A)).
--------------------------------------------------------- regle ilexiste
2 ajouter hypothese big_s(x3)
--------------------------------------------------------- regle reghyp1
2 ajouter hypothese
- ilexiste(A, big_p(A)).
--------------------------------------------------------- regle pel26_1_1
2 ajouter hypothese big_r(x2)
--------------------------------------------------------- regle pel26_2_1
2 nouvelle conclusion vrai
--------------------------------------------------------- regle stop1
theoreme 2 demontre
===================
0 nouvelle conclusion vrai
--------------------------------------------------------- regle et
theoreme 0 demontre
===================
```

```
%--------------------------------------------------------------------------
% File     : MGT005+1 : TPTP v2.2.0. Released v2.0.0.
% Domain   : Management (Organisation Theory)
% Problem  : Complexity increases the risk of death due to reorganization.
% Version  : [PB+94] axioms.
% English  :
% Refs     : [PB+92] Peli et al. (1992), A Logical Approach to Formalizing
%          : [PB+94] Peli et al. (1994), A Logical Approach to Formalizing
%          : [Kam94] Kamps (1994), Email to G. Sutcliffe
% Source   : [Kam94]
% Names    : THEOREM 5 [PB+92]
% Status   : theorem
% Rating   : 0.50 v2.1.0
% Syntax   : Number of formulae    :   25 (   1 unit)
%            Number of atoms       :   72 (  28 equality)
%            Maximal formula depth :    4 (   3 average)
%            Number of connectives :   47 (   0 ~  ;   0  |;  23  &)
%                                         (   0 <=>;  24 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :    9 (   0 propositional; 2-3 arity)
%            Number of functors    :    0 (   0 constant; --- arity)
%            Number of variables   :   90 (   0 singleton;  90 !;   0 ?)
%            Maximal term depth    :    1 (   1 average)
% Comments : mp11, mp12 and mp13 corrospond to mp10, mp11 and mp12
%            respectively from [PB+92]
%--------------------------------------------------------------------------
%----Include equality axioms
include('Axioms/EQU001+0.ax').
%--------------------------------------------------------------------------
%----Substitution axioms
input_formula(class_substitution_1,axiom,(
    ! [A,B,C,D] :
      ( ( equal(A,B)
        & class(A,C,D) )
     => class(B,C,D) )   )).

............ etc ............

%----Problem axioms
input_formula(mp6_1,axiom,(
    ! [X,Y] :
      ~ ( greater(X,Y)
        & equal(X,Y) )   )).

input_formula(mp6_2,axiom,(
    ! [X,Y] :
      ~ ( greater(X,Y)
        & greater(Y,X) )   )).

input_formula(mp11,axiom,(
    ! [X,Y,Z] :
      ( ( greater(X,Y)
        & greater(Y,Z) )
     => greater(X,Z) )   )).

input_formula(mp12,axiom,(
    ! [X,T] :
      ( organization(X,T)
     => ? [P] : survival_chance(X,P,T) )   )).

input_formula(mp13,axiom,(
    ! [X,T,T1,T2] :
      ( ( organization(X,T1)
        & organization(X,T2)
        & greater(T,T1)
        & greater(T2,T) )
     => organization(X,T) )   )).

input_formula(mp7,axiom,(
```

```
    ! [X,Ta,Tb] :
      ( reorganization(X,Ta,Tb)
     => greater(Tb,Ta) )   )).

input_formula(t3_FOL,hypothesis,(
    ! [X,P1,P2,T1,T2] :
      ( ( organization(X,T1)
        & organization(X,T2)
        & reorganization_free(X,T1,T2)
        & survival_chance(X,P1,T1)
        & survival_chance(X,P2,T2)
        & greater(T2,T1) )
     => greater(P2,P1) )   )).

%----t4_FOL - alias a9_FOL
input_formula(t4_FOL,hypothesis,(
    ! [X,P1,P2,T1,T2,Ta,Tb] :
      ( ( organization(X,T1)
        & organization(X,T2)
        & reorganization(X,Ta,Tb)
        & survival_chance(X,P1,T1)
        & survival_chance(X,P2,T2)
        & ~ greater(Ta,T1)
        & greater(T2,T1)
        & ~ greater(T2,Tb) )
     => greater(P1,P2) )   )).

%----Complexity increases the expected duration of reorganization.
input_formula(a10_FOL,hypothesis,(
    ! [X,Y,Re,C,C1,C2,Ta,Tb,Tc] :
      ( ( organization(X,Ta)
        & organization(Y,Ta)
        & organization(Y,Tc)
        & class(X,C,Ta)
        & class(Y,C,Ta)
        & reorganization(X,Ta,Tb)
        & reorganization(Y,Ta,Tc)
        & reorganization_type(X,Re,Ta)
        & reorganization_type(Y,Re,Ta)
        & complexity(X,C1,Ta)
        & complexity(Y,C2,Ta)
        & greater(C2,C1) )
     => greater(Tc,Tb) )   )).

%----Complexity is no survival advantage during reorganization.
input_formula(a11_FOL,hypothesis,(
    ! [X,Y,Re,C,P,P1,P2,C1,C2,Ta,Tb,Tc] :
      ( ( organization(X,Ta)
        & organization(Y,Ta)
        & organization(X,Tb)
        & organization(Y,Tb)
        & class(X,C,Ta)
        & class(Y,C,Ta)
        & survival_chance(X,P,Ta)
        & survival_chance(Y,P,Ta)
        & reorganization(X,Ta,Tb)
        & reorganization(Y,Ta,Tc)
        & reorganization_type(X,Re,Ta)
        & reorganization_type(Y,Re,Ta)
        & survival_chance(X,P1,Tb)
        & survival_chance(Y,P2,Tb)
        & complexity(X,C1,Ta)
        & complexity(Y,C2,Ta)
        & greater(C2,C1) )
     => ( greater(P1,P2)
        | equal(P1,P2) ) )   )).

input_formula(t5_FOL,conjecture,(
    ! [X,Y,Re,C,P,P1,P2,C1,C2,Ta,Tb,Tc] :
      ( ( organization(X,Ta)
        & organization(Y,Ta)
        & organization(X,Tc)
```

```
        & organization(Y,Tc)
        & class(X,C,Ta)
        & class(Y,C,Ta)
        & survival_chance(X,P,Ta)
        & survival_chance(Y,P,Ta)
        & reorganization(X,Ta,Tb)
        & reorganization(Y,Ta,Tc)
        & reorganization_type(X,Re,Ta)
        & reorganization_type(Y,Re,Ta)
        & reorganization_free(X,Tb,Tc)
        & survival_chance(X,P1,Tc)
        & survival_chance(Y,P2,Tc)
        & complexity(X,C1,Ta)
        & complexity(Y,C2,Ta)
        & greater(C2,C1) )
     => greater(P1,P2) )   )).
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% File     : MGT021+1 : TPTP v2.2.0. Released v2.0.0.
% Problem  : Difference between disbanding rates does not decrease
%--------------------------------------------------------------------------
%----Subsitution axioms
...
%----Problem axioms
%----MP. If first movers and efficient producers are present in an
%----environment  at a certain point of time, then this time-point belongs
%----to the the environment.
input_formula(mp_time_point_in_environment,axiom,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => in_environment(E,T) )   )).
%----MP. If first movers and efficient producers are present in an
%----environment  at a certain point of time, then then the environment
%----is not empty at this time.
input_formula(mp_environment_not_empty,axiom,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => greater(number_of_organizations(E,T),zero) )   )).
%----MP. If something increases, then it does not decrease.
input_formula(mp_increase_not_decrease,axiom,(
    ! [X] :
      ( increases(X)
     => ~ decreases(X) )   )).
%----MP. on "greater or equal to"
input_formula(mp_greater_or_equal,axiom,(
    ! [X,Y] :
      ( greater_or_equal(X,Y)
     => ( greater(X,Y)
        | equal(X,Y) ) )   )).
%----A3. Resource availability decreases until equilibrium is reached.
input_formula(a3,hypothesis,(
    ! [E,T] :
      ( ( environment(E)
        & in_environment(E,T)
        & greater(number_of_organizations(E,T),zero) )
     => ( ( greater(equilibrium(E),T)
         => decreases(resources(E,T)) )
        & ( ~ greater(equilibrium(E),T)
         => constant(resources(E,T)) ) ) )   )).
%----L4. A decreasing resource availability affects the disbanding rate of
%----first movers more than the disbanding rate of efficient producers.
input_formula(l4,hypothesis,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => ( ( decreases(resources(E,T))
         => increases(difference(disbanding_rate(first_movers,T),
disbanding_rate(efficient_producers,T))) )
        & ( constant(resources(E,T))
```

```
         => ~ decreases(difference(disbanding_rate(first_movers,T),
disbanding_rate(efficient_producers,T))) ) ) )   )).
%----GOAL: L3. The difference between the disbanding rates of first movers
%----and efficient producers does not decrease.
input_formula(prove_l3,conjecture,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => ~ decreases(difference(disbanding_rate(first_movers,T),
disbanding_rate(efficient_producers,T))) )   )).
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% File     : MGT023+1 : TPTP v2.2.0. Released v2.0.0.
% Problem  : Stable environments have a critical point.
%--------------------------------------------------------------------------
%----Subsitution axioms
...
%----Problem axioms
%----D1=>. A time point is the critical point of an environmental patch,
%----if and only if, it is the earliest time past which the growth rate of
%----efficient producers permanently exceeds growth rate of first movers.
input_formula(d1,hypothesis,(
    ! [E,To] :
      ( ( environment(E)
        & ~ greater(growth_rate(efficient_producers,To),
growth_rate(first_movers,To))
        & in_environment(E,To)
        & ! [T] :
            ( ( subpopulations(first_movers,efficient_producers,E,T)
              & greater(T,To) )
           => greater(growth_rate(efficient_producers,T),
growth_rate(first_movers,T)) ) )
     => equal(To,critical_point(E)) )   )).
%----L12. There is an earliest time point, past which FM's growth rate
%----exceeds EP's growth rate.
input_formula(l12,hypothesis,(
    ! [E] :
      ( ( environment(E)
        & stable(E) )
     => ? [To] :
          ( in_environment(E,To)
          & ~ greater(growth_rate(efficient_producers,To),
growth_rate(first_movers,To))
          & ! [T] :
              ( ( subpopulations(first_movers,efficient_producers,E,T)
                & greater(T,To) )
             => greater(growth_rate(efficient_producers,T),
growth_rate(first_movers,T)) ) ) )   )).

%----GOAL: L5. Stable environments have a critical point.
input_formula(prove_l5,conjecture,(
    ! [E] :
      ( ( environment(E)
        & stable(E) )
     => in_environment(E,critical_point(E)) )   )).
%--------------------------------------------------------------------------
```

ANNEXE 4 Exemple COM

```
%------------------------------------------------------------------------------
% File     : COM003+2 : TPTP v2.2.0. Bugfixed v2.2.0.
% Domain   : Computing Theory
% Problem  : The halting problem is undecidable
% Version  : [Bru91] axioms.
% English  :
% Refs     : [Gan98] Ganzinger (1998), Email to Geoff Sutcliffe
%          : [Bur87b] Burkholder (1987), A 76th Automated Theorem Proving Pr
%          : [Bru91] Brushi (1991), The Halting Problem
% Source   : [Bru91]
% Names    : - [Bru91]
% Status   : theorem
% Rating   : ? v2.2.0
% Syntax   : Number of formulae    :   16 (   1 unit)
%            Number of atoms       :   52 (   0 equality)
%            Maximal formula depth :    8 (   4 average)
%            Number of connectives :   39 (   3 ~  ;   0  |;  15  &)
%                                         (  11 <=>;  10 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :   17 (   0 propositional; 1-4 arity)
%            Number of functors    :    2 (   2 constant; 0-0 arity)
%            Number of variables   :   42 (   0 singleton;  37 !;   7 ?)
%            Maximal term depth    :    1 (   1 average)
% Comments :
% Bugfixes : v2.2.0 - Clauses program_halts2_halts3_outputs_def, program_
%            not_halts2_halts3_outputs_def, program_halts2_halts2_outputs_
%            def, program_not_halts2_halts2_outputs_def, p4 by [Gan98].
%------------------------------------------------------------------------------
input_formula(program_decides_def,axiom,(
    ! [X] :
      ( program_decides(X)
    <=> ! [Y] :
          ( program(Y)
         => ! [Z] : decides(X,Y,Z) ) )   )).

input_formula(program_program_decides_def,axiom,(
    ! [X] :
      ( program_program_decides(X)
    <=> ( program(X)
        & program_decides(X) ) )   )).

input_formula(algorithm_program_decides_def,axiom,(
    ! [X] :
      ( algorithm_program_decides(X)
    <=> ( algorithm(X)
        & program_decides(X) ) )   )).

input_formula(program_halts2_def,axiom,(
    ! [X,Y] :
      ( program_halts2(X,Y)
    <=> ( program(X)
        & halts2(X,Y) ) )   )).

input_formula(halts3_outputs_def,axiom,(
    ! [X,Y,Z,W] :
      ( halts3_outputs(X,Y,Z,W)
    <=> ( halts3(X,Y,Z)
        & outputs(X,W) ) )   )).

input_formula(program_not_halts2_def,axiom,(
    ! [X,Y] :
      ( program_not_halts2(X,Y)
    <=> ( program(X)
        & ~ halts2(X,Y) ) )   )).

input_formula(halts2_outputs_def,axiom,(
    ! [X,Y,W] :
      ( halts2_outputs(X,Y,W)
    <=> ( halts2(X,Y)
```

```
        & outputs(X,W) ) )   )).

input_formula(program_halts2_halts3_outputs_def,axiom,(
    ! [X,Y,Z,W] :
      ( program_halts2_halts3_outputs(X,Y,Z,W)
    <=> ( program_halts2(Y,Z)
       => halts3_outputs(X,Y,Z,W) ) )   )).

input_formula(program_not_halts2_halts3_outputs_def,axiom,(
    ! [X,Y,Z,W] :
      ( program_not_halts2_halts3_outputs(X,Y,Z,W)
    <=> ( program_not_halts2(Y,Z)
       => halts3_outputs(X,Y,Z,W) ) )   )).

input_formula(program_halts2_halts2_outputs_def,axiom,(
    ! [X,Y,W] :
      ( program_halts2_halts2_outputs(X,Y,W)
    <=> ( program_halts2(Y,Y)
       => halts2_outputs(X,Y,W) ) )   )).

input_formula(program_not_halts2_halts2_outputs_def,axiom,(
    ! [X,Y,W] :
      ( program_not_halts2_halts2_outputs(X,Y,W)
    <=> ( program_not_halts2(Y,Y)
       => halts2_outputs(X,Y,W) ) )   )).

input_formula(p1,axiom,(
    ? [X] : algorithm_program_decides(X)
 => ? [W] : program_program_decides(W)   )).

input_formula(p2,axiom,(
    ! [W] :
      ( program_program_decides(W)
     => ! [Y,Z] :
          ( program_halts2_halts3_outputs(W,Y,Z,good)
          & program_not_halts2_halts3_outputs(W,Y,Z,bad) ) )   )).

input_formula(p3,axiom,(
    ? [W] :
      ( program(W)
      & ! [Y] :
          ( program_halts2_halts3_outputs(W,Y,Y,good)
          & program_not_halts2_halts3_outputs(W,Y,Y,bad) ) )
 => ? [V] :
      ( program(V)
      & ! [Y] :
          ( program_halts2_halts2_outputs(V,Y,good)
          & program_not_halts2_halts2_outputs(V,Y,bad) ) )   )).

input_formula(p4,axiom,(
    ? [V] :
      ( program(V)
      & ! [Y] :
          ( program_halts2_halts2_outputs(V,Y,good)
          & program_not_halts2_halts2_outputs(V,Y,bad) ) )
 => ? [U] :
      ( program(U)
      & ! [Y] :
          ( ( program_halts2(Y,Y)
           => ~ halts2(U,Y) )
          & program_not_halts2_halts2_outputs(U,Y,good) ) )   )).

input_formula(prove_this,conjecture,(
    ~ ( ? [X] : algorithm_program_decides(X) )   )).
%--------------------------------------------------------------------------
```

ANNEXE 5 Exemples de définitions SET

$A \subset B \Leftrightarrow \forall X\ (X \in A \Rightarrow X \in B)$

```
definition( A inc B <=> qqs(X,(X app A => X app B))).
input_formula(subset,axiom,(
    ! [A,B] :
      ( subset(A,B)
    <=> ! [X] :
          ( member(X,A)
         => member(X,B) ) )   )).
```

$A \cap B = \{X \mid X \in A \wedge X \in B)$

```
definition(A inter B = [X,X app A et X app B]).
input_formula(intersection,axiom,(
    ! [X,A,B] :
      ( member(X,intersection(A,B))
    <=> ( member(X,A)
        & member(X,B) ) )   )).
```

produit(A) $= \{Y \mid \forall X\ (X \in A \Rightarrow Y \in X)\} = \bigcap_{X \in A} X$

```
definition( produit(A) = [Y, qqs(X, X app A => Y app X)]).
input_formula(product,axiom,(
    ! [X,A] :
      ( member(X,product(A))
    <=> ! [Y] :
          ( member(Y,A)
         => member(X,Y) ) )   )).
```

$\forall X \in A\ \forall Z \in C\ (Z : H(X) \Leftrightarrow \exists X \in B\ (Y : F(X)\ \wedge\ Z : G(Y))$

```
definition(comp(G,F,A,B,C):H <=> qqs(X app A, qqs(Z app C,
                  H(X):Z <=> ilexiste(Y app B, F(X):Y et G(Y):Z)))).
definition(comp(G,F,A,B,C):H <=> qqs(X app A, qqs(Z app C,
             ..[H,X]:Z <=> ilexiste(Y app B, ..[F,X]:Y et ..[G,Y]:Z)))).
definition(comp(G,F,A,B,C)=H <=> qqs(X app A, qqs(Z app C,               FAUX !
             ..[H,X]:Z <=> ilexiste(Y app B, ..[F,X]:Y et ..[G,Y]:Z)))).
```

nouvel énoncé : $\forall X \in A\ \forall Z \in C\ (Z : G_o F_{A,B,C}(X) \Leftrightarrow \exists X \in B\ (Y : F(X)\ \wedge\ Z : G(Y))$

```
     qqs(X app A, qqs(Z app C, ..[comp(G,F,A,B,C),X]:Z <=>
                              ilexiste(Y app B, ..[F,X]:Y) et ..[G,Y]:Z)))).
input_formula(compose_function,axiom,(
    ! [G,F,A,B,C,X,Z] :
      ( ( member(X,A)
        & member(Z,C) )
     => ( apply(compose_function(G,F,A,B,C),X,Z)
      <=> ? [Y] :
            ( member(Y,B)
            & apply(F,X,Y)
            & apply(G,Y,Z) ) ) )   )).
```

transitive(R,E) $\Leftrightarrow \forall X \in E\ \forall Y \in E\ \forall Z \in E\ (R(X,Y) \Rightarrow R(Y,Z) \wedge R(X,Z))$

```
definition(transitive(R,E) <=>
            qqs(X app E,qqs(Y app E,qqs(Z app E,
                        (..[R,X,Y] et ..[R,Y,Z] => ..[R,X,Z]))))).
```

ANNEXE 6 Démonstrations du théorème SYN332+1 (monitoring)

$$\begin{array}{l} \text{Soit } F = \exists X\, \exists Y\, \forall Z\, \Big([f(X,Y) \land f(Y,X) \Leftrightarrow f(X,Z)] \\ \qquad\qquad\qquad\qquad \Rightarrow \Big[[f(X,Z) \Leftrightarrow f(Z,X)] \\ \qquad\qquad\qquad\qquad\qquad \Rightarrow \big([f(X,Z) \Leftrightarrow f(Y,Z)] \\ \qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow \big[([f(Y,X) \Rightarrow f(X,Y)] \Leftrightarrow f(Z,Z)) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow ([f(X,Y) \Leftrightarrow f(Y,X)] \Leftrightarrow f(Z,Y))\big]\big)\Big]\Big) \end{array} \tag{1}$$

Cet énoncé est d'abord réécrit de la façon suivante (assez naturelle et que MUSCADET est capable de faire) :

$$\begin{array}{l} F \equiv \exists X\, \exists Y\, \forall Z\, \Big(\quad [f(X,Y) \land f(Y,X) \Leftrightarrow f(X,Z)] \\ \qquad\qquad\qquad \land [f(X,Z) \Leftrightarrow f(Z,X)] \\ \qquad\qquad\qquad \land [f(X,Z) \Leftrightarrow f(Y,Z)] \\ \qquad\qquad\qquad \land ([f(Y,X) \Rightarrow f(X,Y)] \Leftrightarrow f(Z,Z)) \\ \quad \Rightarrow ([f(X,Y) \Leftrightarrow f(Y,X)] \Leftrightarrow f(Z,Y)) \ \Big) \end{array} \tag{2}$$

1.

Je me suis aidée de considérations sémantiques et j'ai réécrit cette formule sous la forme suivante, équivalente[1] :

$\exists X\, \exists Y\, [f(X,Y) \land f(Y,X) \land \forall Z\, (f(X,Z) \land f(Z,X) \land f(Y,Z) \land f(Z,Z) \Rightarrow f(Z,Y))]$ (3)

$\lor\, \exists X\, \exists Y\, [f(X,Y) \land \neg f(Y,X) \land \forall Z\, (\neg f(X,Z) \land \neg f(Z,X) \land \neg f(Y,Z) \land f(Z,Z) \Rightarrow \neg f(Z,Y))]$

$\lor\, \exists X\, \exists Y\, [\neg f(X,Y) \land f(Y,X) \land \forall Z\, (\neg f(X,Z) \land \neg f(Z,X) \land \neg f(Y,Z) \land \neg f(Z,Z) \Rightarrow \neg f(Z,Y))]$

$\lor\, \exists X\, \exists Y\, [\neg f(X,Y) \land \neg f(Y,X) \land \forall Z\, (\neg f(X,Z) \land \neg f(Z,X) \land \neg f(Y,Z) \land f(Z,Z) \Rightarrow f(Z,Y))]$

et j'ai fait le raisonnement suivant :

- si on a $\exists X\, f(X,X)$, soit a tel que $f(a,a)$, en prenant $X=Y=a$ la première sous-formule est vérifiée
- sinon, on a $\forall X\, \neg f(X,X)$, la formule devient alors

  $\exists X\, \exists Y\, [f(X,Y) \land f(Y,X)]$

  $\lor\, \exists X\, \exists Y\, [f(X,Y) \land \neg f(Y,X)]$

  $\lor\, \exists X\, \exists Y\, [\neg f(X,Y) \land f(Y,X) \land \forall Z\, (\neg f(X,Z) \land \neg f(Z,X) \land \neg f(Y,Z) \Rightarrow \neg f(Z,Y))]$

  $\lor\, \exists X\, \exists Y\, [\neg f(X,Y) \land \neg f(Y,X)]$

  - soient deux éléments a et b s'ils existent
    - si $f(a,b) = f(b,a)$, en prenant $X=a$ et $Y=b$, la première ou la quatrième sous-formule est vérifiée
    - si on a $f(a,b)$ et $\neg f(b,a)$, en prenant $X=a$ et $Y=b$, la deuxième sous-formule est vérifiée
    - si on a $\neg f(a,b)$ et $f(b,a)$, en prenant $X=b$ et $Y=a$, la deuxième sous-formule est vérifiée
  - s'il n'existe pas deux éléments distincts, mais un seul[2], a, on a $\neg f(a,a)$ et en prenant $X=Y=a$, la quatrième sous-formule est vérifiée.

Ce raisonnement ayant permis de trouver l'idée de faire un raisonnement par cas, $\exists X\, f(X,X)$ ou bien $\forall X\, \neg f(X,X)$, on peut maintenant faire un raisonnement plus formel.

2.

$\neg F$ peut s'écrire

$\forall X\, \forall Y\, [f(X,Y) \land f(Y,X) \Rightarrow \exists Z\, (f(X,Z) \land f(Z,X) \land f(Y,Z) \land f(Z,Z) \land \neg f(Z,Y))]$ (3')

$\land\, \forall X\, \forall Y\, [f(X,Y) \land \neg f(Y,X) \Rightarrow \exists Z\, (\neg f(X,Z) \land \neg f(Z,X) \land \neg f(Y,Z) \land f(Z,Z) \land f(Z,Y))]$

$\land\, \forall X\, \forall Y\, [\neg f(X,Y) \land f(Y,X) \Rightarrow \exists Z\, (\neg f(X,Z) \land \neg f(Z,X) \land \neg f(Y,Z) \land \neg f(Z,Z) \land f(Z,Y))]$

$\land\, \forall X\, \forall Y\, [\neg f(X,Y) \land \neg f(Y,X) \Rightarrow \exists Z\, (\neg f(X,Z) \land \neg f(Z,X) \land \neg f(Y,Z) \land f(Z,Z) \land \neg f(Z,Y))]$

On démontre alors facilement, à partir de la première sous-formule, que

$\neg F \Rightarrow \forall X\, [f(X,X) \Rightarrow \exists Z\, (f(X,Z) \land f(Z,X) \land f(Z,Z) \land \neg f(Z,X)]$

---

[1] L'idée et de considérer explicitement les quatre cas possibles pour f(X,Y) et f(Y,X)

[2] Il y en a toujours au moins un car le domaine d'une interprétation est supposé non vide.

d'où $\neg F \Rightarrow \forall X \neg f(X,X)$
puis, à partir de la quatrième sous-formule, que
$\neg F \wedge \forall X \neg f(X,X) \Rightarrow \forall X \forall Y [f(X,Y) \vee f(Y,X)]$
puis $\neg F \wedge \forall X \neg f(X,X) \Rightarrow \forall X f(X,X)$
donc $\neg F \Rightarrow \forall X \neg f(X,X) \wedge \forall X f(X,X)$
$\neg F$ est donc fausse et F est un théorème.

Je me suis alors aperçue que le premier raisonnement utilisait la première, la deuxième et la quatrième sous-formules alors que le deuxième raisonnement n'utilisait que la première et la quatrième sous-formules. En en cherchant les raisons, j'ai simplifié la première démonstration dont la fin devient:

3. comme 1. puis
- si on a $\exists X f(X,X)$, soit a tel que f(a,a), en prenant X=Y=a la première sous-formule est vérifiée
- sinon, on a $\forall X \neg f(X,X)$, soit a quelconque, en prenant X=Y=a la quatrième sous-formule est vérifiée.

Il est intéressant de remarquer que le premier raisonnement, basé sur la sémantique et la recherche d'interprétation a permis de trouver le deuxième raisonnement plus formel qui a permis de simplifier le premier raisonnement.

Cette démonstration ne me satisfaisait pas vraiment, car je considérais quatre sous-formules correspondant à quatre cas dont deux étaient inutiles. Je me suis alors aperçue, que l'on pouvait, version sémantique, conclure directement que

4.
- si on a $\exists X f(X,X)$, soit a tel que f(a,a), en prenant X=Y=a, F (sous la forme (2)) devient
$\forall Z (f(a,Z) \wedge f(Z,a) \wedge f(Z,Z) \Rightarrow f(Z,a))$ qui est vraie
- sinon, on a $\forall X \neg f(X,X)$, soit a quelconque, en prenant X=Y=a, F devient
$\forall Z (\neg f(a,Z) \wedge \neg f(Z,a) \wedge f(Z,Z) \Rightarrow f(Z,a))$ qui est vraie puisque f(Z,Z) est fausse.

Pour retrouver une démonstration formelle, après quelques manipulations, et après avoir remarqué que les interprétations conduisaient à toujours prendre X=Y, j'ai introduit la formule G = $\exists X \forall Z \Big([f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge [f(X,Z) \Leftrightarrow f(X,Z)]$

$$\wedge \big([f(X,X) \Rightarrow f(X,X)] \Leftrightarrow f(Z,Z)\big)$$

$$\Rightarrow \big([f(X,X) \Leftrightarrow f(X,X)] \Leftrightarrow f(Z,X)\big) \Big)$$

équivalente à

$$\exists X \forall Z \big([f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge f(Z,Z) \Rightarrow f(Z,X) \big)$$

On a $G \Rightarrow F$ et $\neg F \Rightarrow \neg G$
Il suffit de montrer que G est un théorème ou que $\neg G$ n'en est pas un.
Le plus facile est de démontrer que $\neg G$ n'est pas un théorème, soit

5.

$$\neg G \equiv \forall X \exists Z \big([f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge f(Z,Z) \wedge \neg f(Z,X)\big)$$

$$\equiv \forall X \exists Z \big([f(X,X) \wedge f(X,Z) \wedge f(Z,X) \wedge f(Z,Z) \wedge \neg f(Z,X)]$$

$$\vee [\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z)]\big)$$

$$\equiv \forall X \exists Z \big(\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z)\big)$$

$$\equiv \forall X \neg f(X,X) \wedge \forall X \exists Z \big(\neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z)\big)$$

qui est fausse car le f(Z,Z) de la deuxième sous-formule est faux à cause de la première sous-formule.

On peut montrer directement que G est un théorème :

6.

$$G = \exists X\ \forall Z\ \big([f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge [f(X,Z) \Leftrightarrow f(X,Z)] \wedge f(Z,Z) \Rightarrow f(Z,X)\big)$$

$$\equiv \exists X\ \forall Z\ \big([f(X,X) \wedge f(X,Z) \wedge f(Z,X) \wedge f(Z,Z) \Rightarrow f(Z,X)]$$

$$\wedge\ [\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z) \Rightarrow f(Z,X)]\ \big)$$

$$\equiv \exists X\ \forall Z\ (\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z) \Rightarrow f(Z,X)\ )$$

$$\equiv \exists X\ \forall Z\ (\ f(X,X) \vee f(X,Z) \vee f(Z,X) \vee \neg f(Z,Z))$$

$$\equiv \exists X\ f(X,X) \vee \exists X\ \forall Z\ (\ f(X,Z) \vee f(Z,X) \vee \neg f(Z,Z)\ )$$

$$\equiv \exists X\ f(X,X) \vee \big(\forall X\ \neg f(X,X) \wedge \exists X\ \forall Z\ (\ f(X,Z) \vee f(Z,X) \vee \neg f(Z,Z)\ )\ \big)$$

$$\equiv \exists X\ f(X,X) \vee \forall X\ \neg f(X,X)$$

qui est vraie, ainsi que F puisqu'on a $G \Rightarrow F$

Cette dernière démonstration est celle que je considère comme la *meilleure*. Il faut insister sur le fait que c'est la succession des autres démonstrations qui m'a permis d'aboutir à celle-ci. Il faut aussi remarquer que toutes les démonstrations utilisent de nombreux savoir-faire de manipulations symboliques et que celles-ci ont été guidées par la sémantique. L'apprentissage a aussi joué son rôle : au début, les sous-formules semblaient obscures, au bout d'un certain temps, elles sont devenues limpides …

On remarquera aussi que l'idée d'exploiter le cas X=Y (coalescence) est une heuristique classique en Intelligence artificielle (Lenat 1982, Pitrat 1990) mais dont l'utilisation doit être prudente.

Automatisation du raisonnement et de la rédaction de preuves en géométrie

Jean Pierre Spagnol
Crip5
Université René Descartes Paris5

Résumé : L’objectif de cette recherche est la conception d’un système à base de connaissances, capable de résoudre de façon automatique des exercices de géométrie niveau collège ou lycée. Il devra ensuite être capable de filtrer la base des faits déduits pour en produire, à partir du graphe de la démonstration, une solution rédigée en français adaptée au niveau de la classe considérée. Dans cet article nous décrivons un certain nombre de techniques mises en œuvre, conçues à partir du système Muscadet, et essayant d’en garder l’esprit qui est basé sur le principe de la déduction naturelle. Ce système est destiné à une utilisation en EIAO.

Mots-clés : DAT, automatisation du raisonnement mathématique, EIAO, explication.

1. Introduction

L’objectif de cette recherche est la conception d’un système à base de connaissances, capable de résoudre de façon automatique des exercices de géométrie niveau collège ou lycée. Il devra ensuite être capable de filtrer la base des faits déduits pour en produire une solution rédigée en français adaptée au niveau de la classe considérée. Ce système est conçu à partir du système Muscadet [PASTRE 84] réécrit en langage Prolog et veut en garder l’esprit, basé sur la déduction naturelle. Ce système est destiné à une utilisation en EIAO pour permettre à un élève de lycée d’améliorer son intuition et d’étayer sa recherche en lui permettant de proposer des conjectures vérifiables par le système à partir d’un énoncé donné. Il est aussi destiné aux professeurs en leur permettant d’exprimer les connaissances mathématiques, sur un thème donné, de façon déclarative et de concevoir des exercices s’y rapportant. Il apparaît difficilement concevable qu’un démonstrateur, résolve des exercices et démontre des théorèmes sans que les preuves puissent être lisibles et vérifiables par un humain. Un gros effort d'explication est requis pour ce système. Dans cet article, après de brèves remarques générales, nous montrerons le fonctionnement général et nous l'illustrerons par un exemple détaillé. Nous montrerons ensuite comment le système reconstruit l'arbre de preuve à partir des faits déduits et à l'aide de celui-ci produit une preuve rédigée en français de l'exercice proposé. Après examen des domaines de recherche du démonstrateur nous parlerons des problèmes rencontrés en démonstration de théorèmes. Nous évoquerons ensuite la possibilité pour le système de construire automatiquement des règles à partir du savoir mathématique exprimé de façon déclarative. Nous examinerons la possibilité qu'a le système de lancer des appels de conjectures si certaines conditions sont réunies dans la base de faits.

2. Remarques générales

Muscadet était un système à base de connaissances dont le moteur d’inférence fonctionnait comme une boîte noire. Dominique Pastre en a fait une retranscription en Prolog [Pastre Berder 99], de nom Puscadet (ou Muscadet version 2), à partir de laquelle commence notre travail. Il est ainsi possible grâce à la bivalence du langage Prolog d’exprimer les connaissances de façon déclarative et de faire du procédural quand cela est nécessaire, en particulier il devient possible d’appeler des procédures de calcul ce qui est parfois nécessaire en géométrie.

3. Fonctionnement général du système

3.1. Lecture de l'énoncé

Le système va lire, dans un fichier, une liste d’énoncés pour résoudre les exercices un à un.

3.2. Analyse de l'énoncé

Il y a ensuite extraction de concepts avec prise en compte du fait de figurer dans la partie hypothèses ou conclusion de l’énoncé. En effet un concept présent dans la partie hypothèses de l'énoncé ne sera pas utilisé de la même façon que s'il figure en conclusion. D’autres concepts sont introduits par le biais de deux relations, la relation hiérarchique est_un, et la relation a_besoin_de. De plus dans le même parcours on procède à l’ élimination de certains symboles fonctionnels.

3.3. Sélection et chargement des règles adéquates, dans un certain ordre.

Le système charge les règles universelles indépendantes de tout énoncé (manipulation logique de l'énoncé, sortie du système, cohérence au niveau des objets et règles nécessaires quel que soit l'énoncé), puis les règles d’introduction des objets initiaux, les règles de condition suffisante de la conclusion permettant de conclure rapidement si un critère suffisant de la conclusion est utilisable dans la base de faits.

Le démonstrateur peut ensuite charger les règles appelant une conjecture si le système détecte des configurations incomplètes et intéressantes en fonction de ce qu’il y a à démontrer. Ceci est une option que peut choisir un utilisateur et il doit alors l'indiquer au système.

Il faut maintenant charger les règles que le système juge utile pour la démonstration. Ce sont les règles obligatoires avec la terminologie Muscadet. En fonction des concepts présents en hypothèse on charge des règles de démarche en avant (type : déduction). Ces règles extraient de la figure toutes les configurations prégnantes sans être guidées par le but. Pour les concepts figurant en conclusion on cherche celles de démarche en fonction du but (par exemple : si on veux montrer que deux droites sont orthogonales alors le concept d’orthogonalité figure en conclusion. Il faut rechercher, après un filtrage en fonction des autres concepts, les règles dont la partie action rajoute en hypothèse des faits d’orthogonalité ou permettant d’induire par la suite de tels faits).

Le système va charger enfin des règles un peu plus dangereuses de création d'objets dont la présence n'est pas vue comme nécessaire par simple analyse de l'énoncé mais qui sont susceptibles de servir à l’application de règles utiles pour l’exercice en cours.

Le moteur d’inférences entre alors en action et va chercher à instancier successivement chacune des règles ayant été placées dans la liste des règles actives.

3.4. Manipulation logique de l'énoncé

Un énoncé est écrit sous la forme : H_1 et H_2 et ….. et H_n => C_1 et C_2 et …. et C_k .
Le système procède à un découpage de l’énoncé et commence à structurer la base de faits selon le principe de Muscadet qui correspond aussi à la démarche du mathématicien. La base de faits se structure en une partie Hypothèses où vont se rajouter tous les faits déduits à partir des hypothèses initiales par application des règles chargées, et d’une partie Conclusion où figurent toutes les conclusions à démontrer. Chacune d’elles sera mise à vrai si le fait correspondant est déduit par l’application d’une règle.

Il faut ensuite introduire les objets liés à l'énoncé et pouvant être utiles à la preuve.

3.5. Recherche d'instanciation de règles dans un certain ordre (Déductions tout venant).

Une règle s’applique si sa partie condition est instanciée, et alors la partie action se déclenche et rajoute la plupart du temps un fait nouveau sous la forme hyp(N,Hyp,Q). N est un argument de Puscadet standard qui mémorise le numéro du sous-théorème actif. Hyp est le nouveau fait qui va être rajouté dans la base et Q va permettre le filtrage de celle-ci pour récupérer le graphe de la démonstration. Le système mémorise les numéros des hypothèses déjà présentes qui ont permis la déduction du fait no :Q.

Chaque fois qu'une règle s'est appliquée, le moteur d’inférences repart au début de la liste des règles actives et recommence sa recherche d’instanciations.

3.6. Sortie du système

Lorsque toutes les conclusions partielles ont été prouvées, mises à vrai par le système, le système sort sur un succès et indique que l’exercice a été résolu. Si par contre le temps de recherche maximum, donné par l’utilisateur, a été dépassé ou si le moteur est arrivé en bout de la liste des règles actives sans qu’aucune ne puisse s’appliquer le système sort sur un échec et va chercher un autre énoncé pour recommencer.

3.7. Récupération du graphe de démonstration

Si le théorème a été démontré une procédure spéciale filtre la base de faits pour ne conserver que les déductions utiles à la démonstration du théorème et récupère les noms internes de tous les objets dont on aura besoin pour produire une rédaction de la démonstration compréhensible par un élève de lycée ou de collège.

3.8. Production de la rédaction en français de la démonstration

A partir du graphe de la démonstration, une procédure reconstitue par ordre chronologique les différents pas de déduction, dans une démarche en avant, en commençant depuis les hypothèses initiales et en remontant ainsi jusqu'au but final.
Chaque nœud de l'arbre est développé en indiquant qu'il a été obtenu à partir d'un certain nombre de faits et par l'application d'une règle. On peut y adjoindre une explication complémentaire explicitant, dans un langage proche de l’élève, pourquoi cette conjonction d’hypothèses a permis cette déduction.

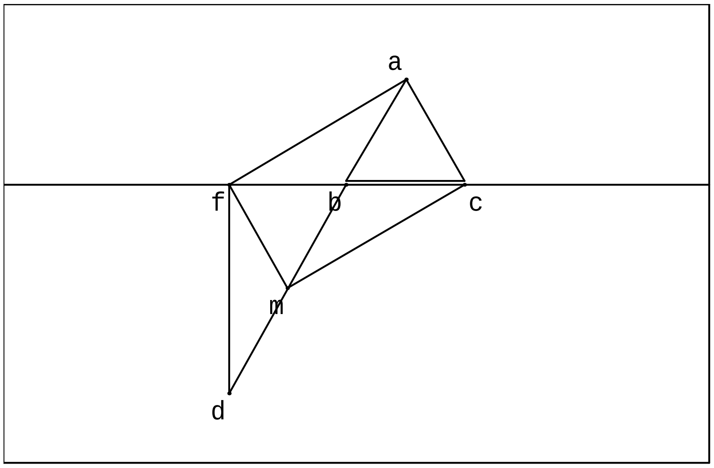
4. Illustration par un exemple détaillé

L’exemple suivant est issu de [Pintado 94].

4.1. Enoncés

Enoncé livresque:
Soit abc un triangle équilatéral, m et d deux points tels que b soit le milieu du segment [am] et m le milieu du segment [bd]. Soit f le projeté orthogonal de d sur la droite (bc). Montrer que acmf est un rectangle.

Enoncé donné au système et utilisant la notation fonctionnelle:
equilateral(a,b,c) et milieu(a,m):b et milieu(b,d):m et projeteOrthog(d,droite(b,c)):f => rect(a,c,m,f).
figure :



4.2. Analyse de l'énoncé

Le système extrait les concepts : equilateral, milieu, projeteOrthog, droite en hypothèse et rectangle en conclusion. Il rajoute ensuite des liens en hypothèse avec le concept isocele car un triangle équilateral est_un triangle isocèle, avec l’orthogonalité car un projeté orthogonal a_besoin_de l’orthogonalité. Il rajoute un lien avec parallélogramme en conclusion, car pour montrer qu'un quadrilatère est un rectangle on peut avoir besoin de montrer que c'est un parallélogramme. Ce genre de connaissances est donné au système sous la forme de prédicats comme par exemple : est_un(equilateral,isocele).

4.3. Elimination des symboles fonctionnels

Le système donne un nom dr à la droite (bc) et range le fait droite(b,c) :dr en mémoire pour un traitement ultérieur. L'énoncé devient alors :
equilateral(a,b,c) et milieu(a,m):b et milieu(b,d):m et projeteOrthog(d,dr):f =>
rect(a,c,m,f).

4.4. Sélection et chargement des règles adéquates, dans un certain ordre.

Une typologie des règles a été constituée, pour l'instant à la main, reliant les noms des règles aux concepts en fonction de leur relation avec ceux-ci. Le système va alors construire la liste des règles actives en fonction de cette typologie à l'aide des concepts ayant été extraits de l'énoncé. Il y aura les règles universelles, puis les règles d'introduction d'objets. Ici on introduit le triangle équilatéral abc, les milieux et le projeté orthogonal f. Il faut aussi introduire le quadrilatère acmf et certains éléments susceptibles d'être utiles à la démonstration. Pour écrire les règles d'introduction des objets initiaux on a fait appel à l'expertise du domaine, le problème étant que lorsque le mathématicien construit la figure il est déjà en train de faire des déductions et introduit donc les objets de façon dynamique. Le système n'a pas cette possibilité et doit donc introduire des objets mais pas trop pour éviter l'explosion combinatoire. L'introduction d'autres objets ne se fera qu'en fin de liste des règles actives par les règles de type création. Le système charge ensuite des règles de condition suffisante de la conclusion, sortes de raccourcis, permettant au système de sortir sans attendre que les règles de déduction classiques fassent leur travail. Ceci permet d'augmenter l'efficacité du démonstrateur. Il faut ensuite charger les règles obligatoires liées à l'énoncé, ceci grâce à la typologie précédente (déductions tout venant ou en fonction du but). Il y a possibilité, si l'utilisateur le désire, d'intercaler des règles appelant des conjectures si le système repère des configurations incomplètes susceptibles de faire avancer vers le but. Pour se faire il suffit d'activer, dans le module de chargement des règles, la procédure d'activation des règles de type conjecture. Ces appels de conjecture sont donc guidés par le but. Ceci est un moyen d'introduire dynamiquement des objets, un peu comme le fait le mathématicien quand il construit la figure. Des règles de création d'objets non immédiatement nécessaires sont ensuite chargées, là aussi en fonction des liens avec les concepts présents.

4.5. Déductions par instanciations de règles

Que fait alors le système ? (voir **annexe 1** pour la base de faits à l'issue de la preuve). Puisque le triangle fbd est rectangle en f alors mf = mb (voir règle en **annexe 2**). Le système déduit alors que a,b,m et d sont alignés dans cet ordre car on a une succession de milieux, que $a\hat{b}c$ est aigu puis que f,b et c sont alignés dans cet ordre. On a alors $m\hat{b}f$ = 60 ° car $a\hat{b}c$ et $m\hat{b}f$ sont des angles opposés par le sommet. Donc mbf est un triangle équilatéral car il est isocèle avec un angle de 60 degrés. On en déduit donc que bf = bm et donc b = milieu(f,c). Ceci était le point important à prouver. Il y a deux modes de fonctionnenent possibles pour le démonstrateur. Lors de l'introduction du quadrilatère acmf, le système sait déjà que b = mil(a,m), il n'a donc pas à créer un des milieux d'une des diagonales. Par contre il peut rajouter, en élément de conclusion, le fait d'avoir à montrer que b = mil(f,c) car de toutes façons il sera intéressant de montrer que acmf est un parallélogramme. C'est ce que déduit d'ailleurs celui-ci juste après. Une règle de condition suffisante s'applique ensuite car acmf est un parallélogramme ayant ses demi-diagonales de même longueur. La conclusion est ainsi mise à vrai. L'autre version possible est de faire appel à une conjecture. En effet, on veut montrer que acmf est un rectangle et b = mil(a,m) il serait donc intéressant de montrer que b= mil(f,c) et donc d'appeler cette conjecture.

5. Graphe et preuve rédigée d'un théorème

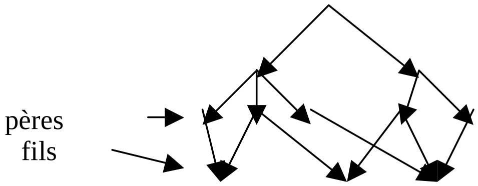
5.1. Obtention du graphe de démonstration

Tous les faits obtenus successivement par le démonstrateur sont classés par ordre chronologique et ont un numéro d'apparition. Pour chaque fait nouveau déduit on mémorise les hypothèses déjà présentes dans la base de faits qui ont permis son émergence. On peut donc remonter, à partir de chaque conclusion partielle obtenue, jusqu'aux hypothèses initiales ayant permis son émergence. L'arbre de démonstration est alors la réunion de tous les sous-arbres obtenus pour chaque élément de la conclusion globale. La mémorisation de la justification d'un fait n'est effectuée qu'une fois, même si ce fait sert à plusieurs endroits dans l'arbre. On récupère aussi les noms d'objets ayant servi dans la preuve, pour pouvoir, dans la rédaction, remplacer des noms barbares par la notation fonctionnelle du mathématicien. Si la droite (ab) a comme nom dr1 on pourra remplacer dans la trace rédigée dr1 par la notation plus interessante (ab). On fait de même pour les longueurs, les angles, etc.

5.2. Obtention de la preuve rédigée

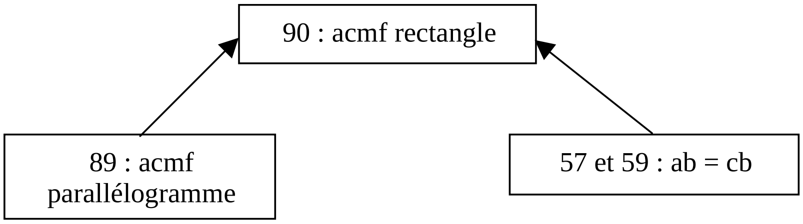
A partir du graphe le système **construit** automatiquement une preuve rédigée en français respectant la chronologie des déductions (voir **annexe 4** pour la **preuve rédigé complète**).
Voici l'algorithme correspondant :

