Reformuler et classer un problème pour le résoudre. L'architecture SYRCLAD et son application à quatre domaines

Nathalie Guin

THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS 6

Spécialité : **Informatique**
Option : **Intelligence Artificielle**

Présentée pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ PARIS 6

par

Nathalie GUIN

Sujet de la thèse :

Reformuler et classer un problème pour le résoudre

L'architecture SYRCLAD et son application à quatre domaines

soutenue le 12 décembre 1997, devant le jury composé de :

Mme Hélène GIROIRE	Codirecteur
Mme Monique GRANDBASTIEN	Rapporteur
M Jacques PITRAT	Directeur
M Jean-Charles POMEROL	Président
M François RECHENMANN	Rapporteur
M Marc ROGALSKI	Examinateur
M Gérard TISSEAU	Codirecteur

Remerciements

Le cours de DEA de Jacques Pitrat est une formidable motivation pour commencer une thèse. Je le remercie de m'avoir accueillie au sein de l'équipe métaconnaissances. Je lui suis reconnaissante de m'avoir proposé un sujet de thèse intéressant et de m'avoir aidée à chaque fois qu'un choix entre plusieurs directions de recherche se posait.

Je remercie Hélène Giroire et Gérard Tisseau qui ont suivi mon travail depuis mon stage de DEA. Je leur suis particulièrement reconnaissante de l'aide qu'ils m'ont apportée dans la rédaction de ce mémoire. Leurs relectures attentives et leurs critiques constructives ont grandement aidé à en améliorer le contenu et la forme.

Je remercie Monique Grandbastien d'avoir accepté de rapporter sur mon travail de thèse. Ses questions m'ont permis de préciser les enseignements à tirer de ce travail et les perspectives de recherche envisageables dans le domaine des EIAO.

Je remercie François Rechenmann d'avoir bien voulu être rapporteur de ma thèse. Je lui suis reconnaissante de l'intérêt qu'il a porté à mon travail, compte tenu de sa grande connaissance des travaux sur la résolution de problèmes par classification.

Jean-Charles Pomerol a su m'intéresser à l'intelligence artificielle dès la maîtrise. Je lui suis reconnaissante de m'avoir accueillie au LAFORIA où j'ai pu effectuer cette thèse dans de bonnes conditions de travail. Je le remercie d'avoir accepté de présider mon jury de thèse.

Je remercie Marc Rogalski d'avoir bien voulu participer à mon jury de thèse. Ses travaux sur l'enseignement de méthodes ont motivé une partie de mon travail et je le remercie de ses questions et des conseils qu'il m'a prodigués.

Ce travail n'aurait pu exister sans les "experts" des domaines auxquels j'ai appliqué SYRCLAD. Je remercie le groupe Combien? : Jacques Duma, Hélène Giroire, Françoise Le Calvez, Gérard Tisseau et Marie Urtasun, de m'avoir fait profiter, dans une ambiance toujours chaleureuse, de leur expérience dans le domaine des dénombrements. Je remercie Jean-Marc Nigro de m'avoir expliqué en détail le fonctionnement d'une partie de Bateleur. Je lui suis reconnaissante d'avoir toujours répondu avec beaucoup de gentillesse à mes questions de débutante sur le jeu du Tarot. Je remercie Gérard Tisseau de m'avoir permis d'utiliser la base de connaissances du résolveur de Modélis. Il m'a beaucoup aidé dans mon travail par sa connaissance du domaine de la thermodynamique.

Je remercie Monique Baron pour la disponibilité et la gentillesse dont elle a su faire preuve en me faisant bénéficier de sa grande culture en intelligence artificielle et dans le domaine des EIAO. Je lui suis reconnaissante d'avoir relu attentivement une partie de ce mémoire. Merci à Michel Pintado d'avoir eu le courage de reprendre UBL pour le tester sur la thermodynamique. Je remercie également

Élisabeth Delozanne, Évelyne Cauzinille-Marmèche et André Didierjean pour les discussions enrichissantes que nous avons eu.

Je remercie Odile Palies et toute son équipe pédagogique, qui m'ont permis d'effectuer mon enseignement dans d'excellentes conditions. Merci également à Ghislaine Mary, Anne Bancel, Jacqueline Le Baquer, Valérie Mangin, Andrée Musial et Primalli Payet, qui m'ont toujours accueillie et aidée avec beaucoup de gentillesse.

Merci à l'équipe métaconnaissances, qui m'a souvent écoutée faire le point sur mon travail. Je remercie également tous les thésards du LAFORIA, qui ont su créer au laboratoire une ambiance de travail amicale.