```
Début
    - charger le graphe
        - mettre dans une liste L les hypothèses initiales
        Tant que L ≠ ∅
                - développer le premier elément de L
                - me marquer et le supprimer de L
                - placer ses « pères » dans L sauf si marqués
                - réordonner L chronologiquement
        fin Tant que
    fin
```



Exemple de **développement** : pourquoi acmf est-il un rectangle ? Ceci correspond au dernier nœud à développer chronologiquement. A partir de l'extrait de graphe suivant :

```
etiquette(90, rect(a, c, m, f)).
fils(90, 89).    fils(90, 57).    fils(90, 59).
etiquette(89, plg(a, c, m, f)).
fils(89, init / 3).   fils(89, 84).
etiquette(57, longueur(seg1) : long2).
etiquette(59, longueur(seg3) : long2).
segment(a, b) : seg1.
segment(c, b) : seg3.
```



la procédure de rédaction automatique donnera en français :
a c m f est un rectangle car a c m f est un parallélogramme et a b = c b
(en effet un parallélogramme ayant ses diagonales de même longueur est un rectangle)

Il y a possibilité de demander au système une rédaction dans un certain esprit. On sait bien qu'un même problème de géométrie peut être résolu par des outils très différents : géométrie des configurations, vectorielle, analytique, barycentrique ou autre. Certains types de problèmes, par exemple des problèmes d'alignement en seconde, peuvent être déjà retraduits en utilisant la géométrie vectorielle ou analytique. Un travail reste à faire pour que l'utilisateur puisse demander une preuve spécifiquement analytique, vectorielle ou autre et la rédaction adhoc.

5.3. Problèmes rencontrés au niveau de l'explication

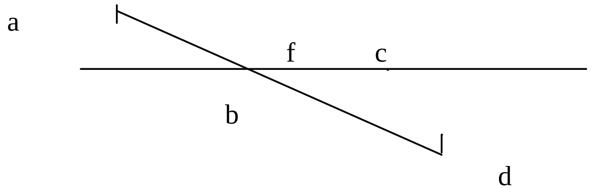
Il n'existe pas de rédaction universelle d'une preuve. Toute explication s'adresse à un public donné qui est plus ou moins compétent ou réceptif sur le sujet. Le problème central est donc de définir ce qui va être dit et ce qui ne sera que suggéré ou passé sous silence.

5.3.1. Gestion des prédicats symétriques ou de dénomination fonctionnelle multiple

On accepte que dans la preuve le parallélogramme acmf apparaisse à certains endroits sous la dénomination acmf et à d'autres sous la forme facm. Ceci ne doit pas entraîner de surcharge cognitive trop forte pour le lecteur car celui-ci a la figure sous les yeux et voit instantanémment qu'il s'agit du même objet. Une droite reçoit un nom interne à partir de deux points et ce nom sera systématiquement retranscris sous la forme droite(a,b) même si ce n'est pas vraiment avec ces deux points qu'on veut l'utiliser.

5.3.2. Gestion des implicites

Dans une rédaction type tout n'est pas justifié noir sur blanc. Il y a des implicites qui dépendent du contrat didactique passé entre le professeur et les élèves ou entre le correcteur et le candidat à un examen. Ceci dépend aussi du niveau d'étude. Une rédaction de solution d'un même exercice de géométrie ne sera pas la même en troisième, en seconde ou en terminale. Il y a des choses que l'on passera sous silence pour éviter des lourdeurs de rédaction ou parce qu'on pense qu'à un certain niveau le lecteur est capable de faire la déduction manquante par lui-même. Il y a un bel exemple de ce genre de problème dans l'exemple décrit plus haut. La règle rajoutant que $f\hat{b}m = a\hat{b}c$ car ce sont des angles opposés par le sommet est obligée de vérifier que f,b et c sont alignés dans cet ordre car sinon elle ferait déduire des bêtises au démonstrateur dans d'autres circonstances. Avec une figure comme ci-dessous, en oubliant la condition précèdente, la règle rajouterait que : $a\hat{b}c = f\hat{b}d$



Pourtant quand on donne cet exercice à un expert, celui-ci n'évoque jamais ce problème. Il le lit sur la figure et considère cette propriété comme allant de soi. Mais si on lui demande de le prouver ça n'est pas immédiat. Pour le démonstrateur le contrat didactique est simple, rien ne doit être passé sous silence.

5.3.3. Conclusion

Le but ultime est de produire des traces rédigées qui retranscrivent fidèlement l'arbre de démonstration mais avec la plus grande économie de moyens possible et en fonction du public auxquelles elles s'adressent.

6. Domaines d'étude du démonstrateur

6.1. Géométrie des configurations (collège et lycée)

C'est a priori le domaine où le démonstrateur est le plus fort. Des règles sur les parallélogrammes, triangles, milieux, cocyclicité, angles géométriques ou orientés, Thalès, Pythagore, etc ont été implémentées sans pour autant prétendre à la complétude sur le domaine.

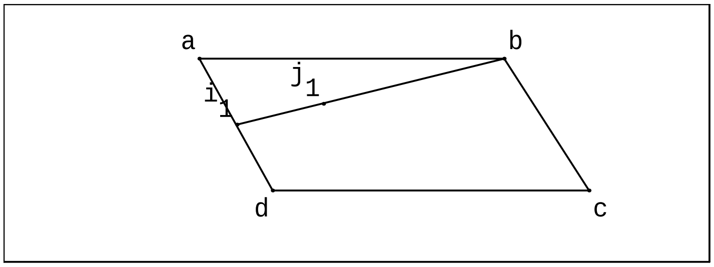
6.2. Géométrie vectorielle et analytique

Le démonstrateur est capable de résoudre des exercices niveau seconde mélangeant le calcul vectoriel avec l'utilisation des résultats sur les configurations. Voici par exemple un problème d'alignement résolu par le système.

Soit abcd un parallélogramme et i = milieu(a,d) et j un point tel que :

$\vec{ij} = \frac{1}{3}\vec{ib}$. Montrer que les points a,j,c sont alignés.

(indication on se placera dans le repère (a,b,d)) figure :

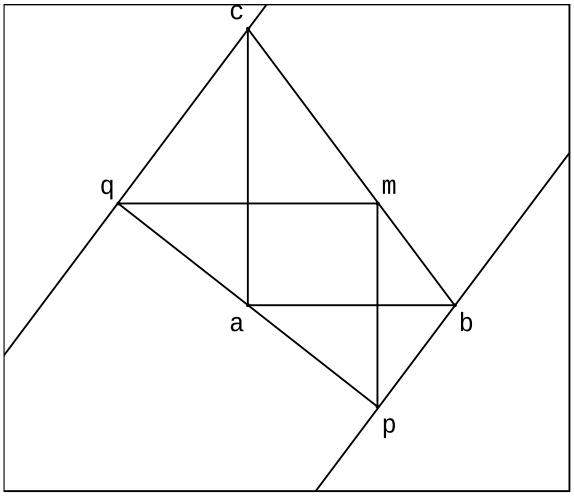


Que fait le démonstrateur ? Il va chercher à exprimer les vecteurs $\overrightarrow{ij}$ et $\overrightarrow{ib}$ en fonction des vecteurs de base $\overrightarrow{ab}$ et $\overrightarrow{ad}$. Il s'agit d'une expertise commune à la géométrie vectorielle ou analytique. Il compare alors les vecteurs pour trouver un lien de colinéarité. Il y aura eu, en option, un appel de conjecture pour vérifier que deux vecteurs fabriqués avec les trois points a,j et c sont colinéaires.

6.3. Transformations planes

Les règles classiques sur les isométries, les homothéties ont été implémentées ainsi que quelques règles de composition, par exemple sur la composée de deux reflexions d'axes sécants ou paralléles. Voici un exemple d'exercice résolu.

Soit abc un triangle rectangle en a, et m un point de la droite (bc). Soient p et q les images de m par les symétries axiales d'axes (ab) et (ac). Montrer que a = mil(pq) et que (cq) // (bp).



Voici comment l'exercice est résolu automatiquement. Le concept de symétrie axiale est reconnu donc toutes les règles liées aux transformations et reflexions sont chargées automatiquement. Une règle heuristique introduit la symétrie centrale de centre a comme composée de deux reflexions d'axes sécants en a. Cette règle dit en substance que si deux symétries axiales d'axes perpendiculaires existent pour le système alors introduire aussi leur composée qui sera alors une symétrie centrale de centre le point d'intersection des deux droites. Les règles cherchent systèmatiquement les images des points et des droites par toutes les transformations définies en sachant qu'une symétrie est involutive. Donc p $\longmapsto$ q par la composée qui est s_c donc a = mil(p,q). Une autre règle rajoute que (cq) $\longmapsto$ (bc) $\longmapsto$ (bp) par la composée qui est s_c donc (bp) // (cq) car l'image d'une droite par une symétrie centrale est une droite paralléle à la première. Ainsi toutes les conclusions partielles sont prouvées.

6.4. Utilisation du calcul formel (très réduite)

Des procédures de calculs, pouvant être appelées lors de déductions à partir de configurations particulières comme par exemple celles de Pythagore ou Thalès ou leur réciproque, ont été implémentées mais leur mise au point n'est pas terminée. Voici un exercice de fin de troisième, avec questions enchaînées, résolu par le système.

Soient a,o,f,c 4 points alignés dans cet ordre avec : ao=of=3, ac = 15. Soit b un point tel que ob=6 et (bo) ⊥ (ac). Le cercle de diamètre [fc] coupe la droite (bc) en h. g est le point d'intersection de la droite (hf) avec la paralléle à la droite (bf) passant par a.

1)Calculer les longueurs ab et bc.

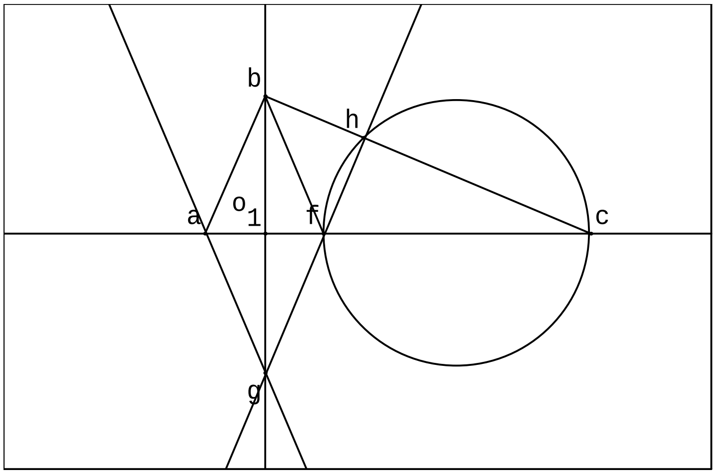
2)Montrer que (ab) ⊥ (bc).

3)Montrer que le triangle hfc est rectangle en h.

4) Montrer que (ab) // (fh).

5) Montrer que le triangle baf est isocèle.

6) Montrer que abfg est un losange.



Le démonstrateur commence par calculer ba= 3 rac(5) car aob est un triangle rectangle en o dont on connaît les valeurs numériques de deux longueurs de côté, puis oc = 12 par diffèrence entre ac et ao, car on en a besoin pour calculer bc ds le triangle rectangle en o obc, puis il calcule bc par la propriété de pythagore et trouve bc = 6 $\sqrt{5}$. Ensuite il découvre que bac est rectangle en b d'après la rèciproque du théorème de pythagore. Il déduit que o = m(a,f) car oa = of = 3 et que les points sont alignés. (bo) = med [af] comme droite passant par le milieu et orthogonale à (af), donc bf = ab = 3rac(5). Le triangle baf est ainsi isocèle. hfc est alors rectangle en h comme triangle inscrit dans un cercle dont un côté est un diamètre. (fh) et (ab) st paralléles car perpendiculaires à la même droite (bc). abfg est un parallélogramme car ses côtés st paralléles deux à deux. Il devient alors un losange car en plus ab = bf.

7. Problèmes rencontrés en Démonstration Automatique de Théorèmes

7.1. Quels objets introduire au départ ?

Il faut effectuer un savant dosage à partir de chaque configuration en hypothèse pour ne pas introduire trop d'objets mais quand même ceux qu'il faut. Par exemple avec le triangle équilatéral que doit-on créer ? Après discussion auprès d'experts du domaine on indique que les trois côtés et les trois angles au sommet ont même mesure. En fait quand l'expert code la figure il est déjà en train de déduire. Un moyen d'implémenter cette démarche est de faire des appels de conjecture permettant de créer des objets en conjonction avec les autres configurations présentes.

7.2. Comment charger les bonnes règles ?

Il faut implémenter la démarche de l'expert c'est-à-dire l'extraction des configurations prégnantes qui sautent aux yeux de l'expert. Il faut mieux cerner l'implication d'une règle en fonction de la conjonction de concepts présents. En quoi par exemple la présence des deux concepts équilatéral et orthogonalité va t'elle influer sur le chargement des règles ? La typologie de règles devrait prendre en compte la conjonction de plusieurs concepts. L'ensemble des règles pourrait être structuré selon un modèle hiérarchique. Mais peut-on réellement fabriquer un tel modèle qui implémenterait vraiment la démarche de l'expert ?

7.3. Ordonner les règles dynamiquement ?

Comment l'expert sent-il qu'il est sur une mauvaise piste ? Ce n'est pas qu'une question de temps passé trop long dans une certaine direction qui oriente l'expert dans une autre direction, comme me le disait un collègue. Comment lui vient-il une nouvelle idée ? La possibilité d'ordonner les règles dynamiquement va influer sur l'efficacité du démonstrateur. Certaines règles importantes pour la démonstration, placées loin, sont obligées d'attendre que toutes les règles précédentes ne puissent plus s'appliquer pour entrer en action. Un moyen de régler ce problème est la possibilité d'appeler des conjectures à partir de règles vérifiant la présence de conditions particulières favorables en fonction du but (Dans l'exemple détaillé appeler la conjecture b = mil(fc) (voir section 4.1)).

7.4. Création d'objets en cours de démonstration

La création d'objets pour le mathématicien n'a pas le même coût que pour le système. Les règles de création sont placées à la fin et sont des jokers si le démonstrateur ne déduit plus rien (problèmes d'explosion combinatoire). Première idée : On crée un objet, on désactive les règles de création, on regarde ce qui se passe; si on a avancé vers un des buts on conserve l'objet sinon on le supprime et on recommence. On peut aussi appeller des conjectures avec suppression des objets créés si échec.

7.5. Repérer les éléments clés d'une démonstration

Dans l'exemple, le verrou à faire tomber est de prouver que b = mil(fc). Comment un système informatique pourrait-il repèrer ces nœuds de difficultés pour se focaliser sur eux et faire de bons appels de conjecture ? Ceci pourrait aussi aider pour la preuve rédigée, et ne pas placer tous les nœuds de développement sur le même plan.

7.6. Gestion de la base de connaissances :

Comment trier les connaissances et ne donner que celles qui vont servir ? Si l'on veut un système utilisable en EIAO il faut qu'un mathématicien ou un professeur de mathématiques puisse de façon simple, sans surcharge cognitive, fournir les connaissances au démonstrateur de façon déclarative et dans une syntaxe proche du langage mathématique. Ceci suppose qu'ensuite le système construise de façon automatique un ensemble de règles traduisant de façon opérationnelle le savoir proposé. Ce problème est abordé dans la suite de l'exposé.

7.7. Interactions homme/machine

Pour une utilisation en EIAO il faut aussi envisager une interface conviviale, qui permette au professeur de donner les connaissances au système sans efforts particuliers, et à l'élève de poser ses problèmes, confirmer ses idées, explorer les différentes déductions possibles à partir de l'énoncé donné et avoir une explication sur tout fait déductible. Il faudrait, en particulier, que, à partir d'un énoncé livresque, le système aide l'élève à extraire les hypothèses données, et lui permette d'écrire l'énoncé sans avoir un certain langage d'expression des connaissances à apprendre. Un système de menus déroulants, avec boîtes de dialogue, et messages d'erreurs ou de confirmation devra lui permettre d'introduire les objets en n'ayant à sa charge que l'instanciation concrète de ceux-ci.

8. Construction automatique des règles

8.1. Idée de base

L'idée est d'exprimer les connaissances de façon **déclarative** dans une syntaxe **proche du langage mathématique.** A partir d'une procédure déjà présente dans Muscadet, il faut en construire une nouvelle qui soit capable d'opérationnaliser le savoir déclaratif relatif au domaine de la géométrie. Actuellement les connaissances sont données au système sous forme de prédicats dont voici quelques exemples.

```
 propriete( centreGravite(A,B,C):G <=>
      G estDans droite(A,milieu(B,C)) inter droite(B,milieu(A,C)) ).

definition( milieu(A,B):I <=> I app droite(A,B) et
```

(longueur(A,I) = longueur(I,B))).
propriete(droiteMilieux,milieu(A,B):I et milieu(A,C):J =>
droite(I,J) paralleles droite(B,C)).

8.2. Principe

Il s'agit de faire construire, par le système, de façon automatique des règles de différents types à partir des connaissances données.

8.3. Démarche

8.3.1. Donner un nom à la règle

Donner un nom à la règle en fonction du type voulu et des concepts présents. Ceci permettra de typer la règle. On pourra construire des règles de type déduction tout venant, déduction en fonction du but ou de création, et des règles de condition suffisante. A partir du concept centreGravite on fabriquera des règles de nom centreGraviteDeduction qui déduiront des propriétés à partir d'un centre de gravité existant et d'autres de nom centreGraviteDeduit qui, si les conditions sont réunies dans la base de faits, rajouteront que tel point est un centre de gravité.

8.3.2. Constituer le corps de la règle

- Il faut que la règle vérifie la présence des objets nécessaires.
- Il faut constituer la partie condition nécessaire à l'application de la règle.
- Il faut introduire un test de non répétition de la règle, puis constituer la partie action et mémorisation.

8.3.3. Optimisation de l'ordre des sous-buts et mise au point finale

Pour qu'une règle puisse s'appliquer de façon optimale, l'ordre des sous-buts est très important. Il faut par exemple placer les configurations rares ou spécifiques en premier pour instancier le plus tôt possible la règle avec les bonnes variables. D'autre part il faut qu'une variable soit instanciée avant que l'on puisse l'utiliser. Sinon on court à la catastrophe. Il faut aussi maitriser le retour arrière. Pour pouvoir réaliser ceci il a fallu et il faut faire appel à une méta-expertise sur la construction des règles. Ceci a nécessité une auto-analyse sur la façon dont on été écrites les règles à la main pour mettre au point ensuite la procédure de construction automatique et d'ordonnancement des sous-buts.

8.3.4. Exemples de règles construites

La définition du milieu d'un segment ci-dessous :
definition(milieu(A,B):I <=> I app droite(A,B) et (longueur(A,I) = longueur(I,B))).
donnera les règles :

```
regle(A, milieuDeduction, B) :-
milieu(A, C, D, E, F),
 droite(A, C, D, G, H),
  not hyp(A, E app G, I),
      ajhyp(A, E app G, J),  memoriser([F], J).
```

Cette règle signifie que si E est le milieu de [CD], si la droite (CD) a pour nom G et si on ne sait pas encore que E est sur la droite G alors rajouter ce fait sous le numéro J et mémoriser que ce fait numéro J provient du fait numéro F.

```
regle(A, milieuDeduction1, B) :-
      milieu(A, C, D, E, F),
droite(A, C, D, G, H),
      longueur(A, C, E, I, J),
segment(A, E, D, K, L),
      not longueur(M, E, D, I, N),
      ajhyp(A, longueur(K) : I, O), memoriser([F, J, L], O).
```

Signification : si E est le milieu de [CD], si la droite (CD) a pour nom G, si I est la longueur CE , si le segment [ED] a été créé et si on ne sait pas que la longueur ED est égale à CE alors rajouter ce fait et mémoiser que ce fait numéro O provient des faits de numéros F,J et L. On aura des traductions du même type pour chacune des règles en exemple ci-après.