Enfin, merci à Pierre de m'avoir aidée à trouver le joli nom de SYRCLAD !

Plan

Introduction

Le thème de la résolution de problème a été abordé très tôt en Intelligence Artificielle et de nombreuses méthodes ont été proposées, aussi bien générales (*generate and test*, *means-ends analysis*, recherche dans des graphes) que spécifiques à des domaines particuliers (systèmes experts). La majorité d'entre elles se focalisent sur l'activité consistant à trouver un chemin menant à la solution à partir des données initiales et des opérateurs légaux (règles, théorèmes, actions). Nous soutenons ici l'idée que d'autres activités entrent en jeu lors de la résolution de problèmes et qu'il est avantageux d'en tenir compte pour réaliser un système résolveur.

Nous nous intéressons à des problèmes scolaires, à un niveau de connaissance donné, et dans des domaines où les énoncés des problèmes correspondent à des situations concrètes, alors que le cours est donné sous forme de connaissances théoriques. Les élèves sont en difficulté face à ce type de problèmes parce qu'ils n'arrivent pas à modéliser ces situations afin qu'elles entrent dans le cadre du cours théorique. Un résolveur de problème destiné à être utilisé par un EIAO dans ces domaines doit tenir compte de ces difficultés. Il faut en effet distinguer les connaissances destinées à modéliser le problème des connaissances du domaine destinées à résoudre un problème modélisé.

Pour construire un nouveau modèle du problème posé, auquel on pourra appliquer les connaissances de résolution du cours, une méthode consiste à utiliser une classification de problèmes pour guider cette reformulation du problème. En effet, on peut essayer de reformuler le problème afin de construire un problème équivalent qui soit instance d'un schéma de problème connu, auquel on sait associer des méthodes de résolution. Pour utiliser ce procédé, il faut d'abord disposer d'un répertoire de schémas, ensuite être capable de construire une formulation du problème suivant le point de vue d'un schéma.

Avantages de la démarche

Certains systèmes experts utilisent effectivement une stratégie de résolution fondée sur une classification des problèmes du domaine, mais celle-ci reste implicite la plupart du temps. Certaines règles ont pour but de reformuler le problème dans les termes de la classification, d'autres servent à reconnaître une classe donnée, d'autres encore appliquent des méthodes de résolution adaptées à chaque classe. Mais il n'existe pas toujours de représentation explicite des classifications, ni des différents processus qui entrent en jeu.

Exprimer de manière déclarative une classification et les connaissances pour l'exploiter présente plusieurs avantages, aussi bien pour l'expression des connaissances que pour leur utilisation :

Tout d'abord, cela permet d'expliciter une démarche de résolution à un plus haut niveau d'abstraction et de structuration qu'un ensemble de règles, même regroupées en paquets. Les classes de problèmes correspondent en effet à des *concepts* et le fait de les représenter explicitement permet d'en parler, de les relier à d'autres concepts et d'effectuer des raisonnements à leur sujet. De plus, le

fait de séparer la résolution d'un problème en différents *processus* généraux (modélisation, classement[1], application d'une méthode) permet de mieux distinguer les démarches et les types de difficultés propres à chaque processus. L'abstraction possède également l'avantage de permettre une réutilisation plus aisée des connaissances liées à la classification.

Ensuite, expliciter une classification permet, grâce à la déclarativité, de faire évoluer l'expertise plus facilement et avec moins de risques d'erreurs. Cette évolution peut consister d'une part à ajouter de nouvelles classes au graphe de classification, et d'autre part à enrichir des classes déjà définies.

Finalement, cette organisation peut être utile dans des applications à but pédagogique pour évaluer et commenter une solution fournie par un élève, donner des explications et planifier une progression. Un système destiné à enseigner une stratégie de résolution fondée sur une classification des problèmes doit avoir accès à une représentation explicite de la classification et des processus mis en jeu. Il pourra ainsi, par exemple, distinguer une erreur de modélisation d'une erreur d'application d'une procédure ou d'une erreur d'association entre une classe et une procédure.

Réalisations

Le but de cette thèse est de proposer, d'une part, un cadre général dans lequel on peut expliciter de manière déclarative une classification de problèmes et les connaissances de reformulation et de résolution qui y sont liées et, d'autre part, un mécanisme d'exploitation de ces connaissances. Nous avons réalisé le système SYRCLAD (SYstème de Résolution de problèmes basé sur une CLAssification du Domaine) qui met en oeuvre ces divers aspects et nous l'avons testé sur quatre domaines d'application.