```
regle(A, milieuDeduit, B) :-
      longueur(A, C, D, E, F),
 droite(A, C, G, H, I),
 hyp(A, D app H, J),
      longueur(A, D, G, E, K),
  not milieu(A, C, G, D, L),
      ajhyp(A, milieu(C, G) : D, M), memoriser([F, J, K], M).
```

La propriété :

```
propriete( droiteMilieux,milieu(A,B):I et milieu(A,C):J =>
droite(I,J) paralleles droite(B,C) ).
```

donnera :

```
regle(A, droiteMilieuxDeduction, B) :-
      milieu(A, C, D, E, F),
milieu(A, C, G, H, I),
      droite(A, E, H, J, K),
droite(A, D, G, L, M),    distincts(J, L),
      not paralleles(A, J, L, N),
      ajhyp(A, J paralleles L, O),   memoriser([F, I], O).
```

```
regle(A, droiteMilieuxDeductionCreation, B) :-
      milieu(A, C, D, E, F),
      milieu(A, C, G, H, I),
      creerDroite(A, E, H, J, K),
      creerDroite(A, D, G, L, M),
      distincts(J, L),   not paralleles(A, J, L, N),
      ajhyp(A, J paralleles L, O),   memoriser([F, I, K, M], O).
```

9. Appels de conjectures

Le système a la possibilité d’appeler des conjectures, guidées par le but en cas de configurations incomplètes permettant d ‘avancer vers une des conclusions partielles à prouver. Un exemple de règle appelant une conjecture se trouve dans la section 8.3.

9.1. Que se passe t’il en cas d’appel ?

De nouveaux concepts sont introduits en fonction de ce que demande la conjecture. Il y a chargement de nouvelles règles et réordonnancement car le système reconstruit une nouvelle liste de règles actives**.** Pour l’instant tout fait déduit est conservé par la suite. Si la conclusion finale est prouvée en milieu de conjecture alors le système sort et la démonstration est terminée. Une conjecture prouvée apparaît comme un nouveau fait déduit et on a mémorisé le sous-arbre ayant permis son émergence. Si il y a échec lors d’un appel, on reprend le fonctionnement global avec éventuellement appels d’autres conjectures.

9.2. Problèmes

Un appel de conjecture est une démarche typiquement heuristique. Cet appel doit être fait au bon moment. Trop tôt et c’est l’echec, trop tard ça ne sert plus à rien. Il ne faut pas que le démonstrateur s’égare. Peut-être faut-il supprimer les règles de création d’objet lors d’un appel de conjectures. Un mathématicien ne fait pas un appel de conjecture par hasard. Dès le codage de la figure il en émet. Il fonctionne par exemple par analogie. Il a déjà rencontré une situation similaire et il essaye de voir si, sur le problème du moment, cela va fonctionner de la même façon. C’est en fonction de son expérience qu’il fera tel ou tel choix. Comment acquerir la **« certitude »** de celui qui **« voit »** sur la figure et à qui il ne manque que le cheminement déductif?

9.3. Exemple de règle appelant une conjecture :

```
regle(N,conjecture,_) :- appelConjecture(non),
%si on veut montrer que ABCD est un parallélogramme,rectangle,carré ou losange
            (eltConcl(N,plg(A,B,C,D))
            ;eltConcl(N,rect(A,B,C,D))
            ;eltConcl(N,losange(A,B,C,D))
            ;eltConcl(N,carre(A,B,C,D))
            ),
%si le milieu d’une des diagonales existe déjà et qu’un appel de conjecture pour montrer que c’est le milieu
de l’autre n’a pas déjà été lancé
                   (milieu(N,A,C,I,_), not milieu(N,B,D,I,_),
                   not conj_traite(N,milieu(B,D):I),
                   assert(conj_traite(N,milieu(B,D):I)),
              X = B, Y = D
                    ; milieu(N,B,D,I,_), not milieu(N,A,C,I,_),
                   not conj_traite(N,milieu(A,C):I),
              assert(conj_traite(N,milieu(A,C):I)),
              X = A, Y = C
                   ),
%ecrire ce message sur le brouillon de recherche
ecrire1([comme,I,'est milieu d''une des diagonales montrons que’ ,I,=,mil,'(',X,Y,') pour montrer
que’,A,B,C,D,’est un parallélogramme’]),
%appeler la conjecture
conj1(N,milieu(X,Y):I,_).
```

Explication en français :
Si on veut montrer que abcd est un parallélogramme, un rectangle, un losange ou un carré et que le milieu d’une des diagonales existe alors essayer de montrer que c’est aussi le milieu de l’autre diagonale.
On peut envisager la possibilité de construction automatique d’une telle règle à partir d’énoncés déclaratifs du type ci-dessus. Un utilisateur mathématicien pourrait ainsi exprimer des heuristiques qui pourraient être transformées en règles automatiquement par le système.

10. Bilan partiel et perspectives

10.1. Bilan

Dans une première version la base de règles a été écrite à la main, testée et mise au point sur des exercices niveau lycée. Le démonstrateur démontre environ 200 théorèmes, en géométrie des configurations, niveau lycée (parallélogrammes, triangles, milieux, cocyclicité, angles géométriques ou orientés,Thalès, Pythagore etc), en calcul vectoriel, géométrie analytique et transformations planes. Les exemples proviennent pour la plupart de livres de secondes [Terracher Hachette Secondes] par exemple, ou ont été proposés, pour certains, par Philippe Jalain, collègue de mathématique que je remercie vivement pour ses remarques pertinentes et pour l’aide qu’il m’a apporté dans la conception de la base de connaissances. Les temps de démonstrations sont inégaux suivant le type d'exercices et bien entendu suivant la façon dont sont ordonnées les règles actives. Pour fixer les idées, la plupart sont obtenues en moins de 20s sur un PC 100 Mz. Le démonstrateur démontre tous les théorèmes figurant dans les thèses [Pintado 94] et [Bazin 93], mais en prenant plus de temps en général pour ceux de [Pintado 94]. Une deuxième version est en cours avec construction automatique des règles à partir de connaissances mathématiques données de façon déclarative.

10.2. Objectifs

Il s’agit de rendre le démonstrateur suffisamment robuste pour qu'il puisse servir en EIAO.
Il apparaît nécessaire de créer une interface utilisateur sur poste fixe ou par internet pour pouvoir vraiment tester cette robustesse et vérifier l’intêret pédagogique que peut apporter un tel système. Il devrait être possible de donner plusieurs preuves de différents types sur un même exercice, en relançant par exemple la démonstration après une réussite en enlevant de la liste des règles actives la règle ayant permis de conclure

sur le but. On devrait pouvoir guider, pas à pas, la recherche d'un éléve en partant du but car à partir de la conclusion on peut développer la preuve niveau par niveau. Il est aussi possible, pour l'élève d'émettre des conjectures que le système pourra être chargé de démontrer. Ceci devrait permettre d'aiguiser le démarche heuristique de l'élève qui abandonne souvent ses idées en cours de route par manque de confiance en lui. On peut imaginer la possibilité de faire construire la figure automatiquement par un couplage avec des logiciels comme Géosplan ou Cabri Géomètre par exemple. Il faut améliorer la construction automatique des règles à partir d'énoncés déclaratifs et pouvoir tester le démonstrateur avec plusieurs utilisateurs mathématiciens. Il faut implémenter des méthodes d'apprentissage correspondant au savoir-faire de l'expert qui reconnaît dans l'expérience présente des situations antérieures déjà rencontrées. Dans une situation analogue on mettra en œuvre des techniques semblables. Ceci devrait permettre une meilleure sélection des règles et des liens entre concepts et règles applicables.

11. Références Bibliographiques

[Bazin 93] Bazin J.M. : *GEOMUS: un résolveur de problèmes de géométrie qui mobilise ses connaissances en fonction du problème posé* **-** thèse université Paris 6, 1993.

[Pastre 84] Pastre D. : *MUSCADET : Un système de démonstration automatique de théorèmes utilisant connaissances et métaconnaissances en mathématique* **-** Thèse d'état Paris VI, 1984.

[Pastre 89] Pastre D. : *MUSCADET : an automatic theorem proving system using knowledge and metaknowledge in mathematics* **-** Journal of Artificial Intelligence 38, 257-318, 1989.

[Pastre 93] Pastre D. : *Automated Theorem Proving in Mathematics* **-** Annals of Mathematics and Artificial Intelligence, Volume 8, No. 3-4, 425-447, 1993.

[Pastre 99] Pastre D. : *Le nouveau MUSCADET et la TPTP Problem Library* **-** Actes de Berder, 1999.

[Pintado 94] Pintado G. : *Apprentissage et démonstration automatique de théorèmes* **-** thèse Paris 6, 1994.

ANNEXE 1 : Base de faits à l’issue de la preuve (résumée)

Hypothèses

1) **Introduction des hypothèses :**
dr = droite(b,c)
(introDroite)
b = milieu(a, m) et m = milieu(b,d)
(introMilieu)
f = projeteOrthog(d, dr)
(introProjeteOrthog)
abc est équilatéral
(introEquilateral)

2) Introduction d’objets à partir de la conclusion :
Création de acmf et des longueurs des côtés
(introConcl)

3) Début des déductions :
mf = mb (trRectDeductionLong)
a,b,m et d alignés dans cet ordre
(alignesOrdDeduitMilieu)
$a\hat{b}c$ est aigu (angleAiguDeduit)
f,b et c alignés dans cet ordre
(appDemiDroiteDeduit)
$m\hat{b}f$ = 60 ° (oppSommetDeduitCreation3)
mbf est équilatéral
(equilateralDeduitAngle1)
bf = bm (equilateralDeductionCreation1)
b = milieu(f,c) (cs_milieu)
Rmq :un élément de la conclusion est supprimé
acmf est un parallélogramme
(plgDeduit)
acmf est un rectangle
(cs_rectangle)
La conclusion est mise à vrai

Conclusion(s)

concl(0,rect(a,c,m,f))
eltConcl(0, milieu(f,c):b)
eltConcl(0, rect(a,c,m,f))

concl(0,vrai)

Fin des déductions

ANNEXE 2 : Déductions par application de règles et procédé de mémorisation (exemples de règles) :

Ex : avec la règle trRectDeductionLong

```
%Si I est le milieu de l'hypothénuse [BC] du triangle rectangle en A ABC et que
les longueurs L et L1 des segments [BI] et [AI] ont été crées et ne sont pas
indiquées comme de même longueur alors rajouter que L et L1 ont la même
longueur(cad BI = AI) et mémoriser que le fait no Q1 provient de l'application
de la règle trRectDeductionLong et des faits déjà présents de no Z1,Z2 et Z3
regle(N,trRectDeductionLong ,_) :-
trRect(N,A,B,C,Z1),
        milieu(N,B,C,I,Z2),
segment(N,B,I,L,Mes,Z3),
        segment(N, A,I,L1,_,_),
 not egalLong(N,L,L1,_),
ajhyp(N,longueur(L1):Mes,Q1),      memoriser([trRectDeductionLong,Z1,Z2,Z3],Q1),
ecrire1([Q1,'d''apres',Z1,Z2,Z3]).
```

Déduction proprement dite: (fd) ⊥ (bc) et m = mil(b,d) donc mf = mb.

Mémorisation: on mémorise le nom de la règle en premier pour pouvoir écrire un commentaire explicatif, puis les numéros des hypothèses permettant le rajout du fait que [mf] et [mb] ont même longueur.

Sortie du système: (exemple)

La règle cs_rect vérifie que acmf est un parallélogramme ayant ses diagonales de même longueur donc elle met la conclusion à vrai car il n'y a qu'une chose à montrer et le théorème est donc prouvé.