SYRCLAD construit un nouveau modèle du problème posé grâce à des connaissances de reformulation et en utilisant un graphe de classification des problèmes du domaine. Ce nouveau modèle est plus opérationnel et est instance d'une classe de problèmes du graphe. À certaines classes de problèmes sont associées des connaissances de résolution propres à résoudre les problèmes de la classe. Ces connaissances de résolution sont utilisées pour obtenir la solution au problème posé.

En utilisant l'architecture de SYRCLAD, nous avons mis au point quatre applications. Le premier domaine étudié lors de notre DEA a été celui des exercices de dénombrement au niveau de la terminale scientifique, ce qui a donné lieu à la réalisation d'un système expert [Guin et al 95]. Une réflexion sur le fonctionnement de ce système et sur les améliorations à y apporter nous a permis de concevoir l'architecture de SYRCLAD. Par la suite, nous avons testé SYRCLAD sur le domaine des problèmes additifs proposés à l'école primaire, en nous basant sur une classification sur papier établie par des didacticiens. Nous avons ensuite étudié comment le système Bateleur [Nigro 95] choisit un plan de jeu au tarot, pour expliciter une classification cachée dans des règles. Nous avons utilisé cette classification pour appliquer SYRCLAD au choix d'un plan de jeu au tarot. Enfin, nous avons étudié la

[1] Nous appelons *classification* une hiérarchie de problèmes et *classement* le processus qui utilise cette classification pour classer un problème donné.

résolution d'exercices de thermodynamique par le système Modélis [Tisseau 90] dans le but d'élaborer une classification. Nous avons fourni la classification ainsi construite et des connaissances de reformulation et de résolution à SYRCLAD pour obtenir un résolveur de problèmes de thermodynamique.

Guide de lecture

La **première partie** de ce mémoire est consacrée à la description du système SYRCLAD.

Le premier chapitre présente un exemple complet de résolution d'un exercice de dénombrement par SYRCLAD. Il a pour objectif de donner au lecteur une première idée de ce que fait le système, de montrer les difficultés rencontrées lors de la résolution d'un tel problème et d'indiquer les stratégies que nous avons choisies de donner à SYRCLAD pour surmonter ces difficultés.

Le deuxième chapitre présente d'une part des travaux de psychologie cognitive et de didactique qui ont motivé et justifié la réalisation de SYRCLAD et d'autre part des travaux d'intelligence artificielle utilisant des classifications, qui seront comparés à SYRCLAD.

Le troisième chapitre présente l'architecture de SYRCLAD et le fonctionnement de son noyau, qui sont indépendants de tout domaine d'application. Il détaille quelles sont les bases de connaissances que l'expert doit donner à SYRCLAD : connaissances de classification, de reformulation et de résolution, en précisant sous quelle forme il doit les donner au système. Il présente également comment le système utilise ces connaissances pour construire un nouveau modèle du problème afin de le résoudre.

> **Certains paragraphes** ouvrent une discussion sur un choix effectué dans la conception et l'implémentation de SYRCLAD ou détaillent un élément d'implémentation. Il seront présentés dans ce style. Ils ne sont pas nécessaires à la compréhension globale de ce que fait le système.

La **deuxième partie** du mémoire présente les quatre applications de SYRCLAD qui ont abouti à la réalisation de quatre résolveurs de problèmes. Les quatre chapitres peuvent être lus de manière indépendante, mais toutefois après la lecture de la première partie. Le lecteur pourra ainsi faire un choix suivant ses centres d'intérêt. Le domaine des dénombrements (chap. 4) est peut-être le plus représentatif des capacités du système SYRCLAD, mais ceux des problèmes additifs (chap. 5) et de la thermodynamique (chap. 7) sont également révélateurs de ses capacités. Par contre, le domaine du tarot (chap. 6) ne permet qu'une exploitation limitée des possibilités du système SYRCLAD et le chapitre correspondant est une illustration qui ne suffit pas à donner un éclairage suffisant sur SYRCLAD.

La **troisième partie** du mémoire présente des travaux effectués transversalement aux quatre domaines et après implémentation de ces systèmes. Le chapitre 8 présente une étude de la complexité de ces quatre domaines, la complexité d'un domaine étant définie par rapport à l'architecture de SYRCLAD. Le chapitre 9 présente comment le système SYRCLAD décompose un problème en sous-

problèmes afin de le résoudre. Il détaille comment le système compare un problème qu'il ne sait pas résoudre aux classes du graphe de classification, pour déterminer en quoi il est différent des problèmes que le système sait directement résoudre. SYRCLAD construit ainsi des sous-problèmes à résoudre pour que le problème donné puisse entrer dans la classification. Cette méthode a été appliquée pour les problèmes additifs et la thermodynamique.