```
regle(N,cs_rect,_) :-
eltConcl(N,rect(A,B,C,D)),
plg(N,A,B,C,D,Z6),
milieu(N,A,C,I,_),
( (segment(N,A,I,L,_), segment(N,B,I,L1,_))
; (segment(N,A,C,L,_), segment(N,B,D,L1,_))
),
egalLong(N,L,L1,Z5),
ajhyp(N,rect(A,B,C,D)  ,Q1),memoriser([Z6,Z5],Q1) ,
conclFinale(N,Concl),              remplacer(Concl,rect(A,B,C,D),vrai,Concl1),
nouvconcl(N,Concl1),
ajhyp(N,fin,Q), memoriser([Q1],Q),  retract(eltConcl(N,rect(A,B,C,D))),
 ecrire1([A,B,C,D,'est un rectangle car parallélogramme et deux diag de même
longueur d apres',Z5,Z6 ]).
```

ANNEXE 3 : Fichier contenant la preuve rédigée du théorème: th210

ENONCE donné au système:
alignesOrd([a, o, f, c]) et cercleDiam(f, c) : cercle et h estDans cercle inter droite(b, c) et droite(b, o) orthog droite(a, c) et longueur(a, c) egal 15 et longueur(a, o) egal 3 et longueur(f, o) egal 3 et longueur(b, o) egal 6 et droite(a, g) paralleles droite(b, f) et g app droite(f, h) => calculer(longueur(a, b)) et calculer(longueur(b, c)) et droite(a, b) orthog droite(b, c) et trRect(h, f, c) et droite(a, b) paralleles droite(f, h) et isocele(b, a, f) et losange(a, b, f, g)

HYPOTHESES DE DEPART:
g est sur la droite (f h) :hypothèse initiale
a c = 15 :hypothèse initiale
a o = 3 :hypothèse initiale
f o = 3 :hypothèse initiale
b o = 6 :hypothèse initiale
alignesOrd([a, o, f, c]) :hypothèse initiale
cercle est le cercle de diamètre [f c] :hypothèse initiale
les droites (a g) et (b f) sont parallèles :hypothèse initiale
les droites (b o) et (a o) sont perpendiculaires : hypothèse initiale
h est à l'intersection des lieux de noms: cercle et dr : hypothèse initiale

CONCLUSION(S):
calculer(longueur(a, b))
calculer(longueur(b, c))
droite(a, b) orthog droite(b, c)
h f c est un triangle rectangle en h
droite(a, b) et droite(f, h) sont des droites parallèles
b a f est un triangle isocèle en b
a b f g est un losange

DEMONSTRATION:

h est sur le cercle de diamètre [f c] car h est à l'intersection des lieux de noms: cercle et dr

a b ^2 = a o ^2 + b o ^2 car les droites (b o) et (a o) sont perpendiculaires

a b = 3 * rac(5) car a b ^2 = a o ^2 + b o ^2 et a o = 3 et b o = 6

o c = 12 car a o = 3 et a c = 15 et alignesOrd([a, o, f, c])

b c ^2 = o c ^2 + b o ^2 car les droites (b o) et (a o) sont perpendiculaires

b c = 6 * rac(5) car b c ^2 = o c ^2 + b o ^2 et o c = 12 et b o = 6

h f c est un triangle rectangle en h car h est sur le cercle de diamètre [f c]

les droites (f h) et (b c) sont perpendiculaires car h est sur le cercle de diamètre [f c]

a c ^2 = 225 car a c = 15

b c ^2 + a b ^2 = 225 / 1 car b c = 6 * rac(5) et a b = 3 * rac(5)

a c ^2 = a b ^2 + b c ^2 car a c ^2 = 225 et b c ^2 + a b ^2 = 225 / 1

b a c est un triangle rectangle en b car a c ^2 = a b ^2 + b c ^2

les droites (a b) et (b c) sont perpendiculaires car b a c est un triangle rectangle en b les droites (f h) et (a b) sont parallèles car les droites (f h) et (b c) sont perpendiculaires et les droites (a b) et (b c) sont perpendiculaires

a g f b est un parallélogramme car les droites (a g) et (b f) sont parallèles et les droites (f h) et (a b) sont parallèles

a g = b f car a g f b est un parallélogramme

b f ^2 = f o ^2 + b o ^2 car les droites (b o) et (a o) sont perpendiculaires

b f = 3 * rac(5) car b f ^2 = f o ^2 + b o ^2 et f o = 3 et b o = 6

(3 * rac(5)) egalLong long9 car b f = 3 * rac(5) et a g = b f

a g = a b car (3 * rac(5)) egalLong long9 et a b = 3 * rac(5)

b f = a b car a g = b f et a g = a b

b a f est un triangle isocèle en b car b f = a b

a g f b est un losange car a g f b est un parallélogramme et a g = a b

ANNEXE 4 : Fichier contenant la preuve rédigée du théorème: th118

ENONCE donné au système:
equilateral(a, b, c) et milieu(a, m) : b et milieu(b, d) : m et projeteOrthog(d, droite(b, c)) : f => rect(a, c, m, f)

HYPOTHESES DE DEPART:
b est le milieu du segment [a m] :hypothèse initiale
m est le milieu du segment [b d] :hypothèse initiale
f est le projeté orthogonal de d sur la droite (b c) :hypothèse initiale
a b c est un triangle équilateral :hypothèse initiale

CONCLUSION(S):
a c m f est un rectangle

DEMONSTRATION:

m b = a b car b est le milieu du segment [a m]

f est sur la droite (b c) car f est le projeté orthogonal de d sur la droite (b c)

les droites (f d) et (b c) sont perpendiculaires car f est le projeté orthogonal de d sur la droite (b c)

a b = c b car a b c est un triangle équilateral

l'angle géométrique a b c mesure 60 degrés car a b c est un triangle équilateral

les points a , b , m et d sont alignés dans cet ordre sur la droite (a m) car b est le milieu du segment [a m] et m est le milieu du segment [b d]

les points d , m , b et a sont alignés dans cet ordre sur la droite (a m) car m est le milieu du segment [b d] et b est le milieu du segment [a m]

les points f , b , c sont alignés dans cet ordre sur la droite (b c) car les points d , m , b et a sont alignés dans cet ordre sur la droite (a m) et l'angle géométrique a b c est aigu et f est le projeté orthogonal de d sur la droite (b c)

m b = m f car les droites (f d) et (b c) sont perpendiculaires et m est le milieu du segment [b d]

l'angle géométrique m b f mesure 60 degrés car les points a , b , m et d sont alignés dans cet ordre sur la droite (a m) et les points f , b , c sont alignés dans cet ordre sur la droite (b c) et l'angle géométrique a b c mesure 60 degrés

m b f est un triangle équilateral car m b = m f et l'angle géométrique m b f mesure 60 degrés

m b = b f car m b f est un triangle équilateral

a b = b f car m b = a b et m b = b f

c b = b f car a b = c b et a b = b f

b est le milieu du segment [c f] car c b = b f et f est sur la droite (b c)

a c m f est un parallélogramme car b est le milieu du segment [a m] et b est le milieu du segment [c f]

a c m f est un rectangle car a c m f est un parallélogramme et a b = c b

Spécification de dialogues et construction d'interfaces modulaires et réutilisables

Groupe COMBIEN
Duma J.[1] Giroire H.[2], Le Calvez F.[3], Tisseau G.[2], Urtasun M.[3]

Résumé : L'objectif du groupe Combien est d'étudier les problèmes que posent la constuction d'un EIAO. De nombreuses interfaces complexes sont à concevoir et réaliser pour contrôler l'activité de l'élève. Comme elles présentent de plus des parties similaires, nous avons réfléchi plus particulièrement au problème de la réutilisation de "morceaux" d'interfaces. Nous décrivons ici les outils généraux permettant la *spécification*, la *réalisation* et la *réutilisation* d'interfaces, que nous avons conçus et réalisés. A partir d'une interface de saisie d'une fiche d'identité, nous introduisons le formalisme IREC de spécification du dialogue d'une interface. Puis nous montrons comment l'environnement de développement d'interfaces EDIREC permet d'éditer cette spécification. A ce propos nous exposons les fonctionnalités d'EDIREC. Celui-ci transforme en code exécutable les spécifications et fournit des aides à la mise au point et au test de composants de l'interface. EDIREC est intégré à VisualWorks et il a été produit à partir de lui-même par amorçage (bootstrap).

Mots-clés : environnement de développement d'IHM, spécification du dialogue, réutilisation de composants, EIAO.

1. Introduction

Le groupe Combien? travaille depuis plusieurs années sur les différents éléments d'un EIAO dans le domaine du dénombrement. Nous avons mené de front des réflexions théoriques et des expériences d'implémentation. A partir de notre expérience dans l'enseignement des dénombrements en classe, nous avons défini les fondements mathématiques d'une méthode de résolution : la "méthode constructive" adaptée aux conceptions usuelles des élèves et permettant d'accéder à la théorie mathématique du domaine [Le Calvez et al. 97] [Tisseau et al. 96]. Un des buts du système est d'enseigner cette méthode aux élèves.

Un autre but est de faire prendre conscience aux élèves de l'existence de classes de problèmes liées à la résolution. Pour cela nous proposons à l'élève une série d'interfaces, les "machines à construire" [Le Calvez et al. 97], spécifiques à chaque classe leur permettant de modéliser un problème de cette classe et de le résoudre suivant la méthode constructive que nous avons définie. Ces interfaces contrôlent l'activité de l'élève, elles sont donc complexes. Elles présentent de plus des parties similaires. Le nombre de classes de problèmes étant important, nous avons été amenés à réfléchir plus particulièrement au problème de la réutilisation de "morceaux" d'interfaces.

Nous avons alors réalisé des outils généraux permettant la *spécification*, la *réalisation* et la *réutilisation* d'interfaces.

Ce papier présente l'utilisation de ces outils à travers des exemples. Il reprend le plan de la présentation faite au colloque métaconnaissances de Berder. Cette présentation comportait la création en direct d'interfaces au moyen de l'environnement de développement d'interfaces EDIREC, qui ne peut pas être transcrite ici. Nous avons essayé, grâce aux figures, d'en donner une approche la plus fidèle possible.

A partir d'une interface de saisie d'une fiche d'identité, nous introduisons le formalisme IREC de spécification du dialogue d'une interface. Puis nous montrons comment EDIREC permet d'éditer cette spécification. A ce propos nous exposons les fonctionnalités d'EDIREC. Celui-ci transforme aussi en code exécutable les spécifications et fournit des aides à la mise au point et au test de composants de l'interface. EDIREC est intégré à VisualWorks et il a été produit à partir de lui-même par amorçage (bootstrap).

---

[1] Lycée Jacquard, Paris.
[2] LIP6 PôleIA, SYSDEF, Université Pierre et Marie Curie.
[3] CRIP5, Université René Descartes.

2. Les systèmes interactifs

L'importance croissante des interactions entre des utilisateurs non informaticiens et les ordinateurs a provoqué l'émergence d'une nouvelle branche de l'informatique : les IHM. De nombreuses architectures pour la conception et la réalisation d'interfaces ont été proposées. Elles prennent en compte le domaine, le dialogue et la présentation. Le *domaine* est le noyau fonctionnel du système, qui dans le projet Combien? est formé des objets et connaissances que nous avons définis pour le dénombrement. Le *dialogue* contrôle la dynamique des échanges entre l'utilisateur et le système. La *présentation* concerne la partie perceptible par l'utilisateur. Elle se compose des widgets qui permettent d'une part de présenter les données à l'utilisateur et d'autre part de donner à l'utilisateur les moyens d'agir sur le système.

De nombreux outils commerciaux, souvent désignés de manière générique sous le terme de Systèmes de Gestion d'Interfaces Utilisateurs (SGIU ou UIMS pour User Interface Management Systems), sont proposés actuellement pour faciliter la réalisation de systèmes interactifs [Myers 95]. La plupart d'entre eux concernent essentiellement la partie «présentation» du système : ils évitent au programmeur d'avoir à utiliser directement les primitives d'une boîte à outils ou d'un système de fenêtrage. Leur mise en œuvre est facilitée par l'existence d'éditeurs graphiques permettant de dessiner directement l'interface.
Ces outils rendent effectivement abordable la réalisation de la présentation d'un système interactif à des programmeurs non spécialistes des primitives graphiques et événementielles d'un environnement particulier, et il est avantageux de les utiliser. Cependant, ils laissent la gestion des autres aspects de l'interface à la charge du programmeur, en particulier la spécification et la réalisation du dialogue, comme par exemple la détermination des commandes qui doivent être rendues disponibles ou indisponibles suivant l'état du système. La plupart du temps, le programmeur doit incorporer ces aspects aux procédures de réaction aux événements et les mélanger ainsi à des appels au noyau fonctionnel du domaine. Or la gestion du dialogue est un aspect essentiel d'une interface pédagogique et elle doit pouvoir être traitée séparément, avec des outils appropriés permettant de spécifier le dialogue et d'engendrer une version exécutable de cette spécification. De nombreux travaux de recherche ont traité le problème du dialogue, en particulier pour définir des formalismes de spécification [Dix 91] [Palanque et al. 97]. La plupart de ces formalismes restent théoriques, mais certains d'entre eux sont exécutables [Bastide et al. 95] [Browne et al. 97]. Cependant, pour le moment, peu d'entre eux ont débouché sur des outils intégrés de génération d'interfaces accessibles sur différentes plates-formes de développement. C'est pour cette raison que nous avons été amenés à réaliser un outil de développement d'interfaces adapté à notre environnement de travail et à nos objectifs, dans le cadre d'un projet de système interactif d'aide à l'apprentissage humain.
Dans cet article les différents éléments du formalisme seront introduits à partir d'un exemple, ce formalisme est plus précisément décrit dans [Tisseau et al. 99].

3. Formalisme de spécification du dialogue

Notre formalisme est inspiré des réseaux de Petri à objets. Ce type de représentation nous permet de prendre en compte les données et leur circulation au même titre que les actions et leur séquencement.

3.1. Interacteur

L'unité de structuration du formalisme est appelé un *interacteur*. Il modélise le comportement d'une fenêtre d'interface ou d'une sous-fenêtre. Pour décrire ce comportement il faut préciser à tout moment quelles actions sont applicables à quels objets et quel est le résultat de l'application de l'action. Dans le formalisme IREC les entités responsables de l'exécution des actions s'appellent des *commandes* ; les entités destinées à contenir des objets s'appellent des *variables*.

Les variables et les commandes d'un interacteur sont structurées en graphe biparti. Nous appelons ce graphe le *réseau de contrôle* de l'interacteur (cf. Figure 1).

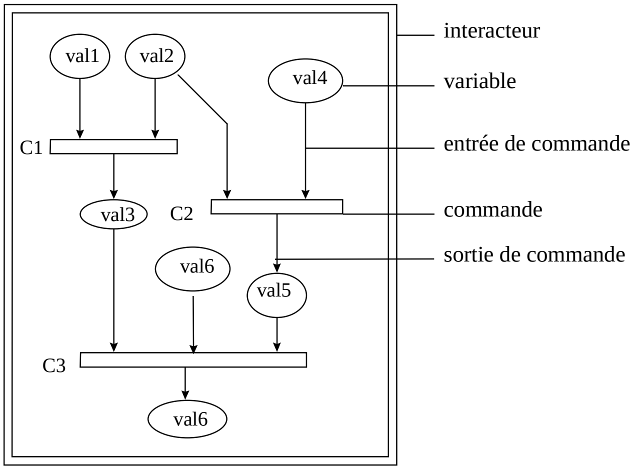


Figure 1 : *un exemple d'interacteur*

Dans un tel réseau les commandes seront déclenchables seulement lorsque leurs variables d'entrée acquerront une valeur. Dans l'exemple ci-dessus les commandes ont plusieurs variables d'entrée et une variable de sortie. C1 ne pourra être déclenchée que lorsque ses variables d'entrée auront des valeurs (ici val1 et val2) ; l'application de l'action de la commande C1 affecte à sa variable de sortie une valeur (ici val3). Cela se passe de la même façon pour les commandes C2 et C3.

3.2. Variable

Une variable peut contenir à tout moment une valeur, qui est un objet quelconque, ou aucune. C'est une restriction et une simplification par rapport aux réseaux de Petri à objets généraux, dont les places peuvent contenir plusieurs jetons étant eux-mêmes des n-uplets d'objets. Cette simplification permet de faire jouer un rôle important à deux sortes d'événements : la *validation* d'une variable (lorsqu'une variable sans valeur acquiert une valeur) et son *invalidation* (lorsqu'une variable qui a une valeur perd cette valeur). Pour une variable donnée d'un interacteur donné, ces événements peuvent avoir seulement trois causes : 1) l'utilisateur peut affecter une valeur à la variable par l'intermédiaire d'un widget, 2) l'interacteur lui-même peut affecter une valeur à la variable comme résultat de son fonctionnement interne et 3) un autre interacteur peut affecter une valeur à la variable (ce qui n'est possible que si celle-ci est déclarée être *partagée* par les deux interacteurs, ce qui sera expliqué plus loin).

3.3. Commande

A une commande sont associées des variables d'entrée, éventuellement un déclencheur, deux préconditions d'autorisation et d'acceptation, une action et la plupart du temps des variables de sortie.

La commande devient disponible si toutes ses variables d'entrée sont valides et si la *précondition d'autorisation* (par exemple : la valeur d'une variable d'entrée est du bon type) est vraie. Lorsqu'une commande est disponible, si un widget (exemple champ de texte ou bouton) lui est associé, il sera disponible pour l'utilisateur. Une commande associée à un widget aura une variable d'entrée particulière appelée *déclencheur*. Ce déclencheur est validé lorsque l'utilisateur agit sur le widget associé.

Les commandes sans déclencheur sont déclenchées lorsqu'elles sont disponibles, celles avec déclencheur sont déclenchées lorsqu'elles sont disponibles et que le déclencheur devient valide.

Nous avons introduit la notion de *précondition d'acceptation*, spécifique des interfaces pédagogiques. Elle permet au concepteur de l'interface de contrôler certaines erreurs de l'élève. L'action associée à la commande sera exécutée si la commande est déclenchée et si la précondition d'acceptation est vérifiée. On laisse ainsi à l'utilisateur la possibilité de faire certaines erreurs (ce qui, dans un contexte pédagogique peut être une bonne chose) mais sans conséquence néfaste (l'action n'est pas exécutée et l'erreur est expliquée).

4. Un exemple

Pour présenter notre travail nous allons nous appuyer sur un exemple très simple. Il se fait en deux étapes. Dans la première partie le domaine est un monde d'"individus". Un individu est décrit par son nom, son prénom, son genre. L'interface permettant à l'utilisateur de rentrer les caractéristiques d'un individu se présente ainsi :

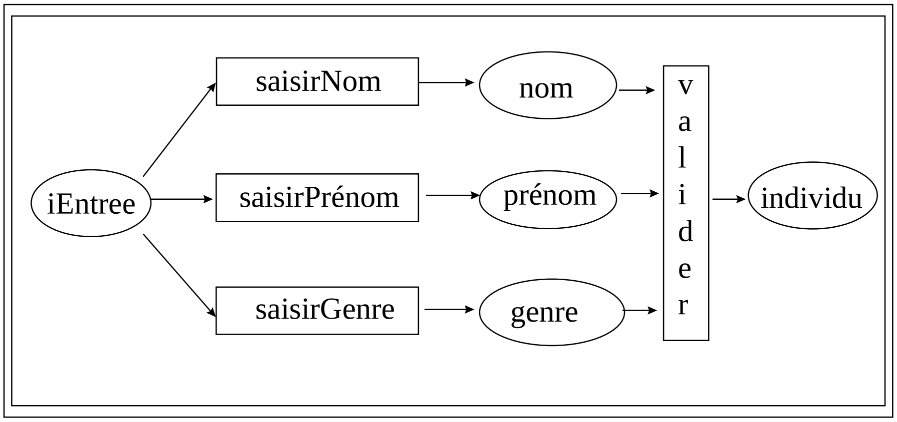


Figure 4 : *interacteur pour une interface de saisie d'un individu*

5. Etat de l'interacteur

Tous les interacteurs dont nous avons eu besoin présentent des comportements communs. Ils nous permettent de construire un objet en saisissant les réponses de l'élève et la fenêtre associée se présente sous diverses formes au début, en cours et à la fin de la saisie. Ceci nous a amené à introduire la notion d'état de l'interacteur. Nous avons défini quatre états : invalide, prêt, en cours, complet. Nous avons ainsi pu associer à ces états la visibilité ou non des widgets.
L'état invalide correspond à un interacteur qui ne peut pas être utilisé (il n'y a pas de fenêtre).
L'état prêt est l'état initial de l'interacteur (cf. Figure 2). Dès que l'utilisateur agit sur l'interface, l'interacteur passe à l'état en cours ou à l'état complet (final) s'il a terminé.
Nous avons associé un automate de transition à chaque interacteur qui prend en compte ces quatre états et les transitions entre ces quatre états. Ces transitions se produisent lors des événements de validation ou d'invalidation des variables.

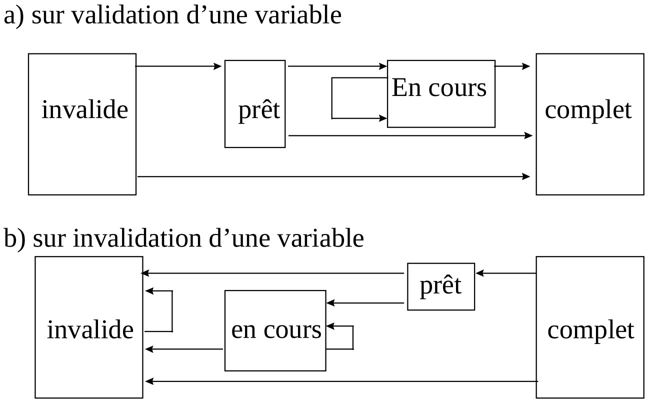


Figure 5 : *états et transitions d'un interacteur*

L'automate étant défini à part une fois pour toutes, chaque interacteur n'a plus à redéfinir cette structure commune. Cet automate complète le réseau de contrôle et sert en fait d'abréviation. Il pourrait être traduit directement dans le réseau, mais cela surchargerait tous les automates. C'est une composante commune à tous les interacteurs.
La détermination de l'état courant et des transitions à exécuter fait jouer un rôle particulier à certaines variables de l'interacteur, qui doivent être distinguées dans le réseau de contrôle. Pour distinguer ces variables, on a ajouté au formalisme la notion de statut d'une variable. Les statuts possibles sont : *variable d'entrée*, *variable interne*, *variable de sortie* de l'interacteur; Dans notre exemple, iEntrée est la variable d'entrée, nom, prénom et genre sont des variables internes et individu est la variable de sortie. Lorsqu'un

événement survient l'interacteur choisit la transition qui le fait passer dans l'unique état dont la condition d'activation est vérifiée.
Voici les conditions d'activation des états :

Prêt : les variables d'entrée sont toutes valides, les variables de sortie ne sont pas toutes valides et la précondition d'autorisation est vérifiée.
En cours : même condition que prêt.
Complet : les variables d'entrée sont toutes valides et les variables de sortie aussi.
Invalide : les conditions d'activation des autres états sont toutes fausses.

Figure 6 : *conditions d'activation des états d'un interacteur*

Nous avons spécifié l'interacteur, nous allons maintenant utiliser EDIREC (Environnement Interactif de Développement d'Interfaces à Réseaux de Contrôle) pour implémenter cet interacteur.

6. EDIREC

Le logiciel EDIREC que nous avons développé permet d'éditer une spécification d'interfaces dans le formalisme IREC, il engendre la quasi-totalité du code exécutable de l'interface avec ses widgets dans l'architecture AGIREC, il assiste le programmeur d'interfaces pour compléter ce code et le mettre au point.
AGIREC pour Architecture Générique pour les Interacteurs à Réseau de Contrôle, est une architecture logicielle à base d'objets qui permet de mettre en oeuvre IREC. EDIREC a été développé en Smalltalk, dans l'environnement VisualWorks.
La fenêtre de l'assistant proposé au concepteur de l'interacteur comporte trois grandes zones, deux concernant la spécification (la liste des composants et la zone d'édition d'un composant) et la troisième les fonctions de génération et de mise au point du code Smalltalk de l'interface. La figure 7 montre cette structure.

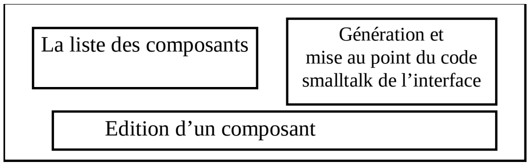


Figure7 : *structure de la fenêtre d'EDIREC*

La figure 8 montre l'assistant de création pour un interacteur au cours de la création de l'interface de saisie d'un individu à partir de l'interacteur décrit au paragraphe 4.1. Dans la zone "La liste des composants" on voit les composants déjà créés.

8.1. Exemple

Le domaine des "individus" s'enrichit de la notion de "professionnel". Le professionnel est un individu qui a une profession, et qui peut avoir ou non un associé (un individu). Nous avons conçu les différents aspects de l'interface en utilisant les interfaces correspondant à individu aussi bien pour la saisie du professionnel que pour celle de son associé.

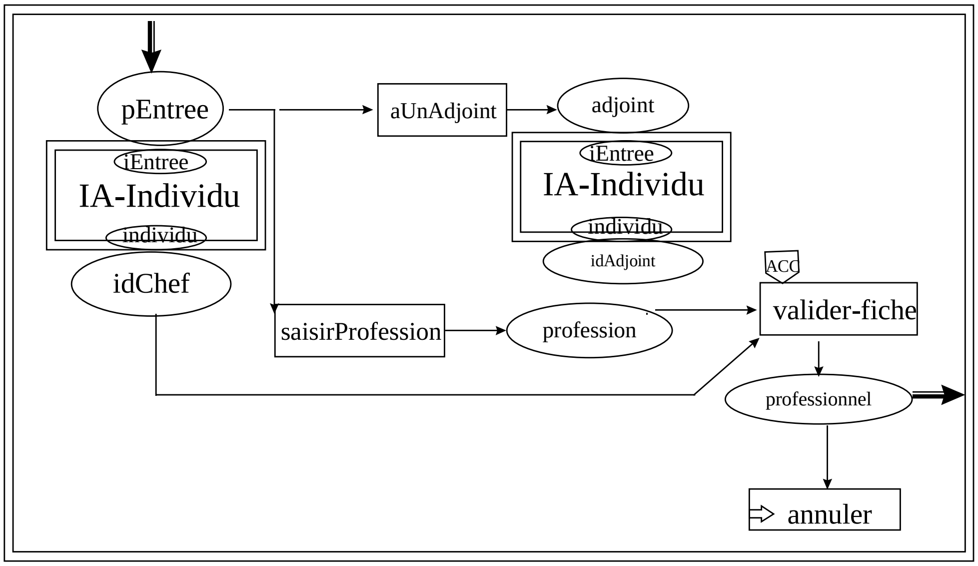


Figure 15 : *interacteur pour une interface de saisie d'un professionnel*

Leurs variables iEntrée et individu sont partagées avec les variables respectives de l’interacteur saisie d’un professionnel pEntrée, idChef d’une part et adjoint et idAdjoint d’autre part. On remarque que dans cet interacteur la commande valider-fiche a deux variables d’entrée idChef et profession. idAdjoint n’est pas variable d’entrée de cette commande car un professionnel n’a pas obligatoirement un associé. Nous avons associé à la commande valider-fiche une condition d’acceptation : si « est assisté par » alors un individu associé doit avoir été créé, qui conditionne l’exécution de l’action associée à la commande. Ceci permet d’être sûr que lorsque la case à cocher « est assisté par » est cochée un individu associé a bien été créé. L’appui sur le bouton commande valider-fiche fait apparaître la fenêtre modale ci-dessous au cas où l'information n'est pas complètement entrée.

[Palanque et al. 97] Palanque Ph. & Paternò F.(eds). : *Formal Methods in Human-Computer Interaction* Springer Verlag 1997.

[Tisseau et al. 96] Tisseau G., Giroire H., Le Calvez F., Urtasun M., Duma J. : *Une méthode « constructive » de résolution de problèmes de dénombrement et sa mise en œuvre* - Rapport interne Laforia 96/11, mai 1996.

[Tisseau et al. 99] Tisseau G., Giroire H., Duma J., Le Calvez F., Urtasun M. : *Spécification du dialogue et génération d'interfaces à l'aide d'interacteurs à réseau de contrôle* - IHM 99, Montpellier, 23-26 novembre 1999

GENSAM: un système généralisant les petits échantillons

Michel Masson
LIP6
masson@lip6.fr

Résumé: Ce papier décrit le système GENSAM, ce système est capable de généraliser un ensemble d'observations lorsque celui-ci est petit. Il est basé sur l'explicitation des connaissances nécessaires à la généralisation, la démarche est générale et permet l'utilisation de connaissances liées au domaine de l'utilisateur pour améliorer l'analyse de l'échantillon étudié. Nous montrons les résultats obtenus et les problèmes liés à la validation de la méthode.

Mots-clés: échantillon, généralisation, méta-connaissances.

1. Introduction

Le cadre de l'étude reste identique à celui présenté dans [Masson 98]: un ensemble de dossiers médicaux rassemble des résultats d'examens biologiques, des données cliniques ainsi que des données techniques (diamètre de la sonde utilisée, durée de l'intubation, etc.). Chaque élément de l'échantillon est assorti d'un classement a posteriori (évolution du patient avec un recul de 6 mois). Il s'agit alors d'obtenir à l'aide des seules données de départ un des diagnostics à 6 mois en utilisant une combinaison *simple* des informations de départ avec une marge d'erreur aussi faible que possible.

Ces données ont été recueillies lors du lavage et/ou brossage bronchoalvéolaire en l'absence de contrôle fibroscopique chez l'enfant sous assistance respiratoire pour l'aide au diagnostic des infections respiratoires [Labenne & Goldfarb 98]. En effet, l'étroitesse de la trachée chez l'enfant ne permet pas l'introduction du matériel de lavage et/ou brossage et de l'endoscope (permettant le contrôle visuel des opérations), il convient alors de valider une série (petite) d'examens réalisés sans contrôle visuel.

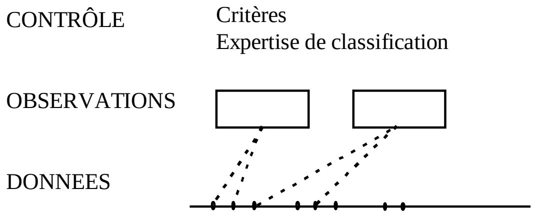
La sensibilité des examens pris en compte impose un protocole très strict, d'autant mieux respecté qu'une même équipe conduit l'ensemble des observations. Ceci explique le petit nombre d'observations (entre 39 et 96 dans notre étude).

Ainsi rapporté, ce problème correspond à la construction d'une classification. De nombreuse solutions ont été proposées: méthodologie statistique (régression, analyse discriminante), [Tomassone 88a], [Tomassone 88b], algorithme ID3 [Quinlan 79], [Quinlan 83], [Quinlan 86], méthode SCART [Breiman & al 84], raisonnement par cas [Kolodner 93], [Kolodner & Simpson 89]. Nous avons montré [Masson 99] qu'aucune de ces approches ne correspondait exactement au problème traité: taille trop faible de l'échantillon, indépendance des éléments non vérifiée, impossibilité de prendre en compte les connaissances du domaine.

Notre approche est basée sur une observation: un opérateur humain perspicace peut contourner ces difficultés en affinant les limites des méthodes précédentes ou en les combinant. L'explicitation de ce raisonnement constitue notre approche.

2. Aperçu général de GENSAM

Le système est basé sur la standardisation des données et une analyse de celles-ci sur trois niveaux. Les données de départ, après standardisation, constituent le niveau DONNEES; ce niveau est enrichi au fur et à mesure du déroulement du processus. Ces données sont regroupées en fonction de leur aptitude à construire une classification; c'est le niveau OBSERVATIONS. Un niveau CONTROLE analyse ces regroupements à travers plusieurs critères indépendants du domaine d'application à l'aide d'une expertise de classification pilotant un ensemble de tâches: regroupement d'ensembles, fixation d'un seuil, etc.



2.1. Notations

Nous utiliserons les notations suivantes:
Un échantillon E est défini de la façon suivante: $E = \{e_1, ... , e_n\}$, où e_i désigne un individu caractérisé par un groupe d'appartenance et une liste de paires < attribut valeur >. Nous appellerons **sous-population** tout sous-ensemble de E.

> Ex. l'ensemble des individus dont le diagnostic à 6 mois (i.e. le groupe d'appartenance) est NIS, l'ensemble des individus dont le poids est inférieur à 3kg et la durée d'intubation est inférieure à 3 jours constituent deux sous-populations de E.

GENSAM doit pouvoir juger ses résultats, pour cela nous introduisons deux mesures (indépendantes du domaine étudié). Soient E_1, $E_2 \subset E$; nous pouvons alors définir la **sensibilité** et la **spécificité** de E_1 par rapport à E_2:
Définition 1:

$$\mathbf{sens}\ (E_1, E_2) = \text{card}\ (E_1 \cap E_2)\ /\ \text{card}\ (E_2)$$

$$\mathbf{spec}\ (E_1, E_2) = \mathbf{sens}\ (E_2, E_1) = \text{card}\ (E_1 \cap E_2)\ /\ \text{card}\ (E_1)$$

Si S_i désigne la sous-population où le signe s_i est présent, si NIS désigne la population dont le diagnostic confirmé à 6 mois est Non Infecté Sûr, **sens** (S_i , NIS) mesure la prévalence de S_i chez la sous-population NIS; elle est d'autant plus proche de 1 que le signe est fréquemment rencontré. Une valeur élevée de **spec** (S_i , NIS) indique que le signe est fréquemment rencontré en dehors de la sous-population NIS. Ainsi S_i caractérise d'autant mieux la sous-population NIS que l'on a:

sens (S_i , NIS) et **spec** (S_i , NIS) proches de 1 (un seuil de précision est fixé initialement).

2.2. Standardisation des données:

Au départ, l'échantillon E peut être décrit comme un ensemble d'éléments de la forme:

$$E = \{ (\text{<groupe } g_i\text{> <}var_1\ val_1\text{> ... <}var_p\ val_p\text{>}) \}$$

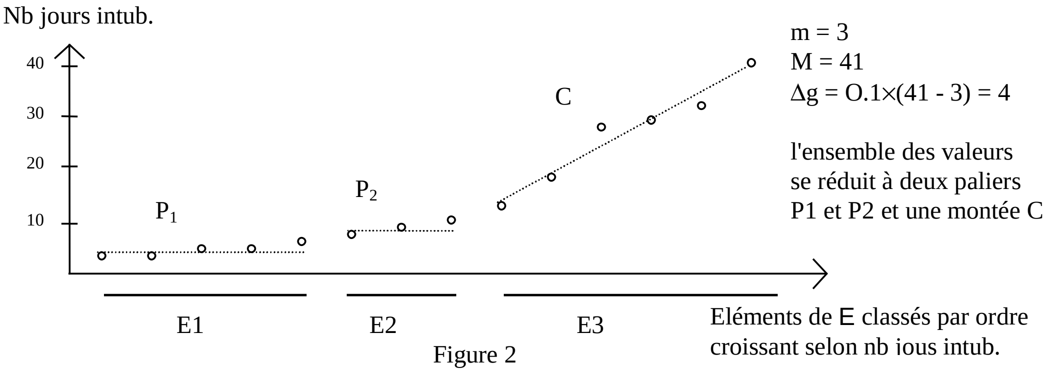
La standardisation repose sur le type des valeurs prises par les variables var_i:

a) Pour chaque variable prenant une valeur qualitative ou booléenne, est associée une sous-population constituée des éléments de E ayant cette valeur pour cette variable. En reprenant les variables utilisées dans l'étude, on obtient:
   E_1 = E {pneumothorax = oui}
   E_2 = E {pneumothorax = non}
   ....

b) Le cas des variables quantitatives est différent car il s'agit de décrire les variations de ces variables. Les valeurs possibles d'une variable quantitative (par ex. **v**) sont classées en ordre croissant, un indice de granularité Δg permet d'ignorer les petites différences de valeur. Au départ, cet indice vaut 0.10×(M - m), où M et m désignent les valeurs extrêmes de la variable étudiée. Un **palier P** est défini de la façon suivante (initialement : $\Delta_g = 0.1$)

$$\mathbf{P} = \{e_i \in E\ /\ \forall\ e_j \in E\ ,\ |\ v\ (e_i) - v\ (e_j)| \leq \Delta_g \times v\ (e_i)\}$$

Les autres intervalles correspondent à des **montées**.
Soit la variable nombre_de_jours_d'intubation (exprimée en jours) d'un échantillon E, les valeurs classées de cette variable fournissant la suite: [3 3 5 5 7 8 10 11 13 18 28 30 33 41 ...]



m = 3
M = 41
Δg = O.1×(41 - 3) = 4

l'ensemble des valeurs se réduit à deux paliers P1 et P2 et une montée C

Figure 2

Chaque segment (P ou C) constitue une sous-population E_i de l'échantillon de départ E.

Au départ, seules les sous-populations correspondant aux variables qualitatives (§ a) et aux variables quantitatives (§ b) de l'échantillon sont présentes; GENSAM construit au fur et à mesure par agrégation des précédentes de nouvelles sous-populations qui sont alors décrites par une liste <attribut valeur> semblable à la précédente.

3. Le niveau DONNEES

Il regroupe l'ensemble des sous-populations manipulées par GENSAM; elles sont de deux types:

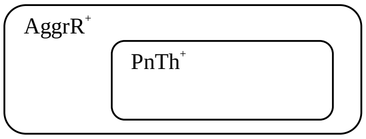
- les sous-populations de type T (pour Test): ce sont les sous-populations engendrées par les tests initiaux à partir de la standardisation, il s'y ajoute des combinaisons de celles-ci (obtenues par intersection, réunion, fixation de seuil sur une variable quantitative) au fur et à mesure de l'avancement du processus de caractérisation.
- les sous-populations de type G (pour Groupe): ce sont les sous-populations correspondant aux groupes prédéfinis initiaux (par ex. NIS, NIP,IP, IS) auxquelles s'ajoutent les nouveaux groupes créés par GENSAM. Ainsi à un instant donné le groupe prédéfini G peut être caractérisé partiellement par E, une sous-population de type T; deux stratégies sont alors possibles: améliorer la caractérisation existante ou considérer le complémentaire de E dans G comme un nouveau groupe à caractériser.

C'est à ce niveau que sont prises en compte certaines connaissances du domaine sous forme de relations entre sous-populations. Elles sont de la forme *(objet lien objet)* où *lien* peut désigner un terme comme *implique*, *évoque*, *varie comme*. Ainsi la dépendance évoquée au § 3.1 peut se traduire par:

"la présence d'un pneumothorax implique une aggravation respiratoire"

Posons $PnTh^{+} = \mathcal{E}$ {présence_pneumothorax = oui}, $AggrR^{+} = \mathcal{E}$ {aggravation_respiratoire = oui}, la connaissance précédente peut se traduire par:

spec $(PnTh^{+}, AggrR^{+}) = 1$



4. Le niveau OBSERVATIONS

Ce niveau regroupe les sous-populations en fonction de leur aptitude à participer à la construction d'ensembles plus complexes. Pour cela, plusieurs catégories de sous-populations sont définies; l'analyse de celles-ci au niveau CONTROLE permet de construire une caractérisation de l'échantillon.

4.1. Sous-population critique

Définition 2: Soit $E \subset \mathcal{E}$, E est dite **sous-population inséparable à x%** (SPI_x) ssi:
$\forall x \, \forall y \in E$, x et y sont semblables sur x% des tests initiaux.
Définition 3: Soit M une sous-population de type G, M est **x-critique** ssi:
$\exists$ E sous-population inséparable à x% telle que: **sens** (M, E) > 100 - x
(au départ, $x=\Delta_I$ est fixé à 0.9)
Interprétation: une sous-population inséparable correspond à un ensemble d'individus qui ne peuvent être discriminés sur x% des tests initiaux. Une sous-population de type G est critique si elle contient un ensemble d'éléments inséparables sur certains critères; si x est élevé cette sous-population est difficile à caractériser.

4.2. Sous-population granulaire

Définition 4:
Une sous-population de type G est granulaire si

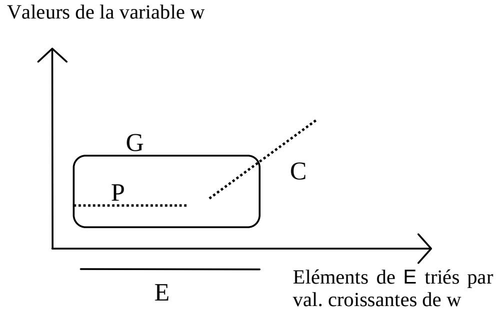
$$\forall E_i, \text{ sous-population avec } \mathbf{spec}\ (E_i, G) = 1 \text{ alors } \frac{Card(\cup E_i)}{\sum_i Card(E_i)} \rangle c_G$$

c_G: coefficient de granularité initialement fixé à 0.5

Interprétation: une sous-population M est granulaire lorsqu'elle contient plusieurs sous-populations dont la réunion couvre significativement M. Une caractérisation de M peut alors être obtenue grâce à $\cup E_i$.

4.3. Sous-population seuillable

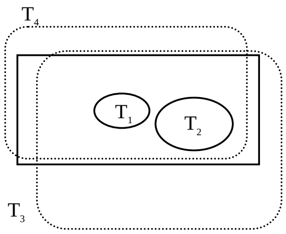
Lorsque qu'une sous-population est obtenue à partir d'une variable quantitative (cf. § 2.2.b), elle peut être redéfinie en fixant un seuil à la variable quantitative: c'est le cas des sous-populations **seuillables**. Soient P et C deux sous-populations quantitatives associées à la variable w: P décrit une suite d'éléments de valeurs identiques (proches de v_1), C décrit une suite d'éléments de valeurs croissantes (comprises entre v_2 et v_3), soit G une sous-population telle que $P \subset G$; il est probable que G peut être caractérisée par la réunion de P et d'une partie des éléments de C en définissant une sous-population $E = \{e_i \in E \ / \ v_1 \leq w(e_i) \leq v', v' \in [v_2, v_3]\}$. On dira que G est obtenue par seuillage à partir de P (i.e. fixation d'un seuil hors des valeurs de P). Les conditions d'appartenance à la catégorie "Sous-populations Seuillables" peuvent s'écrire:



C_3: **sens** (G, P) = 1 la sous-population G contient la sous-population P
C_4: **adj** (P) = C P et C sont deux sous-populations adjacentes
C_5: **type** (P) = Palier P correspond à une sous-population quantitative de type Palier
C_6: **type** (C) = Montée C correspond à une sous-population quantitative de type Montée

4.4. Sous-populations à sensibilité/spécificité élevée

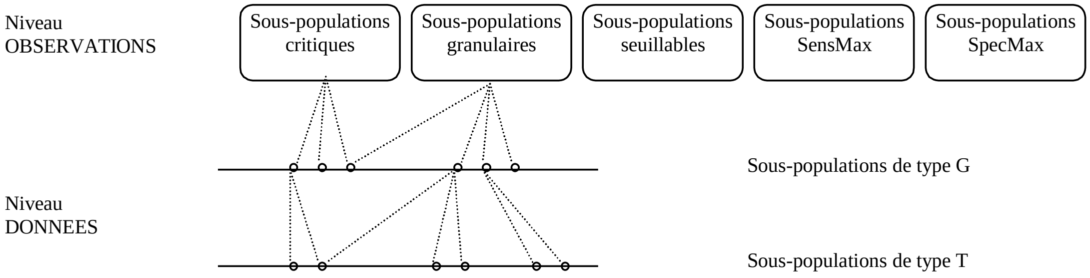
Ces deux catégories répertorient les bons éléments a priori de l'échantillon de départ. Dans la figure ci-contre, M représente une sous-population de type G, les sous-populations T1 et T2 sont très spécifiques de M mais peu sensibles, par contre T3 et T4 sont très sensibles mais peu spécifiques. L'idée est de rechercher les sous-populations les plus spécifiques [resp. sensibles] parmi les sous-populations de forte sensibilité [resp. spécificité]. Dans l'exemple de la figure ci-contre, T_2 [resp. T_4] joue ce rôle.



Définition 5: Etant donnée M une sous-population, SensMax (M) désigne les sous-populations les sensibles à M parmi les sous-populations de forte spécifité.
Posons SPEC (M) = { X ∈ E / **spec** (X , M) > Δ_S} (initialement, $\Delta_S = 0.9$)
SensMax (M) = { X ∈ SPEC (M) / 10% meilleurs valeurs de **sens** (X , M)}
Symétriquement, on a:
Définition 6:
SENS (M) = { X ∈ E / **sens** (X , M) > Δ_S}
SpecMax (M) = { X ∈ SENS (M) / 10% meilleurs valeurs de **spec** (X , M)}

4.5. Conclusion

Le niveau OBSERVATIONS comprend ainsi cinq catégories auxquelles peuvent se rattacher les sous-populations de type G (elles correspondent aux groupes initiaux et aux groupes générés par le système - cf. § 3). A chacune de ces sous-populations de type G est associé un ensemble des sous-populations de type T adaptées aux opérations de sa catégorie. Chaque nouvelle sous-population créée est automatiquement rattachée à sa ou ses catégories en fonction de ses conditions d'appartenance.



5. Le niveau CONTROLE

5.1. Définitions

Ce niveau gère l'ensemble des informations précédemment collectées. A chaque catégorie du niveau OBSERVATIONS correspond une action privilégiée. Par exemple, si M est une sous-population apparentée à la catégorie des sous-populations granulaires, alors il existe (déf. 4) $G_1, \ldots , G_p$ avec **spec** (G_i , M) = 1 (i = 1, ... , p); la sous-population M' = $G_1 \cup \ldots \cup G_p$ fournit une caractérisation de M (qui sera améliorée aux étapes suivantes).
Il convient de déterminer la catégorie la plus intéressante, puis dans cette catégorie les éléments apportant la meilleure caractérisation des groupes prédéfinis ou de leurs dérivés. Pour cela, chaque catégorie est analysée à l'aide d'une mesure appelée **coefficient stratégique** (le calcul de ce coefficient - noté **cs** - est propre à chaque catégorie); ce coefficient permet de calculer trois critères - **intr, distr, tr** - concernant les cinq catégories du niveau OBSERVATIONS.

Coefficient stratégique d'une sous-population critique:
Soit M une sous-population **critique**, il existe alors une sous-population C dont les éléments sont faiblement dissociables. On pose : **cs** (M) = **sens** (C , M)
(ce coefficient est d'autant plus élevé que M comporte une partie de ses éléments difficilement caractérisables)

Coefficient stratégique d'une sous-population granulaire:
Soit M une sous-population **granulaire**, il existe alors $G_1, \ldots , G_p$ avec **spec** (G_i , M) = 1, (i = 1, ... , p).
On pose: **cs** (M) = **sens** (M , $\cup G_i$)
(ce coefficient est d'autant plus élevé que $\cup G_i$ couvre M)

Coefficient stratégique d'une sous-population seuillable:
Soit M une sous-population **seuillable**, il existe alors (cf. § 2.2) une sous-population C de type **montée** caractérisée par les valeurs extrêmes prises par la variable quantitative associée à M. On pose: **cs** (M) = **sens** (C , M)

Coefficient stratégique d'une sous-population

En conclusion, le coefficient stratégique d'un élément est d'autant plus fort que son appartenance à une des cinq catégorie est avérée: c'est donc un élément à traiter prioritairement dans sa catégorie. Il est utilisé pour le calcul des trois critères suivants:

intérêt d'une catégorie C (notée int):

int (C) = Moyenne des 10% meilleurs éléments de C pour la mesure **cs**

distribution d'une catégorie C (notée distr):

$$\textbf{distr}\ (\ C\ ) = \frac{\text{Moyenne des 10\% meilleurs éléments de C pour la mesure } \textbf{cs}}{\text{Moyenne des éléments de C pour la mesure } \textbf{cs}}$$

taille relative d'une catégorie (notée tr):

$$\textbf{tr} (C) = \frac{\text{Card} (C)}{\text{Max} \{ \text{Card} (C_i) / i = 1,5 \}}$$

$C_1, \ldots , C_5$ désignent les catégories du niveau OBSERVATIONS

5.2. L'expertise de contrôle

L'expertise de contrôle permet de sélectionner pour chaque catégorie l'action la plus pertinente. Trois types d'action sont envisageables:

- Améliorer une solution existante (ex. si $x = T_1 \cup T_2 \cup T_3$ est la caractérisation d'un groupe prédéfini et également un élément de la catégorie granulaire, une amélioration de x peut être obtenue en remplaçant T_1 et/ou T_2 et/ou T_3 par des éléments de cette catégorie - les éléments dont la valeur de **cs** est la plus élevée sont alors choisis).
- Complémenter une solution existante (ex. si x, caractérisation de G, ne peut être améliorée, on construit un nouveau groupe en prenant le complémentaire de x dans G.
- Modifier les seuils Δ_g, Δ_I, Δ_S, c_G

Ces actions sont décidées en fonction du résultat des critères **int**, **distr**, **tr** pour les catégories du niveau OBSERVATIONS.

Notations: Soit P un critère appliqué aux éléments de C et prenant ses valeurs dans l'intervalle [0 , 1], $\forall C \in \mathcal{C}$:

$P+ (C)$ ssi $\forall C' \in \mathcal{C} , C' \neq C \; [P+ (C) - P+ (C')] / P+ (C) > 0.3$

$P= (C)$ ssi $\forall C' \in \mathcal{C}, | P (C) - P (C') | / P- (C) < 0.1$

(P+ indique que l'élément C a pour le critère P une valeur plus élevée que les autres éléments de $\mathcal{C}$, P= indique que C a une valeur de P voisine des autres éléments de $\mathcal{C}$)

L'expertise de contrôle prend alors la forme (?C désigne une catégorie du niveau OBSERVATIONS):

R_1 SI [**intr+** (?C) $\vee$ **distr+** (?C)] $\wedge$ **tr=** (?C)] ALORS améliorer ?C
(Si ?C contient de bons éléments ou est meilleure que les autres catégories et comporte peu d''éléments, il faut améliorer les meilleurs éléments de ?C)

R_2 SI [**intr=** (?C) $\wedge$ **distr+** (?C)] ALORS améliorer ?C
(Si ?C contient de meilleurs éléments que ses voisines, il faut améliorer les meilleurs éléments de ?C)

R_3 SI [**distr=** (?C) $\wedge$ **tr+** (?C)] ALORS complémenter ?C
(Si ?C est une catégorie comportant beaucoup d'éléments de même valeur que ses voisines, il faut prendre le complément de ses meilleurs éléments)

R_4 SI [**intr=** (?C) $\wedge$ **tr+** (?C)] ALORS complémenter ?C
(Si ?C est une catégorie comportant beaucoup d'éléments de mauvaise qualité comparativement à ses voisines, il faut prendre le complément des éléments de ?C)

R_5 SI [**distr=** (?C) $\wedge$ **tr=** (?C)] ALORS modifier les seuils: Δg + 5%, Δ_S - 5%
(Si ?C est semblable en intérêt et en taille aux autres catégories, il faut modifier les seuils Δg et Δ_S pour créer de nouvelles sous-populations)

R_6 SI [**intr=** (?C) $\wedge$ **tr=** (?C)] ALORS modifier les seuils: Δ_I - 5%, c_G - 5%
(Si ?C possède peu d'éléments d'intérêt, l'abaissement des seuils Δ_I et c_G crée de nouveaux éléments dans les deux premières catégories du niveau OBSERVATIONS)

L'organigramme général est le suivant:
a) initialisation des coefficients
b) arrêt si les catégories du niveau OBSERVATIONS sont caractérisées
c) application des règles R_1, R_2, R_3, R_4
d) si aucune des règles précédentes n'est applicable, appliquer (une fois) les règles R_5 ou R_6
e) retour en (a)

6. Résultats et Perspectives

La stratégie du niveau CONTROLE ainsi que les calculs des attributs des éléments du niveau DONNEES sont réalisés en C++. Les éléments des niveaux CONTRAINTES et OBSERVATIONS sont des frames auxquels sont associées des règles CLIPS (Système à Base de Connaissances construit autour de l'algorithme RETE). Ce choix d'implémentation, s'il pénalise le temps de réponse, a grandement facilité la comparaison des nombreuses stratégies de contrôle envisagées. Avec un échantillon de 39 éléments, chacun étant décrit par 36 variables et un classement en quatre groupes (IS, IP, NIP et NIS), on obtient une caractérisation de ces groupes en 15 minutes. Sur un échantillon atteignant 96 éléments avec un nombre équivalent de variables, le temps de réponse est de 40 minutes (le nombre de groupes caractérisés a peu d'incidence). En comparant GESAM avec les méthodes classiques [Labenne & Goldfarb 98], les résultats concernant la sensibilité sont semblables, ils sont par compte meilleurs en ce qui concerne la spécificité (ils sont supérieurs à .95 pour les groupes IS et NIS - ce qui correspond à un seul élément mal classé).

		IS	IP	NIP	NIS
Labenne & Goldfarb	**spec**	0.92	0.90	0.94	0.91
	sens	0.92	0.86	0.89	0.90
GENSAM	**spec**	0.95	0.87	0.91	0.95
	sens	0.90	0.84	0.80	0.86

La méthode leave-one-out a permis de tester la stabilité du processus sur les échantillons analysés. Ces résultats valident notre approche qui doit être confirmée sur de nouveaux exemples et en variant les domaines. Les performances pourraient être améliorées en procéduralisant les traitements associés aux catégories du niveau OBSERVATIONS et en utilisant au maximum les connaissances du domaine. Il importe alors de permettre l'introduction de ces connaissances grâce à un langage simple, le langage LCD [Masson 96] adapté à la manipulation de connaissances sur le diagnostic en est une illustration.
Ce travail a été réalisé à partir d'une étude médicale conduite par B. Goldfarb [Laboratoire de Biostatistique et Informatique Médicale, Hôpital Necker & LISE-Ceremade, Université Paris-Dauphine] que je remercie pour m'avoir permis de dégager les éléments essentiels nécessaires au jugement médical.

7. Bibliographie

[Breiman & al 84] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone. *Classification and Regression Trees*, Chapman et Hall, 1984.

[Kolodner 93] J. Kolodner. *Case-Based Reasoning*, Morgan Kaufmann Publishers, San Mateo, Californie, 1993.

[Kolodner & Simpson 89] J. Kolodner, R. L. Simpson. The MEDIATOR: Analysis of an early case-based problem solver, Cognitive Science 13, 507-549, 1989.

[Koton 88] P. Koton. Integrating case-based and causal reasoning, *Proceedings of the Xth Annual Conference of the Cognitive Science Society*, Northvale, 1988.

[Labenne & Goldfarb 98] M. Labenne, B. Goldfarb. Blind-protected specimen brush and brochoalveolar lavage in ventilated children. *Critical Care,* 1998.

[Masson 96] M. Masson. Réifier le raisonnement pour améliorer les explications dans les systèmes à base de connaissances: une application au diagnostic médical, *RJC-IA'96*, Nantes, 1996.

[Masson 98] M. Masson. Généralisation et petits échantillons. *Colloque sur la Métaconnaissance*, Cahiers du LIP6, Berder, 1998.

[Quinlan 79] J. R. Quinlan. Discovering rules by induction from large collections of examples. In *Expert Systems in the Micro-electronic Age*, 168-201, Michie, D., Editor, Edinburgh University Press, Edinburgh, Scotland, 1979.

[Quinlan 83] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine Learning, An Artificial Intelligence Approach*, Pages 463-482, Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., Editors, Tioga Publishing Company, Palo Alto, CA, 1983.

[Quinlan 86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81-106, 1986.

[Tomassone 88a] R. Tomassone. *Comment interpréter les résultats d'une régression linéaire ?* ITCF, Paris, 1988.

[Tomassone 88b] R. Tomassone. *Comment interpréter les résultats d'une analyse factorielle discriminante ?* ITCF, Paris, 1988.

[Utgoff 89] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161-186, 1989.

Un Système Expert et un Outil pour la Reconnaissance Automatique des Manœuvres Automobiles

NIGRO Jean-Marc
Université de Technologie de Troyes
Laboratoire LM2S
12, Rue Marie Curie
BP 2060
10010 Troyes Cedex
E-Mail : nigro@univ-troyes.fr

Résumé: Dans un premier temps, cet article décrit la mise au point d'un logiciel "Intelligent" permettant de reconnaître les manœuvres effectuées par un véhicule. Nous nous sommes pour le moment intéressés à la manœuvre de dépassement. Pour reconnaître la manoeuvre, nous utilisons les données issues de capteurs proprioceptifs et extéroceptifs, embarqués sur un véhicule expérimental de type Xantia break. Le système IDRES réalisé est un système à base de règles doté de plusieurs couches de décision.
Dans un second temps, l'article décrit l'utilité d'un méta-moteur d'inférences pour le développement de techniques en IA basées sur les systèmes à base de règles. Le méta-moteur d'inférences est un moteur d'inférences représenté sous la forme de règles et de faits. Cette représentation permet de concevoir très facilement des méta-règles, des mécanisme de contrôle et d'apprentissage. Le fonctionnement du méta-moteur est illustré avec les règles d'IDRES développé dans le cadre du projet CASSICE.

Mots Clés: Intelligence Artificielle, moteur d'inférences, système à base de règles, méta-connaissance, déclarativité, niveaux de décision.

1. Introduction

La première partie de l'article présente la réalisation d'un dispositif expérimental "Intelligent" permettant de reconnaître en temps réel les manœuvres effectuées par un conducteur automobile. Ce système entre dans le cadre du projet CASSICE (Caractérisation Symbolique de Situations de ConduitE). Il s'appuie sur une étroite collaboration entre des psychologues, des chercheurs en imagerie, en automatique, et en Intelligence artificielle.

Nous nous sommes intéressés dans un premier temps à la manœuvre de dépassement d'un véhicule cible (VC) par notre véhicule expérimental (VE). Le système développé considère qu'un dépassement peut être décomposé en dix étapes : attente de dépassement, signalement de l'intention de dépasser, début de déboîtement, franchissement de la ligne discontinue à gauche, redressement, etc.). Chacune de ces étapes correspond à un état du système "environnement - véhicule expérimental"[1]. La reconnaissance de chaque état est validée à l'aide des différents capteurs et des caméras embarqués dans le véhicule.

Les règles de notre système IDRES (Intelligent Driving Recognition with Expert System) reconnaissent chacune des étapes du dépassement de manière indépendante les unes des autres. Elles se répartissent en 2 niveaux. Le premier niveau de décision génère à chaque instant des hypothèses sur le ou les états dans lesquels VE peut se trouver. L'enchaînement des états correspondant à une manœuvre devant suivre un scénario, le second niveau de décision se préoccupe de reconstituer une séquence d'états cohérente vis à vis des manœuvres que le système est capable de reconnaître.

---

[1] C'est pourquoi dans la suite, nous utiliserons indifféremment les termes "étapes" et "état"

La seconde moitié de l'article présente la construction d'un système d'inférences permettant de construire aisément un niveau de contrôle et d'apprentissage indépendant du domaine d'application. Le système s'applique à un système à base de règles dédié à un domaine bien défini. Dans un premier temps, le système expert (SE) du domaine d'application sera celui utilisé pour la reconnaissance de la manœuvre d'un véhicule (le projet CASSICE **-** Caractérisation Symbolique de Situations de ConduitE [Nigro&al 99]) en fonction des données fournies par des capteurs et caméras embarqués. Par la suite, le méta-moteur pourra être appliqué à d'autres domaines d'application.

L'ajout d'un niveau contrôle et apprentissage à un système à base de règles permettra, par exemple, de diriger le déclenchement des règles vers la solution (la stratégie de déclenchements) ou encore de modifier des paramètres contenus dans les règles du SE du domaine pour de meilleurs résultats (l'adaptation) ou encore modifier la base de règles initiale avec des suppressions, ajouts de règles (l'apprentissage).

Pour atteindre cet objectif, le principe est de construire un méta-moteur d'inférences, c'est à dire un système à base de règles capable de simuler le fonctionnement d'un moteur d'inférences. Le méta-moteur possède donc des règles d'instanciation et de déclenchement d'une règle. Les systèmes experts (règles et faits) du domaine d'application seront réécrites sous la forme d'un ensemble de faits utilisable par le méta-moteur. Celui interprétera les règles du domaine puis tentera de les déclencher sur les faits du domaine.

L'utilisation du méta-moteur permettra non seulement d'ajouter facilement un méta-contrôle à un système à base de règles, mais également de donner la possibilité de définir de vraies méta-règles : une règle pouvant en modifier une autre ou encore capable de se modifier elle-même.

2. Le projet Cassice

La caractéristique primordiale de la conduite automobile tient au fait que l'homme, en temps que conducteur, est entièrement impliqué dans le processus de commande. Tenter d'améliorer la sécurité routière nécessite d'étudier et de prendre en compte le comportement humain en situation de conduite. En effet, depuis une bonne décennie, de nombreux dispositifs d'aide à la conduite ont été développés sans prendre en compte l'aspect humain dû au fait que de tels dispositifs sont utilisés par un conducteur. Cette étude est complexe à cause, d'une part, du sujet étudié (le conducteur humain) mais aussi à cause des situations nombreuses et variées dans lesquelles il peut se trouver, celles-ci influant fortement sur son comportement.

2.1. Objectifs

L’objectif visé dans ce projet est la réalisation d’un système informatisé capable de répertorier des situations de conduite réelles. Par un accès multi-critères à des situations de conduite, ce dispositif permettra aux psychologues de rechercher plus efficacement des situations de conduite ayant eu lieu dans tel ou tel contexte. Cette base de données spatio-temporelle, sera exploitée de diverses manières. Le psychologue utilisateur devra pouvoir en effet demander à accéder à la première manœuvre effectuée à partir d'une distance de 2500 m à partir du point de départ, ou alors visualiser les manœuvres de dépassement suivies d'un freinage brutal, etc. Cet outil permettra de mieux comprendre les motivations, et plus généralement le modèle comportemental justifiant l’attitude du conducteur étudié. Il sera alors possible plus tard d’envisager les conséquences sur la conduite, de l’introduction de tel ou tel dispositif automatique ou semi-automatique d’aide à la conduite.

2.2. Environnement expérimental

Ce projet s’appuie sur une collaboration entre plusieurs laboratoires de recherche, dont le laboratoire LPC (Laboratoire de Psychologie de la Conduite) de l’INRETS. Il s’appuie sur un véhicule équipé d’un ensemble de capteurs proprioceptifs et de caméras, nommé "STRADA". Les acquisitions de données sont réalisées dans des conditions de circulation bien précises : il s’agit de trajets réalisés sur autoroute urbaine, de style périphérique, sur lesquels tous les véhicules se déplacent dans le même sens.

2.2.1. Les données fournies par les capteurs

Les données acquises proviennent de capteurs proprioceptifs, d’origine pour certains d’entre eux : compte tour, vitesse relative du véhicule par rapport au véhicule précédent, vitesse de rotation des roues, éclairage et signalisation, capteur d’angle au volant, capteur de pression sur le frein, accéléromètre. Des capteurs extéroceptifs sont également utilisés : des caméras avant, arrière, latérales et un télémètre avant. Une acquisition est réalisée tous les centièmes de seconde. Les variations de ces données sont importantes suivant le stade de réalisation de la manœuvre. Par exemple, lors d'un dépassement, la valeur de x, représentant la distance séparant deux véhicules successifs, devient positive lors d'un doublement de l'un des deux véhicules par l'autre. Ces variations significatives de certaines données sont à la base de la reconnaissance de la manœuvre effectuée.

2.2.2. Le simulateur

Actuellement, plusieurs chercheurs et ingénieurs s'affairent à l'installation et au réglage des capteurs et caméras. Ainsi, les données fournies au système de reconnaissance des *états* proviennent d'un simulateur logiciel. Cette simulation permet de récupérer différentes valeurs utiles pour la reconnaissance de la manœuvre étudiée. Le tableau ci-dessous (cf Fig. 1) représente les informations fournies par le simulateur, à la base de la reconnaissance effectuée.

Temps	X	Y	V	Teta	Acc	Phi	Rg	Rd
+0.01	-32.00	+0.00	+15.00	+0.00	+0.00	+0.00	-3.50	+1.50
+0.02	-31.85	+0.00	+15.00	+0.00	+0.00	+0.00	-3.50	+1.50
+0.03	-31.70	+0.00	+15.00	+0.00	+0.00	+0.00	-3.50	+1.50
+0.04	-31.55	+0.00	+15.00	+0.00	+0.00	+0.00	-3.50	+1.50
+0.05	-31.40	+0.00	+15.00	+0.00	+0.00	+0.00	-3.50	+1.50
...	...	...	...	...	...	...	...	...
+1.11	-15.66	-1.99	+15.00	-10.13	+0.00	+3.00	-1.49	+3.51
+1.12	-15.52	-2.01	+15.00	-9.91	+0.00	+3.00	-1.46	+3.54
+1.13	-15.37	-2.04	+15.00	-9.68	+0.00	+3.00	-1.44	+3.56
+1.14	-15.22	-2.06	+15.00	-9.46	+0.00	+3.00	-1.41	+3.59

Fig. 1 : *Données acquises par le véhicule expérimental*

Les données représentées ont la signification suivante :

Temps	Horloge (en s)
Acc	Accélération relative de VE par rapport à VC (en m²/x)
Phi	Angle des roues avant de VE (en degré)
Rd	Position de VE par rapport au coté droit de la route (en mètre)
Rg	Position de VE par rapport au coté gauche de la route (en mètre)
Teta	Angle de la cible VC (en degrés)
V	Vitesse relative de VE par rapport à VC (en m/s)
X	Position relative en x de VC par référence à VE (en mètre)
Y	Position relative en y de VC par référence à VE (en mètre)

Fig. 2 : *signification des données*

Les données acquises nous apportent de précieux renseignements sur la position du véhicule, sa vitesse, etc. Par exemple, la lecture des données (fig. 1) montre que VE évolue avec une vitesse constante (V=15m/s plus vite que le véhicule situé devant) et donc sans accélération (Acc = 0). Dans une première partie, VE avance en ligne droite (Y est constant) et est positionné juste derrière VC (Y=0) sur un axe parallèle à l'autoroute sur laquelle circulent les 2 véhicules. Le véhicule VE se rapproche régulièrement (X est croissant) en étant situé derrière le véhicule cible VC (X est négatif). Dans la seconde partie, le véhicule VE se décale vers la gauche (Y négatif et croissant, Rd croissant et Rg décroissant). On constate également que le conducteur a tourné son volant vers la droite afin de redresser sa course (Phi = +3.00).

La moindre variation de ces données doit être considérée avec une grande attention. C'est le rôle de la base de règles décrite au §… Avant d'obtenir ces données, plusieurs traitements sont nécessaires. Nous les décrivons ci-dessous.

2.2.3. Etapes de réalisation du système informatisé

Le projet CASSICE comporte effectivement plusieurs phases, ayant pour rôle, à chaque étape, de transformer les données issues de l'étape précédente, afin de les rendre exploitables par l'étape suivante. Ces phases sont au nombre de 3 :

- ✓ L’archivage : il consiste à récupérer en temps réel les données numériques issues des capteurs installés sur le véhicule expérimental. A ce stade, les données peuvent être incomplètes ou bruitées.
- ✓ Le traitement des données : les informations issues de la phase précédente sont traitées et fusionnées afin d'obtenir les données décrites fig. 1. Cette phase est pour l'instant en temps réel simulé. Autrement dit, elles sont traitées dans leur ordre d’arrivée, sans contrainte sur leur vitesse de traitement.
- ✓ L’interrogation et la restitution des données : la base de données est utilisée par l’expert psychologue pour obtenir les informations relatives à des épisodes de conduite suivant des critères définis. Durant cette phase, il s’agit de passer d’une représentation numérique à une représentation symbolique décrivant les manœuvres effectuées par le conducteur. Le changement de représentation se fait par une transformation sous forme d'états puis la reconnaissance à partir de ces états, des manœuvres effectuées, par les 2 niveaux de règles.

Nous décrivons dans la suite la reconnaissance de la manœuvre de dépassement, qui s'inscrit donc dans le cadre de la phase "interrogation/restitution" décrite ci-dessus.

3. Le système IDRES

Cette section présente tout d'abord synthétiquement le fonctionnement général du système IDRES, puis le premier niveau de décision permettant de déterminer les états possibles et enfin le second niveau de décision afin de deviner la manœuvre en cours.

3.1. Principes

Le système IDRES est constitué de 2 niveaux de règles. Son rôle est de reconnaître à l'intérieur d'une séquence de données, telles que celles décrites fig. 1, les manœuvres effectuées. Nous nous sommes pour le moment intéressés à la manœuvre de dépassement.

Le premier niveau du système IDRES est constituée d'une base de règles-conseils. Elles ont pour objectif de reconnaître, si possible à chaque instant, quelle étape (ou état) de la manœuvre est en cours de réalisation. Etant donné les critères retenus pour reconnaître chaque étape, plusieurs peuvent être candidates à chaque instant. Lors d'un doublement, lorsque VE est sur le point de se rabattre sur la file de VC, 2 états peuvent alors être reconnus : "fin de déboîtement" et "doublement". Le 1er niveau est donc utilisé pour associer à différents pas de temps les étapes pouvant leur correspondre. Le deuxième niveau, quant à lui, permettra de trouver parmi toutes ces hypothèses générées un scénario cohérent, c'est à dire correspondant à une manœuvre répertoriée dans le système. Les niveaux s'articulent comme indiqué ci-dessous :

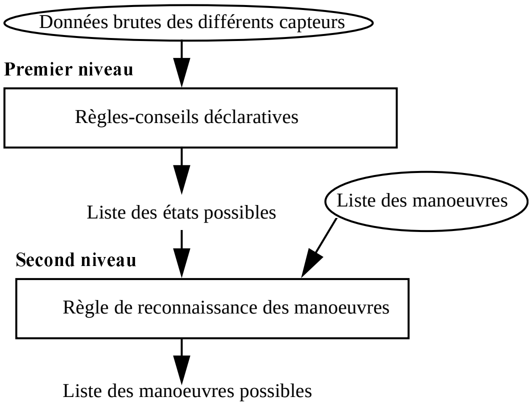


Fig. 3 : *les deux niveaux de décision*

Au niveau de la reconnaissance de la manœuvre, les principes suivants ont été adoptés :

- une manœuvre est décomposée en une séquence d'états dont il convient de respecter l'ordre de réalisation,
- certains états dans une manœuvre peuvent ne pas être reconnus. Par exemple, au cours d'un dépassement, l'état "signalement de l'intention de dépasser", correspondant à l'actionnement du clignotant, est facultatif.

- Lorsque le système n'est pas capable de déterminer un état à un instant t, il utile le concept de persistance de l'état qui a été déterminé précédemment. Cette persistance n'ai valable que durant un laps de temps bien défini.

3.2. Règles du premier niveau

Le premier niveau de décision est composée de règles-conseils chargées d'associer, pour un instant t ou un intervalle de temps, un ou plusieurs états hypothétiquement en cours de réalisation. Chacune des règles est destinée à la reconnaissance d'un état particulier. Les règles-conseils du premier niveau ont été développées en utilisant le formalisme et le moteur d'inférences CLIPS [CLIPS 98][GIARRATANO 98]. Les dix règles sont décrites ci-dessous dans un formalisme textuel :

Règle Attente_de_dépassement **Si** *VE et VC même file* *VE derrière VC* **Alors** Etat = "Attente de dépassement"	**Règle** Intention_de_dépasser **Si** *Rapprochement rapide de VE sur VC* **Alors** Etat = "Intention de dépasser"	**Règle** Début_de_déboîtement **Si** *VE et VC même file* *VE derrière VC* *Mouvement vers la gauche* **Alors** Etat = "Début de déboîtement"
Règle Franchissement_ligne_gauche **Si** *Décalage à gauche* *Franchissement de la ligne discontinue gauche* **Alors** Etat = "Franchissement de la ligne discontinue à gauche"	**Règle** Fin_de_déboîtement **Si** *File(VE) à gauche de File(VC)* *Volant à droite pendant 0.2 s* *VE derrière VC* *Mouvement vers la gauche* **Alors** Etat = "Fin de déboîtement"	**Règle** Doublement **Si** *V(VE) > V(VC)* *File(VE) à gauche de File(VC)* *VE derrière VC ou même niveau* **Alors** Etat = "Doublement"
Règle Doublement_effectué **Si** *V(VE) >= V(VC)* *File(VE) à gauche de File(VC)* *VE devant VC* **Alors** Etat = "Doublement effectué"	**Règle** Début_de_rabattement **Si** *File(VE) à gauche de File(VC)* *Volant à droite pendant 0.2 s* *Mouvement vers la droite* *VE devant VC* **Alors** Etat = "Début de rabattement"	**Règle** Franchissement_ligne_droite **Si** *Décalage à droite* *Franchissement de la ligne discontinue droite* **Alors** Etat = "Franchissement de la ligne discontinue à Droite"
Règle Fin_de_rabattement **Si** *VE et VC même file* *Mouvement vers la droite* *VE devant VC* *Volant à gauche durant 0.2 s* **Alors** Etat = "Fin de rabattement"		

Les prémisses des règles de reconnaissance des états sont construites à partir des données fournies par les différents capteurs et caméras installés dans le véhicule "prototype" STRADA. Ces dix règles sont totalement indépendantes les unes des autres et l'ajout ou le retrait d'autres règles-conseils n'entraîne aucune modification des autres règles. Grâce à cette déclarativité, plusieurs règles peuvent être déclenchées à un instant t. Ainsi, plusieurs états peuvent être valides. Le second niveau de décision du système IDRES a pour rôle de choisir parmi ces états, ceux qui permettront de valider une manœuvre.

3.3. Règles du second niveau

A partir des états générés par les règles-conseils du premier niveau, les règles du second niveau ont pour fonction de sélectionner une séquence cohérente d'états. Nous avons considéré qu'une séquence d'états était cohérente dès lors qu'elle correspond aux étapes de réalisation d'une manœuvre. Par exemple, la manière la plus simple de valider la manœuvre "changement de file à gauche" serait que les trois états qui la composent

("1- Début de déboîtement", "2- Franchissement de la ligne discontinue à gauche", "3- Fin de déboîtement") soit validée en respectant l'ordre chronologique.

Voici les trois règles de reconnaissance des manœuvres décrites dans un formalisme textuel :

Règle Initialisation_d_une_manoeuvre
Si *Un état E a été détecté entre les instants t1 et t2*
Cet état E est le premier état d'une manœuvre M
La manœuvre M n'a pas encore été reconnue
Alors *La manœuvre M est en cours entre les instants t1et t2 avec l'état E*

Règle Suite_manoeuvre_meme_etat
Si *Un état E a été détecté entre les instants t3 et t4*
Cet état E fait partie d'une manœuvre M
La manœuvre M est en cours entre les instants t1et t2 avec le même état E
Le délai entre t2 et t3 est inférieur au temps de persistance (0.5 seconde)
Alors *La manœuvre M est en cours entre les instants t1et t4 avec l'état E*

Règle Suite_manoeuvre_etat_suivant
Si *Un état E2 a été détecté entre les instants t3 et t4*
Cet état E2 fait partie d'une manœuvre M
La manœuvre M est en cours entre les instants t1et t2 avec le même état E1
L'état E1 précède l'état E2 dans la manœuvre M
Le délai entre t2 et t3 est inférieur au temps de persistance (0.5 seconde)
Alors *La manœuvre M est en cours entre les instants t1et t4 avec l'état E2*

Ces règles du second niveau de décision ont été développées indépendamment du domaine d'application. Elles respectent bien les trois "principes" énoncés dans la section 3.1. Actuellement, IDRES trouve toutes les manœuvres possibles effectuées par le véhicule expérimental. Si plusieurs manœuvres sont possibles, il ne cherchera pas à déterminer quelle est la manœuvre la plus pertinente.

Par contre, il est peut être souhaitable dans un futur proche, en cas de conflit portant sur la manœuvre à reconnaître, de considérer le type de conducteur. A un conducteur débutant, les manœuvres de petite granularité pourraient être favorisées alors que pour un conducteur expérimenté, les manœuvres plus globales seraient plus appropriées [Saad&al 96].

3.4. Résultats de IDRES

Nous avons expérimenté le système à partir de données fourni par le simulateur (cf. Fig. 1) pendant la simulation d'un dépassement. L'expérience a consisté à tester le système IDRES et à vérifier qu'il a bien reconnu la manœuvre. La figure 4 montre le résultat de la reconnaissance des états en fonction du temps. Ce résultat provient donc de l'exécution du premier niveau de décision.

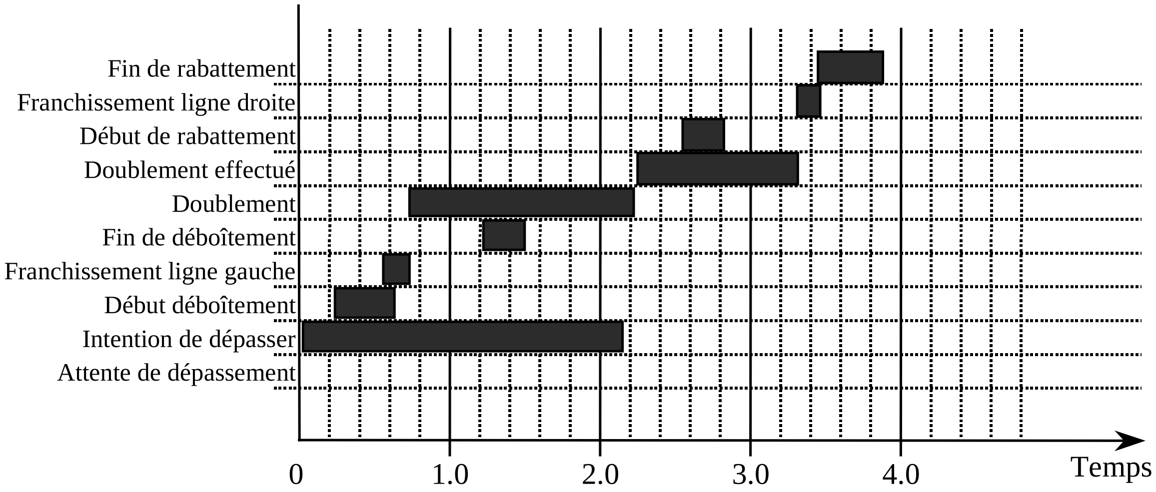


Fig. 4 : *la reconnaissance des états en fonction du temps*

Ce résultat montre bien que plusieurs états peuvent être validés à un instant donné. Par exemple, à l'instant temps=0.25 seconde les états "Attente de dépassement", "Intention de dépasser" et "début de déboîtement" sont valides. Cette particularité entraîne la nécessité d'avoir un second niveau de décision qui devra choisir le bon état pour valider une ou plusieurs manœuvres.

Ce premier résultat devient donc la matière première pour un second niveau de décision qui permettra de deviner la manœuvre du véhicule.

Afin de vérifier qu'IDRES est capable de valider une manœuvre, imaginons quatre manœuvres un changement de file à gauche, à droite, un dépassement normal et un dépassement différé (interrompu momentanément) qui sont décrites sur la figure 5.

Dépassement normal	**Changement de file à G**	**Dépassement différé**	**Changement de file à D**
1.Intention de dépasser 2.Début de déboîtement 3.Franchissement de la ligne discontinue à gauche 4.Fin de déboîtement 5.Doublement 6.Doublement effectué 7.Début de rabattement[1] 8.Franchissement de la ligne discontinue à droite 9.Fin de rabattement	1.Début de déboîtement 2.Franchissement de la ligne discontinue à G 3.Fin de déboîtement	1.Intention de dépasser 2.Début de déboîtement 3.Franchissement de la ligne discontinue à gauche 4.Fin de déboîtement 5.Insertion d'un véhicule sur la file 6.Encrage au véhicule devant 7.Rabattement du véhicule devant 8.Doublement 9.Doublement effectué 10.Début de rabattement 11.Franchissement de la ligne discontinue à droite 12.Fin de rabattement	1.Début de rabattement 2.Franchissement de la ligne discontinue à droite 3.Fin de rabattement

Fig. 5 : *Quatre manœuvres avec leurs états*

Chacune des quatre manœuvres est codées en faisant apparaître les liens de précédence entre les états qui la composent. Il faut remarquer que certaines manœuvres peuvent faire partie d'une manœuvre plus globale. C'est le cas, par exemple de la manœuvre "dépassement normale" qui contient les manœuvres "changement de file à gauche" et "changement de file à droite".

A partir de la liste des états valides fournis par le premier niveau de décision (cf. Fig. 4), la reconnaissance de la manœuvre de dépassement se fera fur et à mesure comme nous le décrit la figure 6.

---

[1] Un rabattement s'effectue toujours vers la file de droite.

Temps	**Etats possibles**	**Validation manœuvre**
0.01 - 0.22	Attente de dépassement **Intention de dépasser**	Dépassement normal
0.22 - 0.58	Attente de dépassement Intention de dépasser **Début de déboîtement**	Dépassement normal
0.58 - 0.63	Attente de dépassement Intention de dépasser Début de déboîtement **Franchissement ligne à gauche**	Dépassement normal
0.63 - 0.75	Attente de dépassement Intention de dépasser **Franchissement ligne à gauche**	Dépassement normal
0.75 - 1.21	Intention de dépasser **Doublement**	Dépassement normal
1.21 - 1.50	Intention de dépasser **Doublement** fin de déboîtement	Dépassement normal
1.50 - 2.15	Intention de dépasser **Doublement**	Dépassement normal
2.15 - 2.22	**Doublement**	Dépassement normal
2.22 - 2.23	***Doublement***	*Dépassement normal*
2.23 - 2.51	**Doublement effectué**	Dépassement normal
2.51 - 2.82	Doublement effectué **Début de rabattement**	Dépassement normal
2.82 - 3.31	Doublement effectué ***Début de rabattement***	*Dépassement normal*
3.31 - 3.32	***Début de rabattement***	*Dépassement normal*
3.32 - 3.45	**Franchissement ligne à droite**	Dépassement normal
3.45 - 3.49	Franchissement ligne à droite **Fin de rabattement**	Dépassement normal
3.49 - 3.90	**Fin de rabattement**	Dépassement normal

Fig. 6 : *tableau pour la reconnaissance de la manœuvre "Dépassement normal"*

Dans le tableau décrit ci-dessus, les états choisis pour déterminer la (les) manœuvre(s) sont écrits en gras. Les états validés grâce à la persistance sont en italique. De même, lorsque la manœuvre "Dépassement normal" a été déterminée grâce à la persistance, elle est écrite en italique.

Dans le premier intervalle de temps, l'état "intention de dépasser" est privilégié car elle est considérée comme postérieur à l'état "attente de dépassement". En ne gardant que les états choisis pour la validation de la manœuvre "Dépassement normal", la figure 7 montre l'évolution de la validation successive des états composant la manœuvre.

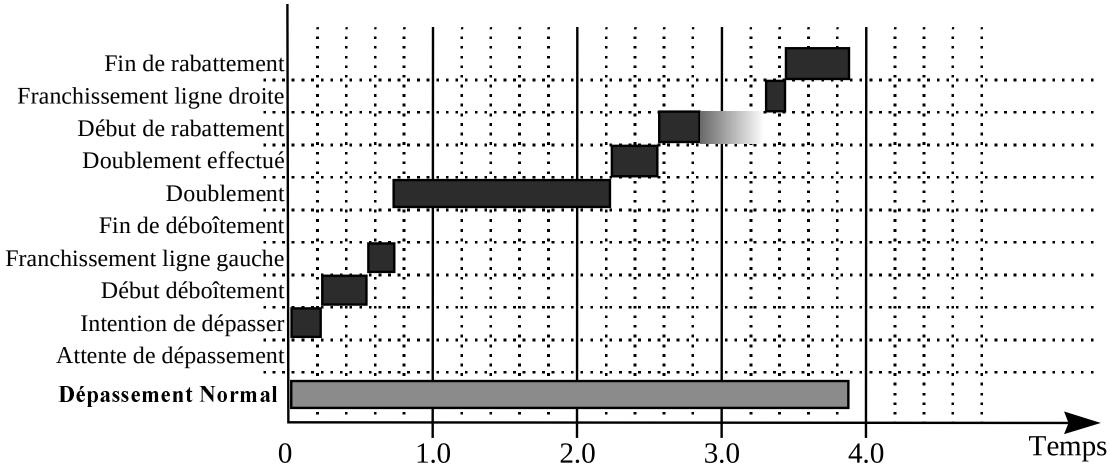


Fig. 7 : *les états utilisés pour la reconnaissance du dépassement normal*

Le système IDRES est également capable d'arrêter la reconnaissance d'une manœuvre s'il n'a pas été capable de trouver un état satisfaisant la manœuvre pendant la durée du délai de persistance (actuellement de 0,5 secondes). Les figure 8 montre l'arrêt de la reconnaissance de la manœuvre "Dépassement différé".

Temps	Etats possibles	Validation manœuvre
0.01 - 0.22	Attente de dépassement **Intention de dépasser**	**Dépassement différé**
0.22 - 0.58	Attente de dépassement Intention de dépasser **Début de déboîtement**	**Dépassement différé**
0.58 - 0.63	Attente de dépassement Intention de dépasser Début de déboîtement **Franchissement ligne à gauche**	**Dépassement différé**
0.63 - 0.75	Attente de dépassement Intention de dépasser **Franchissement ligne à gauche**	**Dépassement différé**
0.75 - 1.21	Intention de dépasser Doublement ***Franchissement ligne à gauche***	***Dépassement différé***
1.21 - 1.50	Intention de dépasser Doublement **Fin de déboîtement**	**Dépassement différé**
1.50 - 2.00	Intention de dépasser Doublement ***Fin de déboîtement***	***Dépassement différé***
2.01 -	***???***	***Annulation de la manœuvre***

Fig. 8 : *tableau pour la reconnaissance de la manœuvre "Dépassement différé"*

Entre les instants 1,50 secondes et 2.00 secondes , le système ne trouve pas d'état validant la manœuvre "Dépassement différé" mais elle valide tout de même la manœuvre en utilisant la notion de persistance sur le dernier état trouvé : "Fin de déboîtement". Après les 2.0 secondes, la persistance n'a plus d'effet, et étant donné qu'IDRES n'a toujours pas trouvé d'état validant le "Dépassement différé", la manœuvre est abandonnée.

Sur l'ensemble des données fournis par le simulateur, le système IDRES a été capable de valider les manœuvres "Dépassement normal" entre les instants 0 et 3.90 seconde ; la manœuvre "changement de file à gauche" entre les instants 0.22 et 1.50 seconde ; la manœuvre "changement de file à droite" entre les instants

2.51 et 3.90 ; et le système a annulé la reconnaissance de la manœuvre "Dépassement différé" après 2 secondes.

3.5. Perspectives de IDRES

Les règles utilisées nous permettent de reconnaître la manœuvre de dépassement. Nous considérerons d'autre manœuvres par la suite. Il est alors fort possible que certains états identifiés ici soient utiles également aux autres manœuvres. Pour pouvoir distinguer sans ambiguïté la manœuvre engagée, il nous est alors nécessaire de prendre en compte le contexte dans lequel se déroule la conduite. Ce contexte est très riche : il devrait comporter des caractéristiques relatives à l'environnement dans lequel se situe le conducteur (densité du trafic par exemple), ainsi que les intentions du conducteur. Les études réalisées par les psychologues, bien qu'elles n'aient pas jusqu'ici abouti à un modèle comportemental complet du conducteur, nous enseignent en effet beaucoup de choses sur le comportement des conducteurs suivant des caractéristiques notamment sociales. Ainsi, un jeune conducteur aura une conduite “ prudente ”, qui se traduira par peu de changements de voie. Cette connaissance *a priori* peut être très utile dans la reconnaissance des manœuvres effectuées. Un des objectifs prochains est donc d'intégrer ces connaissances dans notre système.

Une fois que le dispositif d'acquisition sera en place, nous devrons également prendre en compte l'imprécision et l'incertitude associées aux données sur lesquelles sont basés le raisonnement effectué.

Ces perspectives sont liées à la reconnaissances des manœuvres. Celles-ci, comme nous l'avons vu dans l'expérimentation effectuée, sert à étiqueter les données récupérées. Nous devrons ensuite dans le système réalisé, introduire les fonctionnalités propres aux systèmes de gestion de bases de données avancées. Nous devons notamment prévoir une interrogation de "haut niveau" de la base, ainsi que des interrogations simples. En ce qui concerne l'interrogation de haut niveau, nous envisageons d'utiliser les travaux sur la recherche d'historiques dans le cadre du raisonnement à partir de cas [Loriette 98]. Cette technique permettrait 2 choses : effectuer des recherches approchées, permettant d'accéder à des situations contenant tel ou tel enchaînement de manœuvres, par exemple, anticiper le comportement du conducteur. Cela permettrait de focaliser l'acquisition des données sur tel ou tel capteur. Le volume de données à traiter serait réduit, et les temps de traitement améliorés.

Une autre amélioration possible pour la reconnaissance du comportement du conducteur serait de deviner différents niveaux de décision d'un utilisateur comme le fait le système GénéCom [Nigro&al 98] pour engendrer des commentaires sur la façon dont l'utilisateur a pris ses décisions. En effet, un conducteur peut utiliser un niveau de décision à court terme (par exemple l'évitement d'obstacles imprévus) et un niveau de décision à long ou moyen terme (par exemple, sortir à la prochaine sortie de l'autoroute).

Grâce à la forme déclarative des règles du 1er niveau, un nouveau module pourra être développé afin d'adapter les différents paramètres pour la reconnaissance des états. Ce module contiendrait des méta-règles [Pitrat 90] d'introspection sur les règles du premier niveau et des méta-règles capable de modifier le contenu de ces règles de reconnaissance des états. Cette approche suit les principes décrits par Ta-Tao Chuang [Chuang 98] et le système SADE [Kormann 95] qui propose deux ou plusieurs niveaux de connaissances.

4. Les critiques des SE classiques.

Les générateurs de SE actuels permettent évidemment de séparer les objets qui seront manipulés (les faits) des objets qui effectueront ces manipulations (les règles). Cette distinction est bien évidemment un plus au niveau de la compréhension et de la maintenance du système. Elle permet également d'appliquer différentes stratégies dans le choix des règles à exécuter (par exemple, dernier arrivé premier déclenché ou premier arrivé premier déclenché).
Un des inconvénients des SE actuels vient de la procéduralisation des règles. Elles sont liées les unes aux autres par des faits qui font office de relation temporelle d'enchaînement. En effet, la plupart du temps, elles apparaissent en partie conclusion d'une règle et en partie condition d'une autre dans le seul but de déclencher une règle avant l'autre. Ce genre d'implémentation est totalement à éviter car elle entraîne de réelles difficultés pour la compréhension globale du système et pour la maintenance. Cette procéduralisation est

très dommageable également en Intelligence Artificielle (IA) car elle est un frein à l'utilisation de modules de contrôle, d'apprentissage ou d'explication.

Un exemple de règle procédurale est donné ci-dessous. Elle permet de compter les objets "*cube*" dont la couleur est rouge et affiche le nombre de cubes rouges trouvés.