Enfin, une **conclusion** résume les aspects originaux et spécifiques de SYRCLAD pour la résolution de problèmes, et évoque des perspectives de recherche.

Les **annexes** présentent essentiellement les bases de connaissances fournies au système dans chacun des domaines d'application.

Partie I : le système SYRCLAD

La première partie de ce mémoire est consacrée à la description du système SYRCLAD.

Le premier chapitre présente un exemple complet de résolution d'un exercice de dénombrement par SYRCLAD. Il a pour objectif de donner au lecteur une première idée de ce que fait le système, de montrer les difficultés rencontrées lors de la résolution d'un tel problème et d'indiquer les stratégies que nous avons choisies de donner à SYRCLAD pour surmonter ces difficultés.

Le deuxième chapitre présente d'une part des travaux de psychologie cognitive et de didactique qui ont motivé et justifié la réalisation de SYRCLAD et d'autre part des travaux d'intelligence artificielle utilisant des classifications qui seront comparés à SYRCLAD.

Le troisième chapitre présente l'architecture de SYRCLAD et le fonctionnement de son noyau, qui sont indépendants de tout domaine d'application. Il détaille quelles sont les bases de connaissances que l'expert doit donner à SYRCLAD : des connaissances de classification, de reformulation et de résolution, et sous quelle forme il doit les donner au système. Il présente également comment le système utilise ces connaissances pour construire un nouveau modèle du problème afin de le résoudre.

Chapitre 1 :
Résolution d'un premier problème par SYRCLAD

Ce chapitre présente un exemple complet de résolution d'un exercice par SYRCLAD. Il a pour objectif de donner au lecteur une première idée de ce que fait le système. Nous avons choisi pour cela un exercice de dénombrement. Nous essaierons de montrer les difficultés rencontrées lors de la résolution d'un tel problème. Nous indiquerons les stratégies que nous avons choisies de donner à SYRCLAD pour surmonter ces difficultés.

Plan du chapitre

Nous allons étudier le problème suivant (désigné par m10) :

Combien peut-on former de mots de cinq lettres contenant exactement deux voyelles et deux "b" ?

Il n'est pas nécessaire d'avoir une grande culture mathématique pour aborder un problème de dénombrement tel que celui-ci. L'énoncé est compréhensible, et on peut envisager une résolution de manière intuitive.

On peut tenir le raisonnement suivant : « Je dois avoir deux voyelles dans le mot, j'ai 6 voyelles possibles, ce qui fait 6 * 6 possibilités. Je dois aussi avoir 2 b. Il reste une lettre à choisir, qui ne doit être ni une voyelle, ni un b, ce qui me laisse 19 possibilités. On peut donc former 6 * 6 * 19 = 684 mots. » Cette résolution a été produite de manière intuitive, mais elle est fausse, car on n'a pas bien analysé l'énoncé. En effet, cette résolution ne tient pas compte de la place des lettres dans le mot.

Avant de penser à produire une solution, il faut être sûr d'avoir bien compris quel est le problème à résoudre. Il s'agit dans le cas des dénombrements de décrire l'ensemble à dénombrer. On observe donc l'énoncé du problème afin de repérer des éléments d'information pertinents pour la résolution.

On observe que l'on choisit des lettres parmi les 26 lettres de l'alphabet. Nous appellerons cet ensemble *l'ensemble où l'on choisit*. Nous pouvons distinguer dans cet ensemble des *catégories* : celle des voyelles, que nous savons être de taille 6, et celle formée par la lettre b. On choisit ces lettres pour former un mot. Nous l'appellerons *l'objet formé*. Les mots sont des listes de lettres, il faut donc tenir compte de l'ordre des lettres dans le mot. Il est aussi important de savoir si l'on peut utiliser plusieurs fois la même lettre. On doit respecter des *contraintes* sur le mot ainsi formé en plaçant deux voyelles et deux b.

Nous avons repéré quelles sont dans ce problème les instances des concepts que sont l'ensemble où l'on choisit, les catégories, l'objet formé, les contraintes. Nous avons ainsi reformulé le problème posé :

Combien peut-on former de listes (*objets formés*) de taille 5 dont les éléments appartiennent à l'alphabet (*ensemble où l'on choisit*) et qui vérifient les *contraintes* suivantes : contenir exactement deux occurrences dans la *catégorie* des voyelles et deux occurrences dans la *catégorie* {b} ?

Il faudra donner à SYRCLAD des connaissances de reformulation. Cette reformulation nous a permis de mieux comprendre le problème à résoudre et nous permet de produire une solution plus précise. Nous avons effectivement noté qu'il faut tenir compte des places des lettres dans le mot. Nous allons donc distinguer les différentes places possibles pour les voyelles et la lettre "b".

Après avoir reformulé le problème, on peut produire une nouvelle solution : « Je vais d'abord placer une voyelle, j'ai 6 voyelles possibles, et j'ai 5 façons de la placer. Je dois ensuite placer une deuxième voyelle, j'ai toujours 6 possibilités pour la choisir, mais je n'ai plus que 4 places disponibles. A présent, je dois placer un b, il y a 3 places possibles, puis un deuxième b en

choisissant sa place parmi deux. J'ai enfin 19 lettres possibles pour remplir la dernière place disponible. Il y a donc 6*5*6*4*3*2*19 = 82080 possibilités. ».

Malgré l'analyse de l'énoncé que nous avons faite, cette réponse est fausse. En effet en considérant les deux b (ou les deux voyelles) chacun leur tour, on a compté plusieurs fois certaines configurations. Le mot "bombe" peut être construit de deux manières différentes avec le plan que nous avons donné, alors qu'il ne faut le compter qu'une seule fois. Nous devons donc choisir deux places pour les b, puis les remplir, puis choisir deux places pour les voyelles, puis les remplir.

Nous avons modélisé ce problème de manière correcte, mais notre solution était fausse parce que nous manquions de connaissances de résolution. En effet, modéliser le problème permet de voir celui-ci comme une instance d'une classe de problèmes, mais cela ne suffit pas : il faut également savoir comment se résout un problème de cette classe. Il faudra donner à SYRCLAD des connaissances de résolution.

Pour résoudre des exercices de dénombrement, il n'est pas nécessaire de maîtriser des outils mathématiques tels que le calcul formel. Il n'y a pas de définitions abstraites à assimiler. La difficulté vient plutôt de la complexité des situations rencontrées. Le problème est de reconnaître une situation connue et de savoir comment la résoudre. Il faut pour cela utiliser une méthode. Sans méthode, il est difficile de savoir si un raisonnement est juste ou faux. En situation scolaire, il est fréquent que plusieurs élèves proposent des raisonnements différents conduisant à des résultats différents, sans qu'aucun n'arrive à convaincre les autres de la justesse de son point de vue. La méthode que nous proposons consiste à utiliser un répertoire de classes de problèmes pour chacune desquelles on dispose de connaissances rigoureuses menant à la solution. Ces classes sont organisées hiérarchiquement en une classification. La démarche consiste à reconnaître d'abord à quelle classe correspond le problème, puis à appliquer les connaissances associées à la classe. Une classification des principaux problèmes rencontrés pour le niveau qui nous intéresse permet de recenser des situations connues sans mobiliser de connaissances théoriques inconnues des élèves de ce niveau. Il faut alors reconnaître dans un problème une de ces situations, ce qui demande des connaissances de reformulation, car on ne reconnaît pas une situation telle qu'elle est formulée, mais au contraire on la reformule (en conservant l'équivalence des problèmes) pour la rapprocher d'une situation connue. Si cette classification a été établie dans un but de résolution, on sait comment résoudre un problème que l'on a réussi à classer. Cela peut permettre de résoudre une grande variété de problèmes avec une méthode rigoureuse. Voyons à présent comment SYRCLAD utilise une classification de problèmes pour résoudre le problème qui nous intéresse.

1 Le modèle descriptif du problème

Les exercices ne sont pas donnés à SYRCLAD en langue naturelle, mais nous avons choisi de donner au système un modèle du problème le plus proche possible de l'énoncé en langage naturel. Ce modèle décrit une situation concrète, qui est celle présentée dans l'énoncé. Il contient des traits de surface du problème considéré : ce sont des éléments concrets tels que les jetons, les cartes à jouer. Ils forment l'aspect contextuel du problème, que l'on appelle aussi encodage linguistique. Un même

problème peut être présenté sous plusieurs habillages concrets qui ont des traits de surface différents. Nous avons choisi d'appeler ce modèle : *modèle descriptif* du problème.