```
Règle Initialisation
Si (action (= descriptif comptage_cube_rouge))   ;demande de comptage
Alors  (Créer compteur (= nombre 0))             ;création du compteur
       (Créer étape (= début_comptage vrai))     ;objet d'enchaînement

Règle Comptage
Si   (étape (= début_comptage vrai))          ;comptage en cours
    (compteur (= nombre ?nbr))
    (?cube (= couleur rouge))                    ;un cube rouge
Alors  (Modifier compteur nombre (+1 ?nbr))      ;+1 au compteur
       (Supprimer ?cube)                         ;enlever le cube testé

Règle Fin_de_comptage
Si  (étape (= début_comptage vrai))              ;comptage en cours
    (compteur (= numéro ?nbr)
    (non (?cube (= couleur rouge))               ;il n'y a plus de cube
Alors  (Print " il y avait " ?nbr "cubes rouges")
       (supprimer étape)                         ;enlever l'objet d'enchaînement
```

La règle initialisation se déclenche lorsqu'un fait "action" indique qu'il faut compter le nombre de cubes rouges de la base de faits. Le déclenchement de la règle permet la création d'un compteur et d'un fait (étape) permettant le déclenchement des deux règles "*Comptage*" et "*Fin_de_comptage*". Ce fait est typiquement un fait qui ne sert pas au comptage mais uniquement à contrôler l'enchaînement de règles. Il provoque donc une procéduralisation de la base de règles. La création de ce fait oblige également le concepteur à le supprimer (dans la règle "*Fin_de_comptage*"). Un autre inconvénient est l'obligation de supprimer les cubes au fur et à mesure du comptage pour qu'ils ne soient pas comptés plusieurs fois. Notons également la présence d'une prémisse négative très coûteuse en temps d'exécution.

Afin de pallier à ces inconvénients, les règles et méta-règles de contrôle décrites ci-après peuvent être utilisées. Les trois méta-règles servent à compter le nombre d'instances de n'importe quelle règle. Ces méta-règles sont générales et permettent une meilleure compréhension du système à base de règles. La syntaxe utilisée est la même pour les règles et les méta-règles.

```
Règle un_cube_rouge
Si (?cube (= couleur rouge))       ;un cube rouge existe dans la base
Alors Ne rien faire

Règle Init_comptage                ; Une méta-règle
Si (action (= comptage ?règle))
Alors
(Créer compteur nombre 0)          ;création d'un compteur
(Créer compteur règle ?règle)      ;pour la règle traitée
(Lancer le matchage sur la règle ?règle) ;lancer le moteur sur ?règle

Règle Comptage                     ; Une méta-règle
Si (compteur (= règle ?règle))
   (compteur (= nombre ?nbr))
   (?règle a été déclenchée)
Alors (modifier compteur nombre (+1 ?nbr)) ;+1 au compteur
```

Règle ***Fin_de_comptage*** ***; Une méta-règle***
Si
(?règle n'est plus déclenchable) ; il n'y a plus d'instance pour la règle
(compteur (= nombre ?nbr))
Alors *(print " il y avait " ?nbr "cubes rouges")*
(supprimer le compteur)

Les méta-règles peuvent s'appliquer à elles mêmes. Par exemple, pour connaître le nombre de règles du domaine d'application qui vont être soumises à un comptage, il suffit d'appliquer les méta-règles sur la méta-règle "*Init_comptage*".

Dans notre exemple, l'utilisation du niveau méta permet, en plus, un certain gain de temps en évitant la création puis la suppression de faits d'enchaînement. Notons également la disparition de la prémisse négative.

L'utilisation des méta-connaissances en générale et des méta-règles en particulier n'est pas nouvelle [Pitrat 90][Torsun 95]. Des systèmes tels que "*Teiresias*" [Davis 79], "*Soar*" [Rosenbloom 91], "*Sade*" [Kormann 95] ou "*GénéCom*" [Nigro&al 98] utilisent déjà des méta-règles. Mais, même si elles sont, la plupart du temps, décrites indépendamment du domaine d'application, elles restent fortement liées à une architecture construite pour réaliser une tâche bien précise.
A l'inverse, le méta-moteur n'a aucun objectif, sauf de permettre la génération et l'exécution de systèmes à base de règles et de méta-règles. De plus, il est développé indépendamment des domaines d'application. Il permet ainsi le développement de méta-règles immédiatement réutilisables pour différents domaines d'application.

Le méta-moteur permet de créer facilement des méta-règles et de les déclencher comme des règles "ordinaires". La section suivante décrit le principe de fonctionnement d'un système à base de règles utilisant un méta-moteur d'inférences.

5. Principes du méta-moteur

Le principe du méta-moteur est de simuler le fonctionnement d'un moteur d'inférences par un système à base de règles. Cette structure particulière met en œuvre au minimum quatre types de connaissances :

- La base de règles du domaine d'application,
- La base de faits du domaine d'application,
- La base de règles simulant un moteur d'inférences (le méta-moteur),
- Un moteur d'inférences compilé.

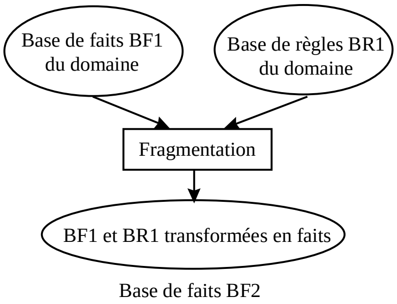


Fig. 9: *Fragmentation des règles et faits du domaine*

Les règles du méta-moteur ne peuvent être déclenchées qu'avec des faits. Puisque le méta-moteur doit déclencher les règles du domaine d'application, il faut que ces règles soient transformées en faits. La figure 9 décrit cette étape où les règles et les faits du domaine sont transformés en faits utilisables par les règles du méta-moteur. Cette étape est appelée "fragmentation".

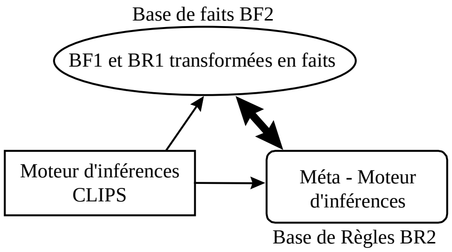


Fig. 10 : *Exécution du méta-moteur sur le SE du domaine*

Une fois la fragmentation effectuée, l'exécution peut être effectuée. Le moteur d'inférences choisi pour le déclenchement du méta-moteur est CLIPS [Clips 98][Giarratano&al 98]. L'exécution se déroule de la manière suivante (voir la figure 10) : CLIPS déclenche les règles du méta-moteur qui simule le déclenchement des règles du domaine sur les faits du domaine. Cette double interprétation permettra l'utilisation de méta-règles pouvant modifier, supprimer ou créer d'autres règles du domaine.

6. Implémentation du méta-moteur

Cette section présente de manière plus précise la construction du méta-moteur ainsi que son exécution. Le domaine d'application est tout d'abord présenté avec un exemple de règles et faits du domaine d'application. La fragmentation de ces règles et faits est ensuite proposée. Quelques règles du méta-moteur sont présentées et commentées, suivies d'un exemple d'exécution.

6.1. Domaine d'application

Le domaine d'application du méta-moteur est un système à base de règles nommé IDRES [Nigro&al 99] permettant de reconnaître en temps réel les manœuvres effectuées par un automobiliste. Ce système entre dans le cadre du projet CASSICE. Il s'appuie sur une étroite collaboration entre des psychologues, des chercheurs en imagerie, en automatique, et en IA.

Dans un premier temps, les différents développements se sont centrés sur la manœuvre de dépassement d'un véhicule. Le système développé considère qu'un dépassement peut être décomposé en dix étapes : "*attente de dépassement*", "*signalement de l'intention de dépasser*", "*début déboîtement*", "*franchissement de la ligne discontinue à gauche*", "*fin de déboîtement*", etc.). La reconnaissance de chaque étape est validée à l'aide des différents capteurs et des caméras embarqués dans le véhicule.

Le système IDRES possède deux niveaux de décision. Le premier niveau de décision génère à chaque instant des hypothèses sur le ou les étapes dans lesquels le véhicule peut se trouver. L'enchaînement des étapes correspondant à une manœuvre devant suivre un scénario, le second niveau de décision se préoccupe de reconstituer une séquence d'états cohérente vis à vis des manœuvres que le système est capable de reconnaître.

6.2. 6.2. Fragmentation des faits et règles

Le méta-moteur est un système à base de règles. Il doit donc agir sur des faits. Par conséquent, les règles et faits du domaine d'application doivent être retranscrites sous la forme de faits. Cette section présente un exemple de réécriture de la règle "*Début_de_déboîtement*" en plusieurs faits.

Règle *Début_de_déboîtement*
(?Donnée (= Y 0))   ; VE et VC même file
(?Donnée (< X 0))   ; VE derrière VC
(?Donnée (> Phi 0))   ; Volant à gauche de VE
(?Donnée (= temps ?t))
Alors
(Create (Etape temps ?t))
(Create (Etape Nom "Début déboîtement")))

Pour que le méta-moteur puisse déclencher la règle décrite ci-dessus, celle-ci doit être fragmentée en faits. Chaque fait a une syntaxe élémentaire de type "*(Objet Attribut Valeur)*". Dans la suite de l'article, le terme "objet" sera employé pour désigner un ensemble de faits (un ensemble de triplets) avec le même premier composant.

Dans un premier temps, un objet général de présentation de la règle est construit pour chacune des règles du domaine. Par exemple, l'objet "*Règle1*" (correspondant à la règle "*Début_de_déboîtement*") est formé par les six faits suivants :
(Règle1 Type Règle)
(Règle1 Règle Début_de_déboîtement)
(Règle1 Nbr_prémisses 4)
(Règle1 Liste_prémisses (Prémisse1 Prémisse2 Prémisse3 Prémisse4))
(Règle1 Nbr_actions 2)
(Règle1 Liste_actions (Action1 Action2))

Dans un second temps, chacune des prémisses de chaque règle est décomposée en un objet comportant sept faits. L'exemple suivant présente quatre objets (de "*Prémisse1*" à "*Prémisse4*") qui regroupent toutes les informations concernant les quatre prémisses de la règle "*Début_de_déboîtement*".

(Prémisse1 Type Prémisse)
(Prémisse1 Nbr_règles 1)
(Prémisse1 Liste_règles (Règle1))
(Prémisse1 Objet ?Donnée)
(Prémisse1 Opérateur =)
(Prémisse1 Attribut Y)
(Prémisse1 Valeur 0)

(Prémisse2 Type Prémisse)
(Prémisse2 Nbr_règles 1)
(Prémisse2 Liste_règles (Règle1))
(Prémisse2 Objet ?Donnée)
(Prémisse2 Opérateur <)
(Prémisse2 Attribut X)
(Prémisse2 Valeur 0)

(Prémisse3 Type Prémisse)
(Prémisse3 Nbr_règles 1)
(Prémisse3 Liste_règles (Règle1))
(Prémisse3 Objet ?Donnée)
(Prémisse3 Opérateur >)
(Prémisse3 Attribut Phi)
(Prémisse3 Valeur 0)

(Prémisse4 Type Prémisse)
(Prémisse4 Nbr_règles 1)
(Prémisse4 Liste_règles (Règle1))
(Prémisse4 Objet ?Donnée)
(Prémisse4 Opérateur =)
(Prémisse4 Attribut temps)
(Prémisse4 Valeur ?t)

Dans un troisième temps, chacune des actions des règles est également codée sous la forme d'un objet composé de sept faits. L'exemple ci-dessous présente la décomposition des deux actions de la règle "*Début_de_déboîtement*".

(Action1 Type Action)
(Action1 Nbr_règles 1)
(Action1 Liste_règles (Règle1))
(Action1 Opération Create)
(Action1 Objet Etat)
(Action1 Attribut temps)
(Action1 Valeur ?t)

(Action2 Type Action)
(Action2 Nbr_règles 1)
(Action2 Liste_règles (Règle1))
(Action2 Opération Create)
(Action2 Objet Etat)
(Action2 Attribut Nom)
(Action2 Valeur "Début déboîtement")

Les faits du domaine doivent également être retranscrits afin de pouvoir être utilisés par les règles du méta-moteur. Dans l'exemple de faits du domaine d'application présentés à la fin de la section 4.2, chacun des deux faits serait retranscrit en deux objets composés de 5 faits :

1 *(Acq1 Type Fait)*
2 *(Acq1 Temps 0.01)*
3 *(Acq1 X -32.00)*
4 *(Acq1 Y 0.00)*
5 *(Acq1 Phi 0.00)*

6 *(Acq2 Type Fait)*
7 *(Acq2 temps 0.05)*
8 *(Acq2 X -31.40)*
9 *(Acq2 Y 0.00)*
10 *(Acq2 Phi 3.00)*

Tous les faits sont numérotés automatiquement. Dans l'exemple présenté ci-dessus, les dix faits sont numérotés de 1 à 10.

6.3. Autres faits utilisés par le moteur

D'autres faits sont créés et utilisés par le méta-moteur, tel que les variables présentes dans les prémisses des règles. Ces variables jouent un rôle important pour les jointures entre les prémisses. Une écriture des variables "*?donnée*" et "*?t*" sont traduites sous la forme de faits.

(Variable1 Type Variable)
(Variable1 Variable ?donnée)
(Variable1 Règle Règle1)
(Variable1 Affectation Prémisse1)
(Variable1 Valeur (Prémisse2 Prémisse3 Prémisse4))

(Variable2 Type Variable)
(Variable2 Variable ?t)
(Variable2 Règle Règle1)
(Variable2 Affectation Prémisse4)
(Variable2 Valeur (Action1))

L'attribut "*Affectation*" décrit la prémisse qui permettra d'affecter une valeur à la variable alors que l'attribut "*Valeur*" contient une liste de prémisses qui utilisera la valeur de la variable pour une comparaison ou l'exécution d'une fonction quelconque.

Un autre objet est utilisé par le méta-moteur pour représenter les instances des règles. En effet, puisque le méta-moteur doit déclencher des règles, il doit pouvoir constituer des instances de règles.

(Instance1 Type Instance)
(Instance1 Règle Règle1)
(Instance1 Nbr_Faits 2)
(Instance1 L_Couples_NumFait_premisse (4 Prem1 3 Prem2))
(Instance1 Nbr_Variable_valeur 1)
(Instance1 L_Couples_Variable_Valeur (?Donnée Acq1))
(Instance1 Nbr_Actions_à_déclencher 2)
(Instance1 L_Actions_à_déclencher (action1 action2))
(Instance1 Etat En_construction)
(Instance1 Date_de_déclenchement Non_défini)

L'exemple présenté ci-dessus, décrit une instance en cours de construction appelée "Instance1" concernant la règle "*Règle1*" (*Début_de_Déboîtement*). Cette description de l'instance1 montre que les deux premières prémisses de la règle ont été instanciées par les faits numérotés 4 et 3. Elle indique également que le système ne connaît pour l'instant que la valeur de la variable "*?Donnée*" valant "*Acq1*".

Ce type de représentation de l'information permet au méta-moteur de gérer facilement à la fois l'instanciation des règles, l'ensemble des règles candidates au déclenchement et l'ensemble des règles déclenchées. Il permet une gestion aisée du déclenchement des règles et donc l'ajout d'un module stratégique (voir la section 5.1).

6.4. Exemple de règles du méta-moteur

Cette section décrit quatre règles du méta-moteur symbolisant les différentes étapes du méta-moteur : l'instanciation des faits sur les règles, l'ajout de la règle dans les règles candidates, le choix de la règle à déclencher et l'exécution de la règle.

```
Règle Match1
  (?fait (= type Fait))                   ; Un fait  du domaine d'appli-
  ?num ← (?fait (= ?attribut ?valeur))    ; cation de numéro ?num

  (?prémisse (= Type Prémisse))           ; "matche" avec une prémisse
  (?prémisse (= Opérateur =))
  (?prémisse (= Attribut ?attribut))
  (?prémisse (= Valeur ?valeur))
  (?prémisse (= Règle ?règle))

  (?règle (= Type Règle))                 ; d'une règle pour laquelle
  (?règle (= Nbr_actions ?nbr_actions)
  (?règle (= Liste_actions ?l_actions)

  (not (and (?Instance (= Type Instance))  ; il y a aucune instance
       (?Instance (= Règle ?règle)) ; déjà créée
Alors
  (Create (NewInstance) Type Instance)    ; Création partielle
  (Create (NewInstance) Règle ?règle)     ; d'une instance
  (Create (NewInstance) Nbr_Faits 1)
  (Create (NewInstance) L_Couples_NumFait_premisse
                           (?num ?prémisse))
  (Create (NewInstance) Nbr_Variable_valeur 0)
  (Create (NewInstance) L_Couples_Variable_Valeur ())
  (Create (NewInstance) Nbr_Actions_à_déclencher ?nbr_actions)
  (Create (NewInstance) L_Actions_à_déclencher ?l_actions)
  (Create (NewInstance) Etat En_construction)
```

Dans un premier temps, une règle de matchage simple est présentée ci-dessus. Sa partie condition gère trois types d'objets : un fait et une règle de domaine d'application avec une des prémisses de cette même règle. La règle "*Match1*" ne se déclenche que si la règle du domaine d'application n'a aucune instanciation en cours, et qu'un fait du domaine d'application "*matche*" avec une de ses prémisses. On notera que la variable "*?num*" contient le numéro du fait matchant avec la prémisse "*(?fait (= ?attribut ?valeur))*".

```
Règle Candidate1
    (?règle (= Type Règle))
    (?règle (= Nbr_prémisses ?nbr_prémisses))

    (?instance (= Type Instance))
    (?instance (= Règle ?règle))
    (?instance (= Nbr_Faits ?nbr_prémisses))
    (?instance (= Etat En_construction))
Alors (modify ?instance Etat Candidate)
```

La règle "*Candidate1*" indique qu'une règle du domaine a toutes ses prémisses satisfaites et qu'elle est donc candidate à un déclenchement.

```
Règle Stratégie_basic
    (?instance (= Type Instance))
    (?instance (= Etat Candidate))
Alors (modify ?instance Etat Prête)
```

La règle "*Stratégie_basic*" indique qu'une règle du domaine dans l'état "candidate" est prête à être déclenchée. Cette règle se déclenche sans aucune préférence entre les règles candidates. Cette étape est importante car elle permet au concepteur de remplacer la règle "*Stratégie_basic*" par un ensemble de règles stratégiques (un module stratégique – voir section 5.1) qui sélectionnera la règle à déclencher suivant des critères à définir.

```
Règle Exécution1
  (?instance (= Type Instance))
  (?instance (= Etat Prête))
  (?instance (> Nbr_Actions_à_déclencher 0))
  (?instance (= Nbr_Actions_à_déclencher ?nbr_actions))
  (?instance (INCLUDE L_Actions_à_déclencher ?action))
  (?instance (= L_Actions_à_déclencher ?l_actions))

  (?action (= Type Action))
  (?action (= Opération ?opération))
  (?action (= Objet ?objet))
  (?action (= Attribut ?attribut))
  (?action (= Valeur ?valeur))
Alors  (modify  ?instance Nbr_Actions_à_déclencher (- ?nbr_actions 1))
       (modify  ?instance L_Actions_à_déclencher (- ?l_actions ?action))
       (eval (?opération ?objet ?attribut ?valeur))
```

La règle "*Exécution1*" permet d'exécuter une action "*?action*" d'une règle prête à être déclenchée : elle décrémente le nombre d'actions restant à exécuter et enlève "*?action*" de la liste des actions à déclencher, puis exécute "*?action*".

6.5. Exemple d'exécution du méta-moteur

Cette section présente une exécution des quatre règles du méta-moteur décrites précédemment ainsi que l'évolution de certains objets au cours des différents déclenchements.

Considérons que les dix faits du domaine, numérotés de 1 à 10 à la fin de la section 4.3, ont été stockés dans la base de faits et que la règle "*Début_de_déboîtement*" a été fragmentée. La règle "*Match1*", décrite ci-après de manière instanciée, peut alors être instanciée avec les faits des objets "*Acq2*" et "*Prémisse1*" du domaine. L'instanciation de la règle "*Match1*" peut s'effectuer ainsi :

```
Règle Match1
  (Acq2 (= type Fait))                    ; L'objet Acq2 du domaine
  ?num ← (Acq2 (= Y 0))                   ; d'application. (?num = 9)

  (Prémisse1 (= Type Prémisse))           ; matche avec la prémisse1
  (Prémisse1 (= Opérateur =))
  (Prémisse1 (= Attribut Y))
  (Prémisse1 (= Valeur 0))
  (Prémisse1 (= Règle Règle1))

  (Règle1 (= Type Règle))                 ; de la Règle1 pour laquelle
  (Règle1 (= Nbr_actions 2))
  (Règle1 (= Liste_actions (Action1 Action2)))

  (not (and (?Instance (= Type Instance))     ; il y a aucune instance
        (?Instance (= Règle Règle1)) ; déjà créée
Alors (Create (Instance1 Type Instance))      ; Création partielle
       (Create (Instance1 Règle ?règle))      ; de l'instance1
       (Create (Instance1 Nbr_Faits 1))
       (Create (Instance1 L_Couples_NumFait_premisse (9 Prémisse1)))
       (Create (Instance1 Nbr_Variable_valeur 0))
       (Create (Instance1 L_Couples_Variable_Valeur ()))
       (Create (Instance1 Nbr_Actions_à_déclencher ?nbr_actions)
       (Create (Instance1 L_Actions_à_déclencher ?l_actions))
       (Create (Instance1 Etat En_Construction))
```

La règle "*Match1*" crée une instance incomplète contenant les informations sur le "*matchage*" du fait "*(Acq2 Y 0)*" sur la première prémisse de "*Début_de_déboîtement*".

Pour déclencher la règle "*Exécution1*", l'instance "*Instance1*" doit être complètement formée. Nous supposerons donc que "*Instance1*" est totalement formée : les quatre prémisses de "*Règle1*" ("*Prémisse1*", "*Prémisses2*", "*Prémisse3*" et "*Prémisse4*") sont instanciées par les quatre faits dont les numéros respectifs sont 9, 8, 10 et 7. L'objet "*Instance1*" totalement formé est alors décrit de la manière suivante :

(Instance1 Type Instance)
(Instance1 Règle Règle1)
(Instance1 Nbr_Faits 4)
(Instance1 L_Couples_NumFait_prémisse (9 Prémisse1 8 Prémisse2 10 Prémisse3 7 Prémisse4))
(Instance1 Nbr_Variable_valeur 2)
(Instance1 L_Couples_Variable_Valeur (?Donnée Acq2 ?t 0.05))
(Instance1 Nbr_Actions_à_déclencher 2)
(Instance1 L_Actions_à_déclencher (Action1 Action2))
(Instance1 Etat En_Construction)

La règle "*Candidate1*", décrite ci-après de manière instanciée, peut maintenant être déclenchée avec les objets "*Règle1*" ("*Début_de_déboîtement*") et "*Instance1*". Puisque le nombre de faits de l'instance correspond au nombre de prémisses de la règle, l'attribut "*Etat*" de l'objet "*Instance1*" devient "*Candidate*".

Règle *Candidate1*
(Règle1 (= Type Règle))
(Règle1 (= Nbr_prémisses 4))

(Instance1 (= Type Instance))
(Instance1 (= Règle Règle1))
(Instance1 (= Nbr_Faits 4))
(Instance1 (= Etat En_construction))
Alors *(modify Instance1 Etat Candidate)*

Règle *Stratégie_basic*
(Instance1 (= Type instance))
(Instance1 (= Etat Prête))
Alors
(modify Instance1 Etat Prête)

La règle "*Stratégie_basic*" est ensuite déclenchée modifiant également l'attribut "*Etat*" de l'objet "*Instance1*" avec la valeur "*Prête*".

La règle "*Exécution1*", décrite ci-après de manière instanciée, peut alors être déclenchée. Elle lance la première action de la "*Règle1*" ("*Début_de_déboîtement*") qui consiste à créer un nouveau fait "*(Etat Nom "Début déboîtement")*".

Règle *Exécution1*
(Instance1 (= Type Instance))
(Instance1 (= Etat Prête))
(Instance1 (> Nbr_Actions_à_déclencher 0))
(Instance1 (= Nbr_Actions_à_déclencher 2))
(Instance1 (INCLUDE L_Actions_à_déclencher Action1))
(Instance1 (= L_Actions_à_déclencher (Action1 Action2)))

(Action1 (= Type Action))
(Action1 (= Opération Create))
(Action1 (= Objet Etat))
(Action1 (= Attribut Nom))
(Action1 (= Valeur "Début déboîtement"))
Alors *(Modify Instance1 Nbr_Actions_à_déclencher (- 2 1))*
(Modify Instance1 L_Actions_à_déclencher (- (Action1 Action2) Action1))
(Eval (Create Etat Nom "Début déboîtement"))

Après le premier déclenchement de la règle "*Exécution1*", les faits de l'objet "*Instance1*" qui ont été modifiés sont les suivants : *(Instance1 Nbr_Actions_à_déclencher 1)*
(Instance1 L_Actions_à_déclencher (Action2))

On notera que la règle "*Exécution1*" peut à nouveau se déclencher et lancer la deuxième action de la règle du domaine d'application "*Regle1*".

7. Bilan du méta-moteur

Cette section présente un bilan du travail réalisé. Elle comporte trois sous-sections présentant les avantages, les inconvénients et les perspectives concernant la conception et le développement du méta-moteur.

7.1. Avantages du méta-moteur

Le méta-moteur présente de nombreux avantages car il est un outil permettant le développement facile de différents type de méta-régles : des règles d'introspection, de suppression de règles, de modification de règles, de création de règles. Ces méta-règles ont également l'avantage de pouvoir être déclenchées avant, pendant ou après l'exécution du système expert du domaine. Cette caractéristique permet le développement de modules de contrôle, d'apprentissage ou de d'explication.
Dans le cadre du projet CASSICE, un module de contrôle consisterait à superviser le déclenchement des règles afin de déclencher en premier les règles concluant sur la reconnaissance d'une étape (par exemple "*Début déboîtement*" ou "*Fin déboîtement*") puis les règles de reconnaissance d'une manœuvre (par exemple, "*dépassement*"). Le remplacement de la règle "*Stratégie_basic*" par les deux règles décrites ci-après permet de spécifier cette priorité sur les règles du domaine d'application.

```
Règle Stratégie1
    (?instance (= Type Instance))               ; Une règle candidate
    (?instance (= Règle ?Règle))
    (?instance (= Etat Candidate))

    (?action (= Type Action))                   ; possède une action
    (?action (INCLUDE Liste_règles ?Règle))     ; qui agit sur
    (?action (= Objet Etape))                   ; l'objet Etape
Alors (Modify ?instance Etat Prête)
```

La méta-règle "*Stratégie1*" indique qu'une règle candidate possédant, en partie action, une opération sur un objet "*Etape*" deviendra "*Prête*" à être déclenchée.

```
Règle Stratégie2
    (?instance1 (= Type Instance))
    (?instance1 (= Etat Candidate))

    (not (and (?action2 (= Type Action))
              (?action2 (INCLUDE Liste_règles ?Règle))
              (?action2 (= Objet Etape))

              (?instance2 (= Type instance))
              (?instance2 (= Règle ?Règle))
              (?instance2 (= Etat Candidate)) ))
Alors (Modify ?instance1 Etat Prête)
```

La méta-règle "*Stratégie2*" se déclenchera s'il n'y a plus de règles candidates concluant sur une action agissant sur un objet "*Etape*". Dans ce cas, les autres règles candidates sont proposées au déclenchement : l'attribut "Etat" de l'instance prend la valeur "Prête".

Un module d'apprentissage par règles est également réalisable puisque le méta-moteur a la possibilité de modifier, d'ajouter ou de supprimer des règles du domaine d'application en cours de résolution. Le méta-moteur permet donc des modifications en temps réel sur les connaissances du domaine (les faits et les règles).
Un exemple de règles d'apprentissage est présentée ci-après. Elle indique la propriété suivante : Si une règle "*R1*" conclut sur un fait "*A*" et une autre règle "*R2*" ne possède que le fait "*A*" en partie prémisse alors elles peuvent être remplacées par une nouvelle règle "*FusionR1etR2*" avec les prémisses de "*R1*" et avec les actions de "*R2*" auxquelles ont ajoute la création du fait "*A*".

Règle *R1*	**Règle *R2***	**Règle *FusionR1etR2***
Si *Prémisse1*	**Si** *le fait A présent*	**Si** *Prémisse1*
Prèmisse2	**Alors**	*Prèmisse2*
…	*Action 1*	…
PrèmisseN	*Action 2*	*PrèmisseN*
Alors	…	**Alors**
Création du fait A	*Action Q*	*Création du fait A*
		Action 1, Action 2, …, Action Q

Cet apprentissage pourrait s'écrire avec des méta-règles de la manière suivante :

```
Règle Apprentissage1
   (?R1 (= Type Règle))
   (?R1 (= Nbr_Actions 1))      ; Une seule action pour R1
   (?R1 (INCLUDE Liste_Actions ?ActionR1))
   (?R1 (= Nbr_Prémisses ?Nbr_prémissesR1)
   (?R1 (= Liste_Prémisses ?Liste_prémissesR1))

   (?ActionR1 (= Type Action))     ;Description de l'action pour R1
   (?ActionR1 (= Opération Création))
   (?ActionR1 (= Objet ?Obj))
   (?ActionR1 (= Attribut ?Att))
   (?ActionR1 (= Valeur ?Val))

   (?R2 (= Type Règle))
   (?R2 (= Nbr_Prémisses 1))  ;Une seule prémisse pour R2
   (?R2 (INCLUDE Liste_Prémisses ?PrémisseR2))
   (?R2 (= Nbr_Actions ?Nbr_actionsR2))
   (?R2 (= Liste_Actions ?Liste_actionsR2))