Dans SYRCLAD, qui est implémenté en Prolog, le modèle descriptif d'un problème est exprimé dans un langage de la logique du premier ordre. Ce modèle descriptif est composé d'un ensemble de propositions représentant les données fournies par l'énoncé du problème. Voici la manière dont le problème que nous traitons est donné à SYRCLAD :

```
problème(m10).              /* soit le problème m10 */
action(m10,f).              /* l'action de m10 est f */
à_dénombrer(m10,d).   /* résoudre m10 consiste à dénombrer l'ensemble d */
est_un(f,action_former).    /* f est l'action de former */
au_moyen_de(f,a).           /* au moyen d'un ensemble a */
tout_résultat(f,r).         /* un objet r */
est_un(r,mot).              /* cet objet est un mot */
taille(r,5).                /* de taille 5 */
effectif_attribut(r,exactement,2,type_lettre,voyelle).
effectif_attribut(r,exactement,2,lettre,b).
              /* qui contient exactement 2 b et 2 voyelles */
est_un(a,ensemble).         /* a est un ensemble */
taille(a,26).               /* de taille 26 */
tout_élément(a,xa).         /* tout élément de a */
est_un(xa,lettre).          /* est une lettre */
est_un(d,ensemble).         /* d est un ensemble */
ens_des_résultats(d,f).     /* c'est l'ensemble des résultats de f */
```

Il s'agit en effet de dénombrer l'ensemble des résultats d'une action f où f est l'action de former des mots de taille 5 au moyen des lettres de l'alphabet. Les résultats de cette action sont soumis à une contrainte : ces mots doivent contenir exactement deux voyelles et deux occurrences de la lettre "b".

Ce modèle du problème est un modèle complet et non ambigu contrairement à l'énoncé de l'exercice. Celui-ci est en effet incomplet, puisqu'il sous-entend que l'on utilise toutes les lettres de l'alphabet. Il faut donc ajouter cette information dans le modèle descriptif.

On remarque que l'on y trouve des prédicats décrivant :

- des objets : `est_un(f,action_former), est_un(a,ensemble), est_un(r,mot),`
- des relations entre ces objets : `au_moyen_de(f,a), tout_résultat(f,r),`
- des propriétés sur les objets : `taille(r,5),`
  `effectif_attribut(r,exactement,2,type_lettre,voyelle),`
- une question : `à_dénombrer(m10,d).`

L'ensemble où l'on choisit est ici l'ensemble désigné par a. L'objet formé est l'objet désigné par r. Il y a deux contraintes sur cet objet.

Le fait que le modèle descriptif soit exprimé dans un langage de la logique du premier ordre est indépendant du domaine. C'est une contrainte qu'impose le noyau de SYRCLAD. Par contre, les prédicats utilisés (au_moyen_de, à_dénombrer) sont spécifiques au domaine des dénombrements. Seul le prédicat problème(nom_du_problème) sera présent dans tous les problèmes posés, quel que soit le domaine ; il est imposé par le noyau de SYRCLAD. Certains prédicats comme taille(ensemble,

entier) ou est_un(objet, définition) se retrouveront dans d'autres domaines. Quant aux objets f, r, a, ils sont spécifiques à l'exercice.

Nous avons choisi de ne pas présenter à SYRCLAD un énoncé en langage naturel pour deux raisons. D'une part parce que l'analyse du langage naturel pose de grandes difficultés à un système artificiel. D'autre part parce qu'il n'est pas indispensable d'expliciter les connaissances utilisées pour passer de l'énoncé en langage naturel au modèle descriptif. En effet, le but de SYRCLAD est d'expliciter sa démarche de résolution et les connaissances qu'il utilise, mais les modèles descriptifs sont la traduction de l'énoncé en langage naturel dans un langage de la logique du premier ordre. Les connaissances utilisées lors de cette traduction ne sont pas des connaissances spécifiques du domaine, et nous pensons qu'il ne serait pas intéressant de les présenter à un élève. En faisant ce choix, nous effectuons une première étape de modélisation à la place du système, mais c'est l'étape suivante de modélisation qui est importante, celle qui est difficile pour les élèves et pour laquelle SYRCLAD utilise des connaissances explicites.

2 Le graphe de classification

SYRCLAD utilise un graphe de classification des problèmes du domaine pour classer le problème. Ce graphe est une hiérarchie de classes au sens de la relation de spécialisation, il contient des *attributs discriminants* de la classification et les *valeurs discriminantes* pour ces attributs qui permettent de répartir les instances d'une classe suivant les différentes sous-classes. Cette structure de graphe est celle choisie pour l'architecture de SYRCLAD, et un graphe doit être présenté ainsi pour être utilisé par le noyau qui permet de classer un problème.

L'expert en dénombrement a donné à SYRCLAD un graphe de classification des problèmes de dénombrements. L'objectif de ce graphe est d'expliciter les connaissances nécessaires à l'application d'une méthode pour résoudre des problèmes variés. Cette méthode doit permettre de présenter les étapes de la démarche de résolution. On présente de manière succincte en figure 1 la partie du graphe des problèmes de dénombrement qui nous intéresse pour l'exercice traité. Pour chaque classe on précise l'attribut du problème qui discrimine les sous-classes suivant sa valeur, que nous avons appelé attribut discriminant. La classification complète des problèmes de dénombrement sera détaillée en partie II et donnée en annexe.

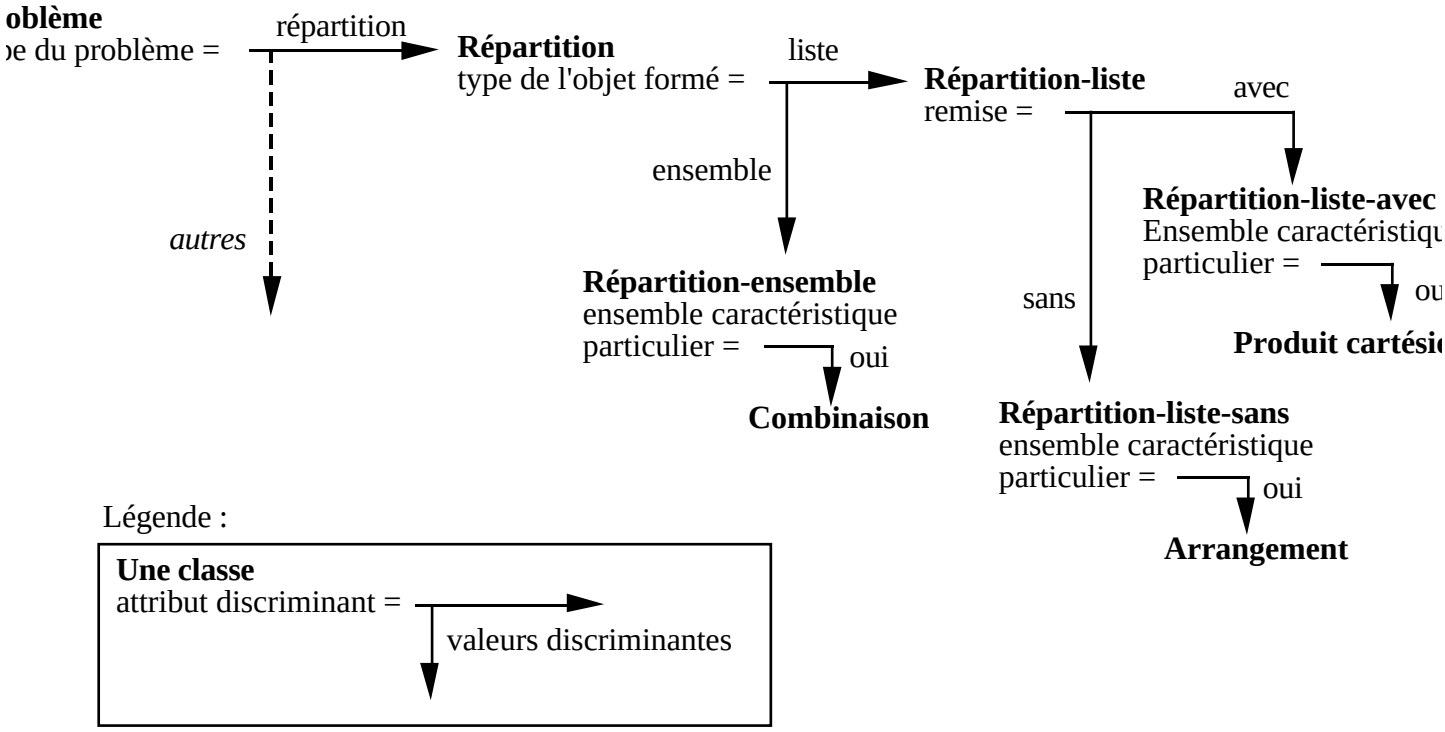


Figure 1 : Une partie du graphe de classification des problèmes de dénombrement

3 Les connaissances de reformulation

Pour utiliser le graphe de classification, il faut posséder des connaissances permettant de déterminer les valeurs des attributs discriminants. SYRCLAD déterminera les valeurs d'attributs discriminants du problème, ainsi que les valeurs d'autres attributs du problème, qui ne sont pas discriminants pour le classement mais qui sont néanmoins utiles. A chaque attribut du problème est attaché un ensemble de règles si-alors qui permettent de conclure sur la valeur de cet attribut. C'est l'expert qui donne au système ces connaissances de reformulation du problème.