   (?prémisseR2 (= Type Prémisse))          ; La seule prémisse de R2
   (?prémisseR2 (= Objet ?Obj))   ; concerne le fait créé en
   (?prémisseR2 (= Attribut ?Att)) ; partie action de R1
   (?prémisseR2 (= Valeur ?Val))
Alors
   (Create (NewRegle) Type Règle)
   (Create (NewRegle) Règle (concat "Fusion" ?R1 "et" ?R2))
   (Create (NewRegle) Nbr_prémisses ?Nbr_prémissesR1)
   (Create (NewRegle) Liste_Prémisses ?Liste_prémissesR1)
   (Create (NewRegle) Nbr_Actions (+1 ?Nbr_actionsR2))
   (Create (NewRegle) Liste_Actions
                    (+ ?ActionR1 ?Liste_actionsR2))
   (Retract ?R1) ; Effacement des règle R1 et R2 sans effacer
   (Retract ?R2) ; Les prémisses et actions des deux règles
```

Un module d'explication peut être également développé. En effet, le méta-moteur donne la possibilité de scruter très facilement les différentes informations représentées sous la forme de faits, règles et instances. De même, les explications peuvent se faire en temps réel sans gêner la résolution du problème.

7.2. Inconvénients du méta-moteur

Le principal inconvénient est la relative lenteur du méta-moteur due à une double interprétation : exécution des règles du méta-moteur qui appliquent les règles du domaine (représentées sous la forme de faits) sur les faits du domaine.

Un autre inconvénient est le manque actuel de souplesse au niveau de la structure des règles. En effet, actuellement les systèmes experts du domaine ne peuvent utiliser des variables que pour des objets et des valeurs. Il n'est pas prévu, pour le moment, que le méta-moteur gère des systèmes experts du domaine d'ordre 2.

8. Conclusion

Le système IDRES est fonctionnel. Les différents niveaux de décision ont été implémentés et donne des résultats satisfaisants avec les données fournies par le simulateur. Le système de reconnaissance des manœuvres est générique : l'ajout d'une nouvelle manœuvre ne changera absolument rien au niveau reconnaissance des manœuvres ou de la gestion de la reconnaissance des états. La prochaine étape sera de confronter IDRES aux données fournies par les capteurs du véhicule expérimental lors d'une situation réelle de conduite.

Actuellement, le méta-moteur d'inférences est en cours d'implémentation. Il permet de mettre en œuvre différents types de méta-règles afin de créer, modifier ou supprimer d'autres règles. Le méta-moteur autorise également la construction de méta-règles d'introspection pour rechercher des informations dans les instances, les prémisses ou les actions des règles. Il permet également de développer des méta-règles réflexives (qui sont capables de s'appliquer à elles-même).

Le méta-moteur est en cours de réalisation. La perspective à court terme est donc de terminer son implémentation. Dans un second temps, afin d'améliorer la rapidité d'exécution du méta-moteur, il sera nécessaire de construire un méta-moteur compilé.

Parallèlement au développement d'un méta-moteur compilé, il est important de construire des modules de contrôle, de stratégie de déclenchement de règles, d'explication et d'apprentissage destinés à des systèmes d'aide à la décision comme le système ADSS [Chuang 98].

Il faut également appliquer le méta-moteur à différents domaines d'application. Ainsi, en plus du système expert IDRES développé dans le cadre du projet CASSICE, il est prévu de tester le méta-moteur dans le cadre du développement d'un système expert pour la modélisation du comportement d'un conducteur automobile en situation de conduite.

9. Bibliographie

[Clips 98] CLIPS98 : www.ghg.net/clips/download/documentation

[Davis 79] Davis R. : *Interactive transfer of expertise: Acquisition of new inference rules* - Artificial Intelligence, volume 12, 1979.

[Giarratano&al 98] Giarratano and Riley : *Expert Systems: Principles and Programming* - ISBN 0-534-95053-1, 1998.

[Kormann 95] Kormann S. : *A Meta-Level Architecture for Self-Monitoring* - workshop "On reflection and Meta-Level Architecture and their Applications in AI", IJCAI 95, Montréal, 1995.

[Loriette 98] Loriette-Rougegrez S. : *Raisonnement à partir de cas pour des évolutions spatio-temporelles de processus* - Revue internationale de géomatique "Les nouveaux usages de l'information géographique", actes des journées Cassini 1998, vol.8, n°1-2/1998, pp.207-227, 1998.

[Nigro&al 98] Nigro J.M., Ricaud P. : *A Method to Diagnose the User's Level* - Lecture Notes in Artificial Intelligence 1484, Progress in Artificial Intelligence - IBERAMIA 98, Ed. Springer Verlag, pp 361-372, 1998.

[Nigro&al 99] Nigro J.M., Loriette-Rougegrez S. : *Characterization of driving situations* - Santiago de Compostela (Espagne), MS'99 (International Conference on Modeling and Simulation), 1999.

[Pitrat 90] Pitrat J. : *Métaconnaissance - Futur de l'Intelligence Artificielle* - Hermès, 1990.

[Rosenbloom 91] Rosenbloom P.S., Laird J.E., Newell A., McCarl Robert : *A preliminary analysis of Soar architecture as a basis for general intelligence* - Artificial intelligence, volume 47, 1991.

[Saad&al 96] Saad F., Villame T. : *Assessing new driving support systems : contribution of an analysis of driver's activity in real situations* - in proceedings of 3rd annual world congress on intelligent transport systems - topic : human factors - Orlando, 1996.

[Chuang 98] Chuang T., Yadav S.B. : *The development of an adaptive decision support system* - Decision support systems, Volume 24, Issue 2, 1998.

[Torsun 95] Torsun I.S. : *Foundations of Intelligent Knowledge-Based Systems* - ed Academic Press, 1995

Un système général de jeu de cartes
Un domaine intéressant pour l'IA

Fabrice KOCIK
Laboratoire d'Informatique de Paris 6
Université Pierre & Marie Curie

Résumé: Cet article explique l'intérêt de la réalisation de systèmes généraux pour l'IA, et particulièrement ceux appliqués aux jeux de cartes. Nous soutiendrons l'idée que l'intelligence d'un système en IA ne peut être évaluée correctement lorsqu'il ne s'applique pas à une classe de problèmes mais à un problème spécifique, que les jeux sont un domaine idéal pour la réalisation d'un système général. Les jeux de cartes en particulier obligent le chercheur à réfléchir à des mécanismes autres que la combinatoire pure.

Une classification des jeux de cartes est proposée ainsi que les différentes capacités requises dans ces domaines. Les différentes étapes et directions de recherche possibles sont indiqués à travers l'explicitation des tâches qu'un système général de jeux de cartes doit savoir résoudre.

Mots-clés: Système général, IA, jeux, jeux de cartes.

1. Introduction

Le but de l'intelligence artificielle est de concevoir et réaliser des systèmes au moins aussi performants que l'être humain dans tous les domaines. Un système d'intelligence artificielle est ainsi évalué sur deux critères d'évaluation: sa performance et sa généralité. Vu l'ampleur et la difficulté de la tâche à accomplir , il est raisonnable de fixer des limites. Un système peut être conçu performant dans une tâche spécifique. Il a alors le mérite d'être applicable rapidement et une partie de la quête de l'IA est ainsi remplie en profondeur. Le concepteur du système peut aussi choisir de sacrifier un peu de performance pour y gagner dans l'étendue du domaine d'application du système. Il devra alors s'attacher à mettre en œuvre des mécanismes généraux. Les mécanismes sont certes insuffisants en eux-mêmes pour la réalisation d'un système aussi performant que l'être humain. Mais il y a fort à parier que cette progression en largeur se révélera rapidement efficace pour un vaste domaine d'application au fur et à mesure de la découverte de ces mécanismes.

Nous développons dans cet article l'idée que la réalisation d'un système général est fondamentale pour l'IA, en particulier que le vaste domaine des jeux de cartes est très intéressant pour son développement futur.

Après un bref état de l'art sur les recherches vers des systèmes généraux, ainsi que sur les jeux de cartes, nous donnerons aux lecteurs non adeptes de ces jeux, les règles des plus usuels. Nous pourrons ainsi réfléchir sur une classification des jeux de cartes pertinentes pour l'IA, et en particulier distinguer les différentes capacités et différents mécanismes mis en jeu dans chacune de ces classes.

Nous évoquerons finalement les différentes étapes nécessaires à la réalisation d'un système du jeu de la carte complet et les difficultés présentes pour sa conception.

2. Etat de l'art

La machine doit être capable de savoir réagir de la manière la plus adaptée à de nombreux problèmes, à tous ceux pour lesquels l’homme a des compétences.

Dans cette mesure, l’élaboration d’un programme qui ne met pas en évidence des mécanismes généraux comporte peu d’intérêt pour l’IA. Si au contraire, de tels mécanismes sont explicités, il n’y alors pas de raisons pour qu’ils ne soient appliqués et testés sur un ensemble de problèmes. *[PELL 94]* explique le problème d’évaluer l’avancement des recherches sur l’efficacité de programmes spécifiques à certains jeux. Est nommé « biais » l’erreur faite sur l’évaluation de tels programmes par rapport aux critères de généralité, due à la difficulté de séparer la part réelle de l’intelligence donnée par le programmeur de celle dont le programme fait preuve.

Le système Alice, *[LAURIERE 76]*, résolvait des problèmes à base de contraintes à variables discrètes. Il montra l'intérêt de donner l'énoncé du problème dans un langage déclaratif et sémantiquement fort. Il montra ainsi la possibilité de concevoir des systèmes capables de s'attaquer de manière efficace à une grande classe de problèmes, grâce d'une part, à l'utilisation de méta-choix, et, d'autre part à l'utilisation de différentes représentations pour les connaissances du système.

Compte tenu de la difficulté de réaliser des systèmes efficaces pour une classe de problèmes aussi vaste, *[PELL 94]* préconise de définir des classes de jeux intermédiaires, suffisamment importantes pour réduire le "biais", et pas trop pour que le problème reste abordable et représentatif d'un domaine.

Les jeux sont fortement étudiés depuis la naissance de l'IA : ils permettent d'isoler des problèmes relatifs à l'intelligence pour faciliter la découverte des mécanismes associés. On a vu en particulier se développer de nombreux programmes basés sur des recherches combinatoires utilisant la force brute des machines. Ces méthodes très efficaces pour de nombreux domaines sont principalement exploitables pour des jeux à informations complètes dont le nombre de coups légaux moyens par tour est faible. On assiste ainsi en 1997, pour la première fois à la victoire de *Deep Blue* sur le champion du monde d'échecs Boris Kasparov. Mais il est temps pour l'IA de dépasser ces méthodes combinatoires et de diriger ses recherches vers des problèmes pour lesquels ces méthodes ne sont pas efficaces.

Les jeux sont un terrain idéal pour l'évaluation de systèmes généraux car, contrairement à des problèmes naturels pour lesquels les lois du monde sont difficilement déterminées, il est possible de donner au système les règles du jeu qui sont en nombre restreint et facilement transmissibles. Un système général de jeu peut ainsi analyser les règles du jeu, et déduire des connaissances spécifiques au jeu proposé, qu'il devra ensuite utiliser pour le choix du bon coup au cours d'une partie. Tel est le projet initié par *[PITRAT 71]* appliqué principalement sur les jeux de plateau rectangulaire, tout particulièrement aux échecs, et que nous nous proposons d'appliquer aux jeux de cartes.

Nous prônons ici l'intérêt pour la recherche de réfléchir sur les jeux de cartes. Ces jeux sont en effet en majorité des jeux à informations incomplètes (cf *[KOLLER 97]* pour classification des jeux) rendant très difficile une approche purement combinatoire. Le domaine incite donc à se pencher sur des problèmes aussi importants et difficiles que variés tels que la représentation de ces adversaires, le bluff, l'estimation de probabilités... *[BILLINGS 98]* répertorie les capacités générales requises pour le *Hold'em poker*. Il s'applique ainsi à réaliser un système capable de "simuler" des capacités ne faisant pas appel à la force brute combinatoire, capacités utiles dans bien d'autres domaines que celui des jeux de cartes. Il n'a cependant pas le soucis de réaliser un système général.

D'autres travaux sont à citer dans le domaine des jeux de cartes:

Chelem, *[POPESCU 84]*, est un système qui utilise des connaissances déclaratives écrites en Snark pour déterminer la ligne de jeu au bridge: un grand nombre de concepts utiles au choix de la carte est dégagé.

[NIGRO 95] a réalisé un système de jeu du tarot qui donne des explications sur les coups à jouer. En début de partie, le système détermine un plan de jeu à partir d'une analyse de sa main. Les cartes sont ensuite jouées conformément à ce plan. La lacune du système réside dans son incapacité à changer de plan en cours de partie. Il montre l'intérêt de l'utilisation de la déclarativité pour le dialogue entre l'homme et la machine.

[TOMASENA 81] lit et traduit en représentation interne les règles du jeu de réussites écrites en langage naturel: on trouvera ainsi dans ses travaux une base de concepts intéressants pour un travail sur le choix de la carte.

[GAMBACK 93] se sert des annonces du bridge pour raisonner sur les connaissances pragmatiques des joueurs, comprendre l'intention de son partenaire à travers la signification cachée des annonces faites.

3. Une grande variété de jeux: quelques exemples

Les jeux de cartes sont de natures variées et en quantité impressionnante. Apparus en Europe en 1370, ils sont actuellement devenus très populaires, ce qui explique sans doute la source inépuisable d'informations qui sont accessibles à leur sujet.

Pour l'élaboration d'un système général, les jeux de cartes constituent un domaine remarquable du fait, dans un premier temps, de la grande base de jeux à disposition, et, dans un second temps, de la grande quantité d'ouvrages disponibles sur les stratégies et conseils pour les jeux les plus populaires et les plus intéressants.

Nous explicitions ici succinctement les règles de quelques jeux connus afin que les non initiés se fassent une idée du domaine. Elles ont été reprises de *[LOHEAC-AMMOUN]* dans lequel on pourra trouver les règles des autres jeux mentionnés dans l'article.

Certains habitués des jeux de cartes ne seront sans doute pas d'accord sur certaines des règles données ci-dessous ou bien nommeront différemment les jeux cités. Le nombre de jeux de cartes possibles est tel que chaque jeu comporte ses propres variantes. De plus, nous avons délibérément simplifier certains des aspects, l'enjeu ici étant une compréhension globale des jeux présentés, de leur intérêt et leur diversité.

3.1. Le rami

Matériel:.................. 2 jeux de 52 cartes
Nbre de joueurs:...... 2 à 6 joueurs
Valeur des cartes:.... As = 11 pts, figures = 10 pts, 10 = 10 pts, 9 = 9pts, 8 = 8pts, 7 = 7 pts...
Combinaisons: .... brelans, carrés, séquences (suite d'au moins 3 cartes de la même couleur)
Déroulement: .... Le donneur distribue 10 cartes à chaque joueur puis en retourne une autre qui devient le tas de défausse. Chaque joueur joue ensuite à tour de rôle, chaque coup étant constitué de 3 phases successives. La première phase, le tirage, donne au joueur la possibilité d'améliorer son jeu en prenant, ou bien la première carte de la défausse (visible), ou bien la première carte du talon (non visible). La deuxième phase, la défausse, est la phase au cours de laquelle le joueur doit rejeter une carte sur le tas des défausses. La dernière phase, la pose, permet au joueur de se débarrasser d'une partie de ses cartes en posant une combinaison ou en complétant une combinaison existante. Le jeu s'arrête lorsqu'un joueur a réussi à se débarasser de toutes ses cartes. Un malus est comptabilisé aux autre joueurs en fonction de la valeur des cartes en main en fin de partie alors qu'un bonus est donné au gagnant.

3.2. Le bridge

Voici un des jeux les plus connus et les plus difficiles, c'est pourquoi, il est source d'inspiration pour les chercheurs en intelligence artificielle.
Matériel:.................. 1 jeu de 52 cartes
Nbre de joueurs....... 4 joueurs = 2 équipes de 2 joueurs placés l'un en face de l'autre.
Force de cartes:....... As, Roi, Dame, Valet, 10, 9, 8, 7, 6, 5, 4, 3, 2.
Force des couleurs (pour les annonces): Sans Atout, Pique, Coeur, Carreau, Trèfle
Déroulement: .... Le donneur distribue 13 cartes à chaque joueur. Au cours des enchères qui suivent, chaque joueur estime le nombre de levées que son équipe sera capable de réaliser selon son jeu et les annonces des autres joueurs. Chacun, à tour de rôle, peut passer (pas d'annonce) ou s'engager à réaliser un certain nombre de levées dans une couleur en montant sur la dernière annonce faite. Le jeu commence lorsqu'aucun joueur ne désire surenchérir sur la dernière annonce. Celle-ci définit l'équipe attaquante et la couleur d'atout. Le joueur du camp attaquant qui le premier à annoncer à la couleur d'atout est le déclarant. Il controlera pendant la partie son propre jeu et celui de son partenaire, dit le mort, dont les cartes seront visibles pour tous.
Le déclarant ouvre le premier pli. L'ouvreur d'un pli pose la carte de son choix. Les joueurs suivants doivent fournir la même couleur (la couleur demandée), ou bien ,soit couper soit se défausser, s'ils n'en ont pas. Le joueur qui a mis la carte la plus forte remporte le pli et ouvre le suivant. La carte la plus forte est le plus fort atout posé, ou s'il n'y en a pas, la carte la plus forte à la couleur demandée. L'équipe attaquante encaisse d'autant plus de points que le nombre de levées annoncé est grand dans le cas où elle les a réalisées. Dans le cas contraire, elle chute.

3.3. Le poker

Matériel:.................. 1 jeu de 52 cartes
Nbre de joueurs:...... 2 à 8
But du jeu: .... posséder ou faire croire que l'on possède un combinaison de cartes supérieure aux combinaisons adverses
Ordre des cartes: ..... As au 2 (décroissant)
Les combinaisons: .. carte isolée (carte la + forte de la main), paire, double paire, brelan, quinte, couleur, full, carré, quinte flush.
Déroulement: .... 5 cartes sont distribuées à chacun des joueurs. Chaque joueur mise ensuite à tour de rôle sur leur chance de gagner: ils peuvent passer, miser en égalisant la somme la plus haute misée ou en relaçant par une somme plus importante. Cette première phase se termine lorsque tous les joueurs passent. Ceux qui ont égalisé la plus haute somme restent, les autres perdent leur mise.
Les joueurs toujours en lice peuvent alors améliorer leur main en échangeant de 1 à 4 cartes.

Les phases d'annonce reprend: chaque joueur peut passer (abandonner sa mise), égaliser ou relancer. Si la plus forte relance n'a pas été égalisée, l'auteur remporte les mises sans être obligé de montrer son jeu. Dans le cas contraire, le joueur qui a la plus forte combinaison parmi ceux qui ont égalisé la plus forte mise, remporte les gains.

3.4. Le black jack

Matériel:.................. 2 jeux de 52 cartes
But: .... se rapprocher le plus possible de 21 sans le dépasser
Valeur des cartes: ... As = 11 pts ou 1 pt, figures = 10 pts, 10 = 10 pts, 9 = 9 pts, 8 = 8 pts...
Déroulement: .... Le banquier distribue à chaque joueur (lui-même compris) 2 cartes, l'une visible pour tous, l'autre visible uniquement pour le propriétaire. Le joueur qui a un naturel (21 points exactement) le montre et se fait payer. Si le banquier a un naturel, il remporte les mises. Dans le cas contraire, tout joueur peut demander au banquier des cartes tant que sa somme ne dépasse pas 21. Le banquier est obligé de tirer une nouvelle carte si son total n'excède pas 16, et de laisser sa main inchangée dans le cas contraire. Lorsque tous les joueurs sont servis, les jeux sont découverts, et le banquier payent ceux qui ont un total supérieur à lui et remportent les mises de ceux qui ont un total inférieur.

4. Classification des jeux de cartes

Les deux principales classifications des jeux de cartes (cf *[PAGAT])* sont basées sur les mécanismes d'une part et sur les objectifs d'autre part. La première regroupe et sépare les différents jeux principalement en fonction du type des actions possibles. La seconde est basée sur le but du jeu. Ces classifications sont très directement liées aux règles du jeu et à leur formulation. Elles forment donc une base intéressante pour la définition d'un langage de représentation des règles d'un jeu de cartes. Nous préférerons ici une classification en 5 grandes catégories essentiellement déterminées par les buts implicites des jeux, c'est-à-dire sur les capacités intellectuelles ou physiques que le jeu est censé devoir mettre en application et développer.
Nous différencions les jeux de vitesse, les jeux de hasard, les jeux de mémoires, les jeux de pokers et les jeux de stratégie.

4.1. Les jeux de vitesse

Ils ne comportent pas grand intérêt pour l'IA car reposant uniquement sur les capacités physiques et réflexes du sujet.

4.2. Les jeux de hasard

Les jeux de hasards (ou jeux avec chances) sont des jeux qui présentent eux aussi a priori peu d'intérêt pour l'IA. Ils mettent en évidence cependant la nécessité de savoir évaluer des probabilités, même si les faibles capacités des êtres humains dans leur estimation montrent qu'une évaluation grossière des probabilités est suffisante dans la plupart des situations. Un système informatique peut faire la différence face à un être humain grâce à de meilleurs estimations des probabilités.
Parmi les jeux de hasards, citons bien sûr la bataille, mais aussi le black jack dont la difficulté réside uniquement dans la connaissance des probabilités.

4.3. Les jeux de mémoire

Les jeux de mémoire comportent aussi peu d'intérêt puisqu'ils se limitent à devoir se souvenir de l'emplacement des cartes auparavant découvertes. L'intérêt de ces jeux dus aux capacités limitées des être humains (sur leur mémoire à court terme), n'apparaît guère pour les machines pour lesquelles mémoriser des millions d'informations est désormais chose aisée. Ces capacités pourront cependant prendre tout leur intérêt dans des jeux plus complexes où il est nécessaire de faire un choix entre les informations à retenir et celles à oublier.

4.4. Les jeux de poker

Les jeux de poker (§ 3.3) sont des jeux à au moins 2 joueurs. Ce sont des jeux d'argent où les joueurs doivent miser en fonction de cartes visibles seulement par certains. Les joueurs qui excellent dans ces jeux sont ceux qui font preuve d'une grande aptitude à deviner le jeu de ses adversaires à travers leur comportement, leur choix présents et passés, et inversement à savoir bluffer : leur faire croire à un bon jeu lorsqu'il est mauvais (pour qu'ils se couchent), et à un mauvais quand il est bon (afin qu'ils misent beaucoup).

4.5. Les jeux de stratégie

La dernière classe de jeu est celle des jeux de stratégies comportant en fait les jeux de combinaisons et les jeux de levées.

Les deux reposent sur des choix de cartes représentant en général des ressources limitées, c'est-à-dire dont l'utilisation est limitée. Les jeux de combinaison consistent en général à élaborer un maximum de combinaisons, comme des suites, des paires, des carrés... Les seconds consistent à réaliser des levées ou plis dont le gagnant est déterminé en fonction de la force des cartes sur le tapis.

Cette classe de jeu révèle des problèmes similaires à ceux des jeux stratégiques classiques (échecs, go, awalés...) liés à la présence de concepts tels que la force, la liberté, la valeur et aussi l'avantage du trait.

Le Rami (§ 3.1), est un jeu de combinaisons, la belote, le tarot, le bridge (§ 3.2) sont des jeux de levées.

5. Capacités requises aux cartes: domaines d'étude pour l'IA

Nous abordons dans cette partie l'intérêt des jeux de cartes pour l'intelligence artificielle. Nous allons voir que les mécanismes mis en jeux touchent directement cette discipline et que ces mécanismes sont nombreux et intéressants. La figure 1 montre les mécanismes généralement utilisés dans chacune des classes de jeu définies au §4.

5.1. Modélisation du jeu - Estimation des probabilités

Il s'agit de se représenter l'état du monde en déterminant l'emplacement des cartes à l'aide de sa mémoire et de son intelligence, à savoir ici, une bonne analyse des coups joués par les autres participants.

A titre d'exemple, considérons la situation suivante au Tarot:

J1 ouvre avec le Roi ♥. J2 coupe avec le 2 d'atout.

En se référant à la règle du jeu qui stipule que le joueur doit fournir de la couleur demandée (ici ♥) si il en possède, nous déduisons que J2, n'ayant pas mis de ♥, ne peut en posséder.

L'analyse des coups n'est malheureusement dans de nombreux cas pas aussi simple et aussi nette que pour ce dernier exemple. Certaines analyses aboutiront en effet sur des conclusions plus ou moins incertaines et imprécises.

Toujours au tarot, considérons l'annonce d'une Garde faite par un joueur A. Cette annonce nous renseigne de manière imprécise sur le jeu de A. Elle signifie que A possède très probablement un bon jeu, mais pas excellent, auquel cas ce dernier aurait fait une annonce plus importante, par exemple une Garde Sans, voire une Garde Contre. A possède donc, ou bien une longue à l'atout ou beaucoup de points ou beaucoup de bouts, ou encore un mélange des trois... On ne pourra néanmoins pas dire quelles cartes il a ou il n'a pas à moins bien sûr que nous possédions les 3 bouts ainsi que beaucoup de points: il sera alors clair, à condition que le joueur soit raisonnable, qu'il n'a pas pris sans posséder un nombre minimum d'atouts.

Lorsque nous n'avons aucune information concernant la position des cartes ou que les informations sont incertaines, il est intéressant de savoir estimer la probabilité de leur emplacement et par là-même d'autres probabilités utiles au jeu.

Il est par exemple souvent utile à la belote de connaître plus ou moins précisément la probabilité qu'au moins un joueur adverse n'ait pas de la couleur demandée et qu'il ait de l'atout. On saura ainsi si ce dernier risque de couper et remporter le pli.

5.2. Modélisation des autres joueurs

Indispensable, nous l'avons signalé (§ 4.4), au poker, la capacité de modéliser les autres joueurs est utile dans de nombreux jeux. La modélisation des intentions d'un joueur peut permettre de prédire le coup qu'il va choisir. Elle permet donc de limiter la combinatoire et de deviner l'influence d'un choix à plus long terme. De plus, lorsque les connaissances et le comportement des autres joueurs sont bien modélisés, les prédictions sont plus précises et plus exactes que si on considère l'équiprobabilité de leur choix. La qualité de jeu s'en trouve nettement améliorer.

Considérons la situation suivante du jeu de l'ascenseur:

chacun des 4 joueurs J1, J2, J3 et J4 possède 2 cartes. La couleur d'atout est ♠. J1 annonce qu'il fera 1 pli (et 1 seul). J2 annonce 0 pli. J3 annonce 1 pli et J4 1 pli aussi.

La somme totale des annonces (3) dépassant le nombre total des plis qui seront réalisés, les joueurs savent qu'ils auront des difficultés à faire les plis qu'ils ont annoncé. On dit que le jeu est à qui gagne gagne par opposition à qui perd gagne.

J1 entame avec le Roi♥, J2 suit avec le 9♥. J3 a alors le choix entre l'As♠ et le 2♠, qui sont, rappelons-le tous les deux des atouts.

Pour faire le meilleur choix, J3 devra envisager les diverses situations possibles pour la main de J4 et prédire le choix de ce dernier pour chacun des cas. Dans la situation où J4 a un ♠ et un ♥ par exemple, J3 pourra faire le raisonnement suivant: si je joue le 2♠, J4, désireux de faire un pli (qui gagne gagne), jouera son atout et remportera le pli; J3 réalisera ainsi son contrat en remportant le pli suivant avec son As♠.

Si J3 n'avait pas modéliser le raisonnement de J4, il aurait considéré le cas équiprobable où J4 choisit le ♠ et celui où il choisit le ♥, ce qui l'aurait amené à la conclusion qu'il avait 1 chance sur 2 de réaliser son contrat, et 1 chance sur 2 d'échouer (en remportant 2 plis). J3 passerait donc plus de temps à envisager un plus grand nombre de situations, et ses conclusions seraient moins proches de la réalité.

Dans cet exemple, J4 est modélisé comme étant raisonnable, indépendamment de sa personnalité. Un joueur peut être représenté par des critères plus personnels, comme son ambition, son aversion au risque, son habileté au jeu. Un bon joueur prendra par exemple plus de risque face à un joueur inexpérimenté.

La modélisation d'un joueur comporte donc 2 niveaux de connaissances: des connaissances globales sur ses caractéristiques individuelles, et des connaissances sur ses choix dans une situation particulière du jeu.

5.3. Le bluff

Le bluff est l'action de mentir, ou de faire croire aux autres participants en un état du monde différent de son état réel, concernant par exemple l'emplacement probable des cartes. Au poker, par exemple, où le bluff est un outil indispensable, les joueurs sont amenés à miser haut avec une main très faible afin de forcer les autres à se coucher et abandonner ainsi leurs mises par peur de perdre plus. Au contraire, avec une main forte, un joueur de poker va souvent miser faiblement pour que les autres participants, confiants, soient désireux de miser davantage.

Dans les jeux de levées, les impasses sont des exemples de bluff: au cours d'une partie de belote, Ouest entame un pli avec le 7♦. Nord, qui possède à ♦ l'As et le Valet, peut décider de faire l'impasse, c'est-à-dire de jouer le valet. Il fait ainsi croire _ du moins laisse-t-il un doute_ à Est que l'As est dans la main de Sud: Est ne se risquera probablement pas à jouer son 10♦. La décision prise par Nord de faire l'impasse n'a bien entendu d'intérêt que s'il pense que le 10♦ est dans la main de ses adversaires, et qu'il n'y a aucune chance qu'il perde le prochain pli à ♦ en se faisant couper. Ce risque ne peut être pris que par un attaquant.

5.4. Les alliances

Certains jeux se déroulent par équipe. Citons la belote où 2 équipes de 2 sont fixés au préalable et tout au long du jeu, le tarot où les équipes sont définies à chaque partie par le désignation du preneur et de son contrat.

D'autres jeux sont propices à la constitution d'alliances qui ne découlent pas, contrairement à la constitution des équipes, directement et uniquement des règles du jeu, mais aussi de la personnalité et des traits des participants. Cet aspect de la constitution des alliances en fait un problème difficile à traiter.

Par exemple, au cours d'un jeu quelconque à 3 joueurs A, B et C, la situation ci-après se présente: le joueur A est en tête alors que les 2 autres B et C, en arrière, sont sensiblement à même hauteur. Selon si le but de B est de terminer premier (1) ou de ne pas arriver dernier (2) en fin de jeu, il décidera de s'allier plutôt avec C (1) afin de ralentir A, de le rattraper et le dépasser, ou avec A (2) pour distancer C et s'assurer ainsi une deuxième place.

La mise en place des alliances déjà délicate lorsque la parole et la négociation sont autorisés devient très subtile lorsqu'elle doit se faire de façon plus implicite comme c'est le cas dans la plupart des jeux de cartes où la transmission d'informations explicites (signes visuels, auditifs...) est prohibée.

5.5. Réflexions stratégiques

Les réflexions stratégiques, la construction de plans et leur mise en œuvre, sont utilisés dans les jeux de stratégies, à savoir les jeux de levées ou de combinaisons. Ils font en effet intervenir des concepts valables pour un grand nombre de jeux et utiles dans de nombreux problèmes de la vie courante, concepts d'un niveau d'abstraction plus élevé (avoir le trait, force des pièces ou cartes, être en attaque, en défense...). Alors que la difficulté pour l'IA de la plupart des jeux de stratégie classique (échec, go...) réside dans le grand nombre de coups légaux à chaque tour, aux cartes, elle réside essentiellement dans l'incomplétude des informations. Dans les deux cas, l'utilisation d'une force brute combinatoire est impossible. Dans les deux cas, le sujet élaborent des plans, utilise des heuristiques basées sur une analyse plus ou moins grossière du jeu.

5.6. Gestion du risque

La représentation explicite de l'inconnu dans les jeux de cartes (jeux à informations incomplètes) en font des jeux relativement adaptés à une étude sur la gestion du risque. En règle général, le perdant prend des risques pour gagner, le gagnant est plutôt prudent. A plusieurs joueurs, le choix d'une prise de risque est plus complexe, dépendant aussi des ambitions des joueurs.

	Jeux de hasard	Jeux de mémoire	Pokers	Jeux de stratégie
Modélisation du jeu - Estimation des probabilités	++	++	++	++
Modélisation des joueurs	-	-	++	+
Bluff	-	-	++	+
Les alliances	-	-	-	++
Réflexions stratégiques	-	-	-	++
Gestion du risque	+	-	++	++

Légende: - inutile + utile ++ important

Fig.1: Capacités mises en jeu dans les différentes classes des jeux de cartes

5.7. Interdépendances des capacités

La grosse difficulté pour l'élaboration d'un système de jeux de cartes accompli est l'interdépendance des capacités mises en jeu. La figure 2 résume ces dépendances.

La modélisation du plateau de jeu (l'emplacement des cartes) et l'estimation des probabilités est à la base de toute autre réflexion.

La modélisation des joueurs peut porter sur tous les types de connaissances puisqu'il s'agit d'une modélisation des connaissances des autres joueurs. Cette modélisation pour être complète nécessite donc l'existence préalable de connaissances dans tous les domaines précités.

Le bluff fait intervenir 3 aspects. Une modélisation des autres joueurs et l'estimation des probabilités sur le lieu des cartes permettent d'évaluer la situation. La décision de bluffer est ensuite une décision de prise de risque.

La gestion de risque est basée sur une étude de l'incertain (probabilités). Elle est fondamentale pour la prise de décision, qu'elle intervienne au cours d'un choix stratégique, ou qu'elle soit relative à une décision de bluffer. Elle nécessite aussi souvent, comme nous l'avons vu précédemment (§ 5.4), une modélisation préalable des alliés du fait de l'impossibilité de communication directe dans la plupart des jeux de cartes.

Les alliances sont souvent utiles pour la réalisation de buts à long terme, stratégiques.

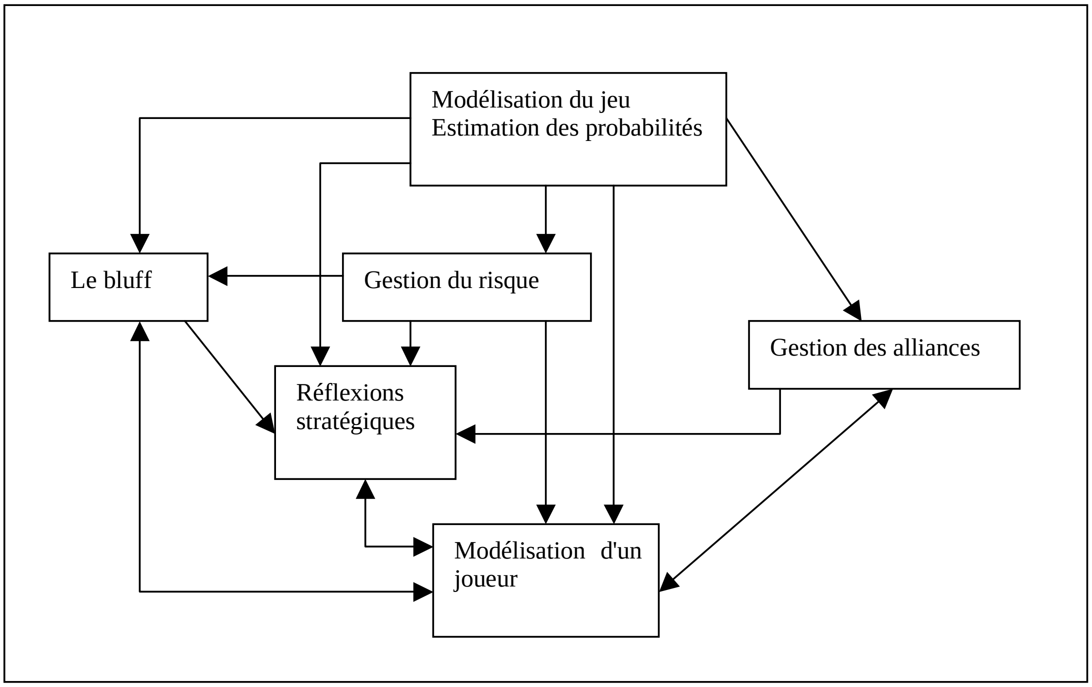


Fig.2: interdépendances des capacités requises aux jeux de cartes

6. Un système général de jeux de cartes

Nous donnons ici un aperçu des problèmes à traiter pour la réalisation d'un système général de jeux de cartes.

6.1. Etapes

La réalisation d'un système général basé sur l'étude d'un énoncé, ici les règles d'un jeu, nécessite un certain nombre d'étapes pour qu'il soit complet et entièrement autonome.

6.1.1. Un langage pour représenter les règles du jeu

Le système devant analyser les règles du jeu, il est nécessaire qu'il puisse y accéder à travers un langage adapté. Ce langage doit être suffisamment général pour accepter la définition d'une multitude de jeux, mais aussi basé sur des concepts proches du domaine pour faciliter une analyse: il n'est en effet pas nécessaire, du moins dans un premier temps, de rendre la tâche plus difficile pour le système qu'elle ne l'est pour l'être humain.
Ce langage devra être le plus déclaratif possible. D'une part, il sera lisible facilement. D'autre part, ce langage s'attachera à décrire un jeu et non pas la manière dont il faut le résoudre, cette tâche étant réservé au système.

6.1.2. Un langage pour représenter les connaissances pour bien jouer

Le système doit avoir des connaissances pour effectuer de bons choix lors du jeu. L'expression de ces connaissances sous forme déclarative facilitera leur lisibilité et leur mise au point. Ces connaissances doivent être engendrer par l'analyse des règles du jeu présentées au système, de préférence à l'aide de métarègles données elles aussi sous forme déclarative. Leur forme pourra être plus ou moins proche de celles des règles du jeu.

6.1.3. Des connaissances ou un mécanisme pour utiliser ces connaissances

Au cours du jeu, le système doit être capable d'utiliser au mieux les connaissances qu'il aura découvertes.

6.1.4. Un apprentissage

Le système pourra éventuellement être doté d'une capacité d'apprentissage au cours des parties. Cet apprentissage pourra se faire, par une première généralisation, spécifiquement sur un jeu, ou par une seconde généralisation, sur un ensemble de jeux. Les généralisations sont réalisées à partir des connaissances sur chacune des domaines.

6.2. Les connaissances

L'utilisation du symbolique pour la représentation des connaissances aux cartes s'impose. Le numérique est plus adapté à des problèmes plus bas niveau. Nous mettons en évidence ici trois aspects de ces connaissances: leur niveau de généralité, sur quoi elles portent et lorsqu'il s'agit d'expertises, ce qu'elles traitent.

6.2.1. Niveaux de généralité

Les connaissances mises en jeu dans la réalisation d'un système général ont différents degrés de généralité: elles peuvent porter sur l'ensemble des jeux de cartes, être valide pour un jeu spécifique ou encore sur une situation particulière d'un jeu particulier.

6.2.2. Sujet d'expertises

Une analyse des règles peut porter sur le plateau, le matériel, les buts. Au cours du jeu ou après, les expertises peuvent être sur le choix des autres joueurs ou la répartition connue des cartes (ex.: analyse de sa main.).

6.2.3. Objets des connaissances

Les connaissances engendrées par l'analyse des règles du jeu peuvent correspondre à une classification du jeu. En cours de jeu, les connaissances pourront être des indices sur la répartition des cartes, les autres joueurs, l'intérêt des choix.

7. Conclusion

Nous avons abordé ici le problème de la réalisation d'un système général de jeux de cartes dans sa globalité. En particulier, nous avons mis en évidence différents mécanismes "intelligents" qu'il est possible de traiter à travers ce domaine. Ces mécanismes très importants aux cartes dépassent bien entendu le cadre des jeux et apparaissent dans de nombreuses activités de la vie courante.

Nous avons décrit les principales étapes pour une réalisation d'un système complet et autonome en insistant sur la nécessité d'une analyse automatique des règles des jeux qui lui sont présentées.

Nous n'avons pas proposé de solutions détaillées pour la réalisation de ce but. Nous avons simplement dressé une architecture envisageable comme un plan de travail à moyen et long terme sans nous soucier précisément des techniques à développer tout en privilégiant une utilisation intensive de la déclarativité pour l'expression d'un maximum de connaissances..

Ce travail de synthèse sur les jeux de cartes a porté essentiellement sur le choix de la carte lui-même, bien que la réalisation d'un système général dans le domaine des cartes ait pu porter sur la génération et la découverte de nouveaux jeux intéressants ou sur l'explication des coups joués.

8. Bibliographie

Les articles concernent des travaux effectués sur les systèmes généraux, les systèmes de jeux, et plus spécifiquement sur les systèmes de jeux de cartes.

Les livres et l'adresse URL sont des références sur les jeux de carte, donnant des règles du jeu et des conseils pour le choix de la carte.

8.1. Articles

[BILLINGS 98] Billings D., Papp D., Schaeffer J., Szafron D. (1998) Poker As a TestBed for Machine Intelligence Research . AI'98, in Advances in Artificial Intelligence (Robert Mercer and Eric Neufeld, eds), Spinger Verlag, pp. 228-238, 1998.

[GAMBACK 93] Gambäck B., Rayner M., and Pell B. (1993). ``Pragmatic Reasoning in Bridge''. Technical Report 299, University of Cambridge, Computer Laboratory, Cambridge, England, April. Disponible aussi comme SRI International Technical Report CRC-030, Cambridge, England.

[KOLLER 97] Koller, D., Pfeffer, A. (1997). Representations and solutions for game-theoric problems. Artificial Intelligence 94, pp 167-215, 1997.

[LAURIERE 76] Laurière J.L. (1976). Un langage et un programme pour énoncer et résoudre les problèmes combinatoires, Thèse d'état Paris VI, 1976.

[LENAT 83] Lenat D.B. (1983), EURISKO: A program That Learns New Heuristics and Domain Concepts. AI n° 21, p 31-59 (1983)

[NIGRO 95] Nigro J.M. (1995). La conception et la réalisation d'un générateur automatique de commentaires: le système genecom. Application au jeu de Tarot. Thèse de l'université de Paris VI, pp. 51-101, 1995.

[PELL 94] Pell B. (1994). A strategic metagameplayer for general chess-like game. AAAI 1994, pp. 1378-1385, 1994.

[PITRAT 71] Pitrat J. (1971). A general game-playing program. Findler and Meltzer eds, Artificial intelligence and heuristic programming. Edinburgh University Pres. Pp 125/155, 1971.

[POPESCU 84] Popescu R. (1984). CHELEM, 1 système expert pour trouver la ligne de jeu du déclarant au bridge. Thèse de l'université P&M Curie Paris 6, 1984.

[SMITH 93] Stephen J.J Smith, Dana S. Nau (1993). Strategic Planning for Imperfect-Information Games. Games: Planning and Learning, Papers from the 1993 Fall Symposium, AAAI Press, 1993.

[TOMASENA 81] Tomasena M. (1981). Réalisation d'un système de compréhension de textes décrivant en français des règles de réussites, Thèse de 3ème cycle Université Pierre & Marie Curie, pp 13-67, 1981.

8.2. Livres

[ALBERT] Laurence Albert. Comment jouer et gagner au tarot. Règles et astuces. Eds De Vecchi.

[LOHEAC-AMMOUN] Frak Lohéac-Ammoun. Tous les Jeux de cartes. Eds SOLAR.

[MONTMIREL] François Montmirel. Poker ! Apprenez l’excellence. Eds Joker Deluxe.

8.3. adresses URL

[PAGAT] http://www.pagat.com/class/index.html#objective

Un tournoi de programmes de Phutball

Tristan Cazenave
Labo IA, Dept Informatique, Université Paris 8,
2 Rue de la Liberté, 93526 Saint-Denis, France.
cazenave@ai.univ-paris8.fr

Résumé: Pour mieux comprendre comment on découvre des concepts qui permettent à un programme de bien résoudre des problèmes complexes, nous avons tenté une expérience collective visant à écrire des programmes pour un jeu complexe inconnu de tous les participants. Chaque programme a été écrit par une équipe de quatre personnes. Chaque équipe contenait une personne chargée de noter les idées qui étaient exprimées. Le jeu concerné était le Football des philosophes ou Phutball.

Mots-clés: Jeu, Tournoi, Programme, Phutball, Découverte de concepts, Monitoring.

1. Introduction

Pour mieux comprendre comment on découvre des concepts qui permettent à un programme de bien résoudre des problèmes complexes, nous avons tenté une expérience collective visant à écrire des programmes pour un jeu complexe inconnu de tous les participants. Chaque programme a été écrit par une équipe de quatre personnes. Chaque équipe contenait une personne chargée de noter les idées qui étaient exprimées. Le jeu concerné était le Football des philosophes ou Phutball.

Le Football des philosophes est décrit dans Winning Ways [Berlekamp 1982], un ouvrage sur la théorie combinatoire des jeux [Conway 1976] appliquée à de nombreux jeux. Il a été surnommé Phutball par J. H. Conway et les auteurs de Winning Ways pensent que ce jeu ne peut pas être totalement analysé par la théorie combinatoire des jeux. Je pense au contraire que ce jeu a de bonnes chances d'être résolu en utilisant des techniques de recherches appropriées et de la démonstration de théorèmes tactique dans le style d'Introspect [Cazenave 1998], mais cela reste encore à faire…

Nous commençons par décrire les règles du jeu, puis nous explicitons les contraintes et les programmes qui ont été donnés à chacune des équipes [Cazenave 1999]. Nous décrivons ensuite les résultats de chaque équipe. Puis nous donnons certains concepts intéressants mis a jour par l'équipe Socrate. Nous tentons enfin de conclure en analysant les résultats de cette expérience.

2. Règles du jeu

Le Football des philosophes est joué sur un damier de Go de 19x19, une pierre noire représente la balle et les pierres blanches représentent les joueurs de Football. Toutes les pièces sont communes aux deux joueurs, et les deux joueurs ont les mêmes coups légaux.

La partie commence avec un damier vide, la balle est placée sur l’intersection centrale. Ensuite, à chaque coup, chaque joueur doit :

- soit poser une nouvelle pierre blanche sur une intersection vide
- soit faire sauter la balle par dessus des pierres blanches, en enlevant les pierres sautées.

Un saut peut être dans n’importe laquelle des 8 directions.

Le but du jeu est de faire parvenir la balle sur la première ligne du camp adverse, ou derrière cette première ligne.

3. Contraintes sur les Joueurs informatiques

1) Chaque joueur dispose de 5 minutes pour jouer une partie. Le premier joueur qui dépasse 5 minutes de jeu a perdu.
2) Une partie est déclarée nulle, si au bout de 200 coups aucun des joueurs n’a gagné ni n’a dépassé son temps imparti.
3) Une partie gagnée vaut 2 points, une partie nulle 1 point, et une partie perdue 0 points.
4) Chaque joueur a un nom de philosophe parmi : Socrate, Kant, Platon, Descartes, LaoTseu. Les fichiers contenant les fonctions à écrire sont les fichiers commençant par le nom du philosophe et suffixée par .cpp (fonctions c++ ou c) et .h (en-têtes).
5) L’algorithme utilisé pour choisir les coups est un Alpha-Béta simple. Chaque joueur peut régler la profondeur en utilisant la variable définie au début du fichier .cpp, par exemple int ProfondeurMaxSocrate=5; pour Socrate dans Socrate.cpp.
6) La valeur Infini est définie par 1 000 000, le fonctions d’évaluation doivent donc renvoyer des entiers entre –1 000 000 et 1 000 000.

4. Fonctions Prédéfinies

Le damier est un damier de Go de 19 intersections sur 19 intersections. Il est représenté par un tableau de 23 entiers non signés :

```
unsigned int DamierPrincipal [23] ;
```

Les fonctions suivantes sont disponibles :

```
Vide(damier,i)      => indique si l’intersection i du damier est vide
Couleur(damier,i)   => renvoie la couleur de l’intersection i parmi AMI, ENNEMI, VIDE
IntersectionBalle(damier) => renvoie l’intersection sur laquelle se trouve la balle, les
intersections varie de 0 à 360, on a en plus deux intersections particulières définies
par les constantes BUT_GAUCHE et BUT_DROIT
BordGauche(i)       => renvoie 1 si la balle est sur le bord gauche
BordDroit (i)       => renvoie 1 si la balle est sur le bord droit
BordHaut (i) => renvoie 1 si la balle est sur le bord haut
BordBas (i)         => renvoie 1 si la balle est sur le bord bas
NumeroIntersection(x,y)   => renvoie l’intersection de coordonnées x,y
Abscisse(i)         => Abscisse d’une intersection
Ordonnee (i)        => Ordonnée d’une intersection
```

On dispose aussi des trois tableaux :

```
int TableauVoisines[361][5]; int TableauDiagonales[361][5]; et
int TableauVoisinesEtDiagonales[361][9];
```

Si on veut parcourir toutes les Voisines et Diagonales d’une intersection Inter on utilise la boucle :

```
int InterProche ;
for (i=1; i<TableauVoisinesEtDiagonales[Inter][0]+1 ; i++)
   {
     InterProche=TableauVoisinesEtDiagonales[Inter][i] ;
   }
```

5. Exemple de Programme

Le programme suivant correspondant au joueur Socrate donne une idée de l’utilisation possible des primitives du jeu. La fonction à écrire pour chaque philosophe est la fonction :

```
int EvaluationSocrate(uint *Position)
```

qui renvoie l’évaluation d’une position, on peut aussi modifier la fonction qui envisage les coups possibles :

```
void TrouveCoupsPossiblesSocrate(uint *Position,int &NombrePositionsSuivantes,uint
PositionSuivante[NOMBRE_COUPS_MAX][23])
```

Voici ce que donne le programme initial de chaque philosophe :

```
#define SOCRATE_C

#include <stdio.h>
#include <iostream.h>

#include "damier.h"
#include "search.h"
#include "Socrate.h"

int ProfondeurMaxSocrate=4;
int NombreNoeudsMaxSocrate=100000;

static int TableauIntersectionsAtteignables [500];

void TrouveCoupsPossiblesSocrate(uint *Position,int &NombrePositionsSuivantes,uint
PositionSuivante[NOMBRE_COUPS_MAX][23])
  {
   int i,j,balle=IntersectionBalle(Position),inter;
   // commencer par les deplacement de la balle parce que ce sont
   // souvent des coups qui rapportent des points
   TableauIntersectionsAtteignables[0]=0;
   AjouteElementTableau(TableauIntersectionsAtteignables,balle);
   CoupsPossiblesBalleSocrate(Position,NombrePositionsSuivantes,PositionSuivante);
   for (j=1; j<TableauIntersectionsAtteignables[0]+1; j++)
      {
       inter=TableauIntersectionsAtteignables[j];
       if ((inter>=0)&&(inter<NombreIntersections))
       if (Couleur(Position,inter)==VIDE)
         {
          AffecteDamier(PositionSuivante[NombrePositionsSuivantes],Position);
          Joue(PositionSuivante[NombrePositionsSuivantes],inter,PION);
          NombrePositionsSuivantes++;
         }
       if ((inter>=0)&&(inter<NombreIntersections))
       for (i=1; i<TableauVoisinesEtDiagonales[inter][0]+1; i++)
       if (Couleur(Position,TableauVoisinesEtDiagonales[inter][i])==VIDE)
         {
          AffecteDamier(PositionSuivante[NombrePositionsSuivantes],Position);

Joue(PositionSuivante[NombrePositionsSuivantes],TableauVoisinesEtDiagonales[inter][i],PIO
N);
          NombrePositionsSuivantes++;
         }
      }
   if (NombrePositionsSuivantes>=NOMBRE_COUPS_MAX)
     cout << "debug";
  }

static int DeltaPossibles[]={1,-1,19,-19,20,-20,18,-18};
void CoupsPossiblesBalleSocrate (uint *Position,int &NombrePositionsSuivantes,uint
PositionSuivante[NOMBRE_COUPS_MAX][23])
  {
   int i,j,Delta,inter1;
   uint *PositionTemporaire;
   for (i=0; i<8; i++)
      {
       Delta=DeltaPossibles[i];
       inter1=IntersectionBalle(Position)+Delta;
       if ((inter1>-1)&&(inter1<NombreIntersections))
       if (Couleur(Position,inter1)==PION)
         {
          while (!Bord(inter1)&&(inter1>-
1)&&(inter1<NombreIntersections)&&(Couleur(Position,inter1)==PION))
            inter1+=Delta;
          // on est forcement a l'interieur du damier
          if (Couleur(Position,inter1)==VIDE)
            {
             // dupliquer la position
             PositionTemporaire=PositionSuivante[NombrePositionsSuivantes];
             AffecteDamier(PositionTemporaire,Position);
             NombrePositionsSuivantes++;
```

```
             // jouer le coup et oter les pions sautes
             JoueVide(PositionTemporaire,IntersectionBalle(PositionTemporaire));
             JoueVide(PositionTemporaire,inter1);
             for (j=IntersectionBalle(PositionTemporaire)+Delta; j!=inter1; j+=Delta)
                JoueVide(PositionTemporaire,j);
             Joue(PositionTemporaire,inter1,BALLE);
             AffecteBalle(PositionTemporaire,inter1);
             AjouteElementTableau(TableauIntersectionsAtteignables,inter1);
             // appel recursif aux coups suivants possibles

CoupsPossiblesBalleSocrate(PositionTemporaire,NombrePositionsSuivantes,PositionSuivante);
            }
          // inter1 n'est pas vide et on est donc au bord
          else if (BordGauche(inter1)||BordDroit(inter1))
            {
             // dupliquer la position
             PositionTemporaire=PositionSuivante[NombrePositionsSuivantes];
             AffecteDamier(PositionTemporaire,Position);
             NombrePositionsSuivantes++;
             // jouer le coup et oter les pions sautes
             JoueVide(PositionTemporaire,IntersectionBalle(PositionTemporaire));
             JoueVide(PositionTemporaire,inter1);
             for (j=IntersectionBalle(PositionTemporaire)+Delta; j!=inter1; j+=Delta)
                JoueVide(PositionTemporaire,j);
             if (BordGauche(inter1))
               AffecteBalle(PositionTemporaire,BUTGAUCHE);
             else if (BordDroit(inter1))
               AffecteBalle(PositionTemporaire,BUTDROIT);
            }
         }
      }
  }

int EvaluationSocrate(uint *Position)
  {
   int Eval;
   if (IntersectionBalle(Position)==BUTDROIT)
     Eval=INFINI;
   else if (IntersectionBalle(Position)==BUTGAUCHE)
     Eval=-INFINI;
   else
     Eval=Abscisse(IntersectionBalle(Position));
   if (JoueurAmi==GAUCHE)
     return Eval;
   else
     return -Eval;
  }
```

6. Les différentes équipes

6.1. Platon

Platon est le meilleur programme de Phutball de cette compétition. La fonction d'évaluation de Platon est aussi la plus simple, elle ne renvoie que trois valeurs, -INFINI lorsque la partie est perdue, +INFINI lorsqu'elle est gagnée, 0 autrement :

```
int ProfondeurMaxPlaton=6;

int EvaluationPlaton(uint *Position)
 {
  int Eval;
  if ((IntersectionBalle(Position)==BUTDROIT)||(Abscisse(IntersectionBalle(Position))==19))
   Eval=INFINI;
  else if ((IntersectionBalle(Position)==BUTGAUCHE)||(Abscisse(IntersectionBalle(Position))==1))
   Eval=-INFINI;
  else
   Eval=0;
  if (JoueurAmi==GAUCHE)
   return Eval;
```

```
  else
    return -Eval;
}
```

Cette fonction d'évaluation simpliste permet en fait de trouver un gain ou d'empêcher une perte que l'on peut prévoir 6 coups à l'avance. Elle n'est pas utilisée pour choisir les coups si un gain n'est pas en vue. L'approche de Platon est de donner directement le meilleur coup à jouer. C'est le coup qui se trouve à un saut de la pierre la plus proche du bord adverse :

```
extern int ProfondeurCourante;
void TrouveCoupsPossiblesPlaton(uint *Position,int &NombrePositionsSuivantes,uint
PositionSuivante[NOMBRE_COUPS_MAX][23])
 {
  int i,j,balle=IntersectionBalle(Position),inter;
  int trouve=0;
  // commencer par les deplacement de la balle parce que ce sont
  // souvent des coups qui rapportent des points
  if (Even(ProfondeurCourante)) {
  if (JoueurAmi==GAUCHE) {
    if (Vide(Position,balle+1))  {
      trouve=1;
      AffecteDamier(PositionSuivante[NombrePositionsSuivantes],Position);
      Joue(PositionSuivante[NombrePositionsSuivantes],balle+1,PION);
      NombrePositionsSuivantes++;  }
    for (inter=balle+1; ((inter<361)&&!trouve); inter+=1)
    if (inter<361)
    if (Vide(Position,inter)) {
      if (Abscisse(inter)<19) {
        if (Vide(Position,inter+1)) {
          trouve=1;
          AffecteDamier(PositionSuivante[NombrePositionsSuivantes],Position);
          Joue(PositionSuivante[NombrePositionsSuivantes],inter+1,PION);
          NombrePositionsSuivantes++; }}}}
  else{
    if (Vide(Position,balle-1)){
      trouve=1;
      AffecteDamier(PositionSuivante[NombrePositionsSuivantes],Position);
      Joue(PositionSuivante[NombrePositionsSuivantes],balle-1,PION);
      NombrePositionsSuivantes++;}
    for (inter=balle-1; ((inter<361)&&!trouve); inter-=1)
    if (inter>=0)
    if (Vide(Position,inter)){
      if (Abscisse(inter)>1){
        if (Vide(Position,inter-1)){
          trouve=1;
          AffecteDamier(PositionSuivante[NombrePositionsSuivantes],Position);
          Joue(PositionSuivante[NombrePositionsSuivantes],inter-1,PION);
          NombrePositionsSuivantes++;}}}}

  TableauIntersectionsAtteignables[0]=0;
  AjouteElementTableau(TableauIntersectionsAtteignables,balle);
  CoupsPossiblesBallePlaton(Position,NombrePositionsSuivantes,PositionSuivante);
  }
  else  {
  TableauIntersectionsAtteignables[0]=0;
  AjouteElementTableau(TableauIntersectionsAtteignables,balle);
  CoupsPossiblesBallePlaton(Position,NombrePositionsSuivantes,PositionSuivante);
  for (j=1; j<TableauIntersectionsAtteignables[0]+1; j++)  {
     inter=TableauIntersectionsAtteignables[j];
     if ((inter>=0)&&(inter<NombreIntersections))
     for (i=1; i<TableauVoisinesEtDiagonales[inter][0]+1; i++)
     if (Vide(Position,TableauVoisinesEtDiagonales[inter][i])){
       AffecteDamier(PositionSuivante[NombrePositionsSuivantes],Position);
       Joue(PositionSuivante[NombrePositionsSuivantes],TableauVoisinesEtDiagonales[inter][i],PION);
       NombrePositionsSuivantes++;}} } }
```

Pour arriver à ce résultat l'équipe de Platon est passé par les états suivants :

Elle a commencé par jouer contre les programmes qui étaient fournis, elle fait tout de suite deux constatations : il vaut mieux que les pions soient espacés d'une intersection plutôt que d'être contigus, et on ne peut pas bloquer l'adversaire sauf si on lui court-circuite la chaîne la plus longue vers son but. Les

questions qui se posent tout de suite sont : le minimax est il utile? La profondeur de l'Alpha-Béta peut-elle être modifiée en fonction du temps restant?
L'équipe joue ensuite avec l'ordinateur pour trouver des coups intéressants : après des sauts successifs, il est bon de manger le maximum de pierres autour de la balle tout en restant près du but, il faut empêcher l'adversaire d'arriver à l'avant-avant dernière ligne verticale…
La première idée qui vient pour évaluer une position est de maximiser la valeur nb(son camp)-nb(notre camp). Ensuite l'équipe joue avec Simplet et se rend compte qu'il les aide à gagner ! Elle décide alors de mettre l'Alpha-Béta à une petite profondeur car elle sait où jouer : on joue toujours entre la balle et le but adverse et le plus près possible de la balle. Elle teste alors cette stratégie dans une partie humain-humain, elle note alors que dans cette situation :

```
+ + O + O + O @ O + O + O + O
```

il faut manger pour se retrouver dans cette situation ou Droit ne peut plus empêcher Gauche de gagner :

```
+ @ + + + + + + O + O + O + O
```

Le temps pressant, l'équipe passe alors à la programmation. Le premier essai consiste à faire intervenir la balance des pions de façon très simple, et cela donne un mauvais programme !
La fonction d'évaluation est alors changée : les pions de notre coté compte –10 et ceux du coté de l'adversaire compte +10. La fonction d'évaluation donne la même valeur pour un coup que l'équipe souhaitait et pour un coup que l'équipe ne souhaitait pas. Au lieu de jouer du coté de son but, il préfère manger vers son camp ! Un test avec +200/-100 ne marche pas non plus. L'équipe essaie ensuite une balance pondérée et de compter les bons et les mauvais pions mais ca ne marche pas.
L'équipe décide alors d'abandonner cette approche et de faire ce qui les tentaient depuis longtemps : programmer le seul coup qui leur semble bon à chaque fois. Ils se rendent compte qu'ils connaissent le bon coup à jouer dans chaque situation mais qu'il est difficile de le faire trouver par une fonction d'évaluation numérique. La stratégie est la suivante : nous sommes dans le camp droit, nous voulons atteindre le bord gauche. Si la position immédiatement à gauche de la balle est vide : placer le joueur, sinon on cherche l'extrémité horizontale vers le but, on passe une intersection vide et on place le joueur sur l'intersection vide suivante. Après des tests, l'équipe se rend compte qu'elle ne peut pas se passer de l'Alpha-Béta à cause des coups extrêmes (ceux qui sont près du bord et qui permettent le gain). Ils remettent alors un Alpha-Béta extrêmement simpliste. Toutefois lorsque le programme envisage les coups de l'adversaire dans l'Alpha-Béta, il ne considère que le coup obligatoire ce qui l'empêche parfois de voir une parade de l'adversaire ou d'empêcher l'adversaire de gagner. Le programme est alors modifié pour que tous les coups de l'adversaire soient pris en compte. La programmation n'est pas terminée pour la compétition du soir, mais le programme est terminé plus tard dans la nuit, il contient alors correctement les idées énoncées auparavant. Platon gagne alors contre tous ses adversaires, dans le camp gauche comme dans le camp droit !

6.2. Lao-Tseu

Lao Tseu est arrivé deuxième du tournoi, lui aussi avec une fonction d'évaluation assez simple et sans recherche arborescente !

6.2.1. Voici la fonction d'évaluation de Lao-Tseu :

```
int ProfondeurMaxLaoTseu=1;

int EvaluationLaoTseu(uint *Position)
  {
   int Eval = 0;
   if(IntersectionBalle(Position)==BUTDROIT)
     Eval=INFINI;
   else if(IntersectionBalle(Position)==BUTGAUCHE)
     Eval=-INFINI;
   else {

     int abc = Abscisse(IntersectionBalle(Position));
     int ord = Ordonnee(IntersectionBalle(Position));


     if(JoueurAmi==GAUCHE) {
       Eval = abc;
       for(int i = abc+1; i <= 19; i++) {
          int inters = NumeroIntersection(i,ord);
          int j = i - abc;
          if((j % 2 == 1) && (Couleur(Position,inters) == PION)) {
```

```
           Eval += 19 - j;
        }//end if(Couleur(Position,i) == PION)
     }//end for i
     return(Eval);

    }//end if(JoueurAmi==GAUCHE)

   if(JoueurAmi==DROITE) {
     Eval = 19-abc;
     for(int i = 1; i < abc; i++) {
        int inters = NumeroIntersection(i,ord);
        int j = abc - i;
        if((j % 2 == 1) && (Couleur(Position,inters) == PION)) {
           Eval += 19 - j;
        }//end if(Couleur(Position,i) == PION)
     }//end for i
     return(Eval);
    }//end if(JoueurAmi==DROITE)

 }//end else

 if (JoueurAmi==GAUCHE)
   return (Eval);
 else
   return (-Eval);
}
```

6.2.2. Voici comment elle a été élaborée :

L'équipe a commencé par rechercher des caractéristiques de la fonction d'évaluation et les poids associés à chaque coefficient. La première idée qui leur est venue a été de compter le nombre de pierres dans chaque camp :

L'équipe a ensuite pensé qu'il était plus urgent de se déplacer que de poser des pierres. En comparant aux dames chinoises, l'équipe a ensuite noté que les pions sautés étaient enlevés et que le retour arrière était donc plus difficile. Une autre idée apparue assez tôt fut que l'on devait comptabiliser le nombre de coups légaux à partir de la position de la balle. L'équipe a ensuite décidé de faire des parties pour mieux se rendre compte de ce qui était important. En jouant, elle s'est rendue compte qu'il était intéressant de poser les pierres toutes les deux intersections, et que les cases paires par rapport à la direction de la balle devaient être privilégiées. L'équipe a ensuite pensé de nouveau qu'il était intéressant de se déplacer le plus vite possible dans la direction du but (dès que plus de deux pierres sont alignées dans cette direction). Une question est alors apparue : faut-il privilégier les parcours qui font passer chez l'adversaire pour lui prendre des pierres :

L'équipe est ensuite revenu sur cette assertion en se rendant compte qu'en début de partie, il fallait privilégier la pose des pierres par rapport au déplacement. Toutefois, si un coup amène au but de l'adversaire, il faut le jouer. L'équipe s'est alors intéressée aux stratégies de blocage : on ne peut pas empêcher un déplacement en ligne droite, mais on peut empêcher un déplacement en zigzag. Trois stratégies différentes ont donc été mises à jour : Blocage, Développement et Déplacement. Tout cela étant resté très théorique jusqu'ici, l'équipe a décidé de commencer par écrire une fonction d'évaluation qui ne prend en compte que le nombre de pierres de chaque coté de la balle. En testant cette fonction d'évaluation, les programmeurs se rendent compte que si le programme joue Gauche, il déplace la balle vers son camp de

façon à avoir le maximum de pierres à sa droite, et que s'il part vers la droite, il remet des pierres dans la partie gauche et joue alors contre son camp !
L'équipe décide alors de réfléchir aux bons coups de déplacement et aux bons coup pour poser une pierre. Elle décide alors d'enlever de la fonction d'évaluation le nombre de pierres, et plutôt de prendre en compte la position des pierres les unes par rapport aux autres en cherchant des patterns. L'accent est alors mis sur la recherche de chaînes qui permettent d'avancer. Après plusieurs idées et essais infructueux, le programme ne faisant jamais ce qu'on pensait qu'il ferait avec la stratégie implémentée, l'équipe est découragée. Elle joue alors contre le programme fournit au départ et trouve une stratégie très simple pour gagner contre lui : à chaque fois qu'il s'avance, remettre une pierre derrière lui pour s'accrocher !
Ils se rendent compte alors qu'il arrivent bien à exprimer une stratégie mais qu'ils n'arrivent pas à la programmer comme une fonction d'évaluation. Ils décident donc de ne plus utiliser l'alpha-béta et de définir directement une stratégie à profondeur 1 : si pas de pierre à droite de la balle, on augmente au maximum pour qu'elle aille vers la droite et ne cherche pas à reculer. Cette stratégie marche quand on joue en premier mais pas quand on joue en deuxième !

6.3. Socrate

Voici la fonction d'évaluation de Socrate :

```
int ProfondeurMaxSocrate=4;

int EvaluationSocrate(uint *Position)
  {
   int Eval;
   if (IntersectionBalle(Position)==BUTDROIT)
     Eval=INFINI;
   else if (IntersectionBalle(Position)==BUTGAUCHE)
     Eval=-INFINI;
  else
     {
      //if (JoueurAmi==GAUCHE)

Eval=AbscisseMaxAtteignableSocrate(Position)+AbscisseMinAtteignableSocrate(Position);
      //else
      //  Eval=AbscisseMinAtteignable(Position)-AbscisseMaxAtteignable(Position);

      Eval=Eval;//*100+Abscisse(IntersectionBalle(Position));
     }
     /*else
     {
      if (JoueurAmi==GAUCHE)
        Eval=AbscisseMaxAtteignableSocrate(Position);
      else
        Eval=AbscisseMinAtteignableSocrate(Position);

      Eval+=0*Abscisse(IntersectionBalle(Position));
     }
        */
   if (JoueurAmi==GAUCHE)
     return Eval;
   else
     return -Eval;
  }
```

Pour commencer, l'équipe de Socrate joue avec l'ordinateur et sur un damier. Elle découvre alors plusieurs concepts: l'importance du retour, il est intéressant d'avoir des pions 'couvrants' entre la balle et le but (compter le nombre de pions entre la balle peut être intéressant), elle suppose ensuite que la fonction dépendra fortement de la distance de la balle au but (cette idée sera complètement remise en cause plus tard). L'équipe se scinde alors en deux sous-groupes : un sous-groupe qui cherche à piéger l'adversaire et qui critique fortement le principe de l'Alpha-Béta, ce premier sous-groupe réfléchit sur la mémorisation des coups de l'adversaire et pense à copier ce qu'il fait, il pense ajuster des paramètres de la fonction d'évaluation en fonction de l'historique, il propose vers la fin de mettre au point un paramètre d'attaque qui augmente si on n'a pas été en danger au cours des dix derniers coups. Le deuxième groupe a une approche plus pragmatique et accepte d'utiliser l'Alpha-Béta. Il cherche donc à écrire une fonction d'évaluation simple sachant qu'il a peu de temps pour le faire. Il joue alors avec l'ordinateur et examine la fonction d'évaluation du programme fourni le plus fort. Elle essaie de régler des paramètres de cette fonction d'évaluation, mais ne pense pas toujours à optimiser le jeu à la fois quand on joue en premier et quand on joue en deuxième. Vient une idée : faire intervenir le max de la position atteignable par l'adversaire, elle utilise alors dans la fonction d'évaluation l'expression max-min+a*pos-balle. Elle se rend compte alors sur un exemple et après réflexion que l'expression qui convient à leur idée est en fait max+min+a*pos-balle. Après des tests, elle se rend compte que a=0 donne la meilleure fonction d'évaluation et bat le programme fourni le plus fort aussi bien en premier qu'en deuxième. Elle essaie alors de mettre un coefficient sur max, mais les essais ne sont pas concluants.

6.4. Kant

Voici la fonction d'évaluation de Kant :

```
int ProfondeurMaxKant=4;
#define SeuilCoupsPossibles 100
#define COEFBALLE                100
#define COEFNBCOUP               5
#define COEFORDONNEE      100
#define COEFPIERRE               20
#define AbscisseRelative(i)      ((JoueurAmi==GAUCHE)? Abscisse(i) : (19 - Abscisse(i)))

int EvaluationKant(uint *Position)
{   int Sum=0,SumOrd=0,nbpierre=0;
   int n=0,Eval=0,iold,inew,ordold;
   static uint PosSuiv[NOMBRE_COUPS_MAX][23];
    int diffabs;


       /* critere 1 */
    /*iold=IntersectionBalle(Position);
   CoupsPossiblesBalleKant(Position,n,PosSuiv);
    for (int c=0; c<n; c++)
    {  inew=IntersectionBalle(PosSuiv[c]);
       diffabs=AbscisseRelative(inew)-AbscisseRelative(iold);
       Eval+=diffabs;
    }*/

       /* critere 2 */
    if (IntersectionBalle(Position)==BUTDROIT)
    {  if (JoueurAmi==GAUCHE)
             return(INFINI);
       else
             return(-INFINI);
    }
   else if (IntersectionBalle(Position)==BUTGAUCHE)
   {   if (JoueurAmi==GAUCHE)
             return(-INFINI);
       else
             return(INFINI);
    }
   else
       Eval=COEFNBCOUP*Eval + COEFBALLE*AbscisseRelative(iold);

        /* critere 3 */
        ordold=abs(Ordonnee(iold)-10);
        Eval+=COEFORDONNEE*ordold;

       /* critere 4 */
   for (int i=0; i<361; i++)
       if (Couleur(Position,i)==PION)
        {   Sum+=AbscisseRelative(i);
             SumOrd+=abs(abs(Ordonnee(i)-10) - ordold);
             nbpierre++;
        }
   if (nbpierre)
       if (JoueurAmi==GAUCHE)
        if (AbscisseRelative(iold)>10)
             Eval+=(COEFPIERRE*(Sum + SumOrd))/nbpierre;
        else  Eval+=(COEFPIERRE*(Sum - SumOrd))/nbpierre;
       else  Eval+=(COEFPIERRE*(Sum + SumOrd))/nbpierre;

    return Eval;
}
```

L'équipe de Kant n'étant pas extrêmement fier de ses résultats, elle a préféré ne pas s'étendre en détail sur ses idées. Une des idées qu'elle a expérimentée était de maximiser le nombre de coups possible pour compliquer au maximum la situation et peut-être faire perdre l'adversaire au temps. Toutefois cette stratégie amène à des positions ou plus de 1 000 coups sont possibles sur une seule position, ce qui a fait exploser la pile que j'avais prévue pour stocker toutes les positions qui suivent une position !

6.5. Descartes

Voici la fonction d'évaluation de Descartes :

```
int ProfondeurMaxDescartes=4;

static uint PositionsTemporairesDescartes[NOMBRE_COUPS_MAX][23];

int EvaluationDescartes(uint *Position)
  {
   int Eval,entierinutile=0;
   ScoreChemins = 0 ;
   AbscisseBalleInit = Abscisse(IntersectionBalle(Position));
   LongueurCheminsDescartes(0,Position, entierinutile,PositionsTemporairesDescartes) ;
   if (IntersectionBalle(Position)==BUTDROIT)
     Eval=INFINI;
   else if (IntersectionBalle(Position)==BUTGAUCHE)
     Eval=-INFINI;
   else
     Eval = (Abscisse(IntersectionBalle(Position)))+ScoreChemins ;

   if (JoueurAmi==GAUCHE)
     return Eval;
   else
     return -Eval;
  }
```

L'équipe de Descartes n'a pas réussi a trouver d'idées qui améliore les programmes fournis. Elle a donc utilisé le meilleur programme fourni avec une profondeur incrémentée, cependant le programme résultant était souvent trop lent (temps de réponse parfois de l'ordre de la minute).

6.6. Résultats du tournoi

Tous les programmes ont rencontré tous les autres programmes, une fois en étant le joueur Gauche, une fois en étant le joueur Droit. Une partie nulle compte pour un point, et une partie gagnée compte pour deux points. Comme chaque programme a 6 opposants, il fait douze parties, le score maximum est donc de 24. Ce score a été atteint par Platon qui a gagné toutes ses parties avec Gauche et avec Droite. Les deux joueurs déjà programmés donc le code était fourni dès le début aux participants étaient Simplet et Introspect :

Platon	24
Lao-Tseu	16
Socrate	16
Introspect	11
Kant	8
Descartes	8
Simplet	1

7. Les concepts mis à jour par la suite

Jean-Yves Lucas de l'équipe Socrate a continué par la suite la mise à jour de concepts du Phutball. Pour réfléchir sur les règles du jeu, il propose de modifier les règles pour voir ce qui change dans le jeu :

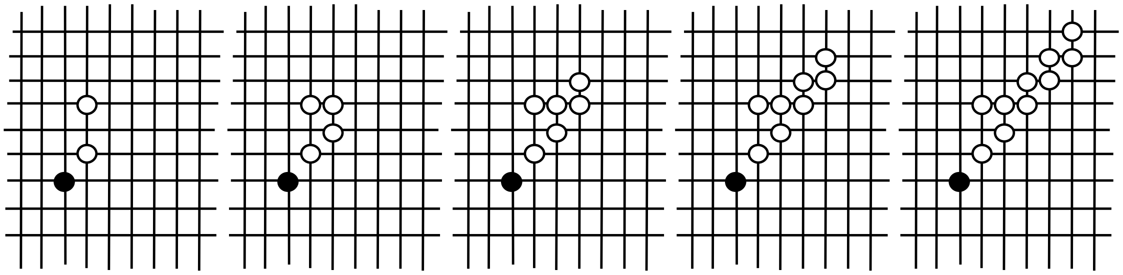
- modifier la taille du tableau de jeu.
- modifier la règle de prise.
- modifier les coups (2 coups, ôter un joueur…).
- modifier le nombre de joueurs.
- modifier l'emplacement initial de la balle.
- placer des joueurs au départ (handicap).

Il propose ensuite d'utiliser le monitoring pour améliorer le jeu, par exemple il propose de détecter par monitoring du programme par lui-même :

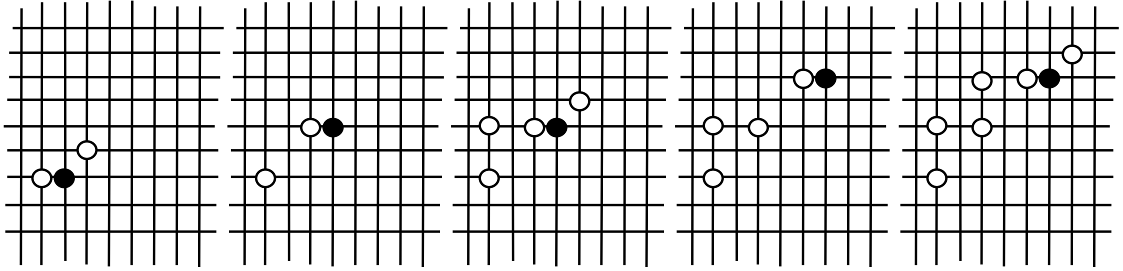
- les similitudes entre situations.
- les situations gagnantes.
- les coups qui permettent de gagner au plus vite.

Il présente alors les concepts de bord et de zone de danger.

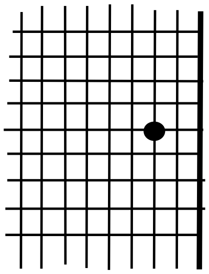
Du point de vue du monitoring du programme par lui-même, il déplore par exemple que le programme ne se rende pas compte qu'à chaque fois qu'il joue il se retrouve dans une situation quasi-identique à la précédente mais de plus en plus défavorable. L'exemple le plus flagrant est Simplet qui mange la pierre qui est posée à coté de la balle vers son camp parce qu'il pense que tous les coups sont équivalents (il joue au niveau 3 et ne peut donc pas empêcher l'adversaire de jouer ce coup) ! Jean Yves Lucas aimerait que le programme détecte qu'il va vers une perte inéluctable s'il ne change pas de stratégie. Un autre exemple est donné par un autre programme de Phutball qui continue à ajouter des pierres en diagonales dans son camp pour se ménager un retour :



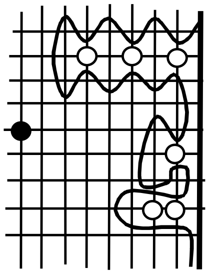
Un autre exemple de similitude défavorable rencontrée contre un programme est :



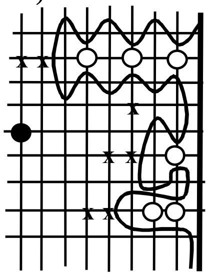
Un exemple de situation gagnante qui pourrait être déterminée directement sans calcul est quand la balle est isolée sur la troisième ligne. On peut utiliser de la démonstration de théorèmes pour trouver ces situations et les intégrer dans le programme. J'indique au passage qu'Introspect a été utilisé dans ce sens et a commencé à démontrer des théorèmes tactiques du Phutball… Exemple de situation gagnée pour Gauche quelque soit le trait :



Concept de bord : ensemble des intersections qui donnent le gain si elles peuvent être atteintes par la balle. Exemple :



Concept de zone de danger : ensemble des intersections qui permettent une suite de coups gagnants (prés du bord et pas de joueurs alentours pour s'éloigner) :



8. Conclusion

Il apparaît clairement que les meilleurs programmes sont ceux dont les équipes ont su sortir du cadre de l'Alpha-Béta, car dans ce jeu il existe une stratégie simple qui donne tout de suite des résultats, alors que la définition des concepts qui permettent d'évaluer une position est difficile. On peut noter toutefois l'exception de l'équipe Socrate qui a réussi à écrire une fonction d'évaluation simple qui a de bons résultats. Le Phutball semble donc être plus proche du Go que des Echecs, non seulement parce qu'il utilise un damier de Go mais aussi parce que les fonctions d'évaluation des coups marchent mieux que les fonctions d'évaluation des positions (ceci dans le temps très limité qui était donné aux différentes équipes pour écrire leur programme).

La motivation initiale de cette expérience était de mieux comprendre les mécanismes qui nous permettent de découvrir des concepts utiles pour programmer des résolveurs de problèmes complexes. Ce papier est un résumé de la chronologie des différentes idées qui sont apparues et de leur évaluation. C'est une expérience de terrain qui est encore bien loin d'une théorie assez formalisée pour en faire un programme, elle peut tout de même apporter un éclairage sur les méthodes à suivre ou à éviter pour chercher et trouver des heuristiques utilisables dans des programmes.

Le Phutball semble se prêter à l'utilisation de la démonstration de théorèmes tactiques qui permettraient de prévoir le gain de nombreux coups à l'avance, et peut-être de le résoudre, Introspect sera bientôt utilisé dans cette optique. Il reste aussi à tester les concepts introduits par Jean-Yves Lucas et à tester leur efficacité pour mieux jouer et pour accélérer la preuve de positions. J'ai fait une page web du Phutball pour toutes les personnes désireuses de participer et de continuer le tournoi [Cazenave 1999].

9. Bibliographie

[Berlekamp 1982] E. Berlekamp, J.H. Conway, R.K. Guy. *Winning Ways*. Academic Press, London 1982.

[Cazenave 1999] T. Cazenave, *http://www.ai.univ-paris8.fr/~cazenave/computergames.html*

[Cazenave 1998] T. Cazenave, *Metaprogramming Forced Moves.* Proceedings ECAI98,Brighton, 1998.

[Conway 1976] J. Conway, *On Numbers and Games,* Academic Press, Londres/New-York, 1976.