Dans une classe donnée, l'expert définit un attribut du problème si cet attribut a un sens pour les problèmes de cette classe. Il précise les règles qui permettent de trouver la valeur de cet attribut. Dans le graphe de classification des problèmes de dénombrement, l'un des attributs discriminants est le *type de l'objet formé*, selon que l'on le considère comme un ensemble ou une liste d'éléments. Cet attribut est défini pour tous les problèmes, il est donc défini dans la classe "Problème" :

```
attribut(c_problème, type_objet_formé(Problème, Type), [liste de noms de règles
concluant sur le type de l'objet formé, dont R1]).
```

Le prédicat "attribut" fait partie de l'architecture de SYRCLAD, il sera utilisé quel que soit le domaine.

Pour l'exercice que nous traitons, une des règles associées à cet attribut s'applique :

> R1 : Si l'objet formé O d'un problème P est un mot, alors le type de l'objet formé pour le problème P est liste.

L'objet formé est un attribut du problème P, dont la valeur O (ici le résultat de l'action-former) est aussi déterminée par une des règles qui lui sont associées.

4 Le classement

Nous allons à présent suivre la démarche de SYRCLAD pendant le classement de l'exercice que nous traitons ici. SYRCLAD se base sur le graphe de classification dont une partie est donnée en figure 1. Il utilise des connaissances de reformulation[1] pour construire un nouveau modèle du problème à partir du modèle descriptif. Le classement et la construction de ce nouveau modèle du problème ont lieu simultanément.

Le processus de classement est indépendant du domaine. Il forme le noyau de SYRCLAD et sera détaillé dans le chapitre 3. Ce processus prend en entrée un modèle descriptif D d'un problème P. En sortie, on obtient une classe C telle qu'il existe une instance M de C représentant un nouveau modèle de P. Pour cela, SYRCLAD descend dans le graphe de classification, de manière à spécialiser petit à petit la représentation qu'il se fait du problème. Les attributs discriminants du classement seront déterminés au fur et à mesure grâce aux règles de reformulation. SYRCLAD construira ainsi M, nouveau modèle du problème, mieux adapté à la résolution, et que nous avons appelé *modèle opérationnel* du problème.

Pour commencer à classer l'exercice, le premier attribut discriminant est le **type du problème**, c'est-à-dire le type de contraintes sur l'objet formé. Pour utiliser les règles permettant de déterminer le type du problème, on a donc besoin de déterminer quel est l'*objet formé* et quelles sont les *contraintes* sur cet objet. Une règle permet de déterminer qu'ici l'objet formé est le résultat r de l'action-former f, action du problème. Pour établir la liste des contraintes, on construit un élément-générique y de l'ensemble à dénombrer d ; on rapporte les (deux) contraintes (effectif-attribut) à cet élément-générique ; puis on dresse la liste de ces contraintes :

```
[effectif_attribut(y,exactement,2,type_lettre,voyelle),
effectif_attribut(y,exactement,2,lettre,b)]
/* y doit contenir exactement 2 voyelles et 2 b */
```

Pour déterminer le type du problème, on a également besoin de connaître les attributs[2] intervenant dans les contraintes. Une règle permet de reconnaître que l'ensemble où l'on choisit est l'alphabet. Les attributs "type-lettre" et "lettre" sont donc des attributs possibles. Ils définissent des catégories correspondantes, celle des voyelles de taille 6 et celle du b de taille 1.

Une règle conclut alors sur le type du tirage : nous sommes dans le cas où il y a plusieurs contraintes de même type (effectif-attribut), mais dont les attributs sont différents (type-lettre et lettre),

---

[1] Toutes les règles déclenchées par SYRCLAD au cours du classement de cet exercice sont données en annexe A.

[2] Attention : "attribut" est ici un concept du domaine des dénombrements (propriété d'une lettre de l'alphabet) et non pas un concept du noyau de SYRCLAD (attribut d'un problème).

définissant des catégories disjointes (le b n'appartient pas aux voyelles). Le type du problème est donc connu. Il s'agit de ce que nous avons appelé une *répartition* entre ces catégories. La règle calcule en même temps l'ensemble caractéristique correspondant à cette répartition. Il s'agit de préciser pour chaque catégorie définie par les contraintes : la taille de la catégorie, le nombre d'éléments à choisir dans cette catégorie et sa description. Ici, l'ensemble caractéristique est : ((1, 2, b) , (6, 2, voyelle)).

Après avoir déclenché 21 règles, SYRCLAD a déterminé la valeur du type du problème. Il en déduit que le problème appartient à la sous-classe *répartition* de la classe problème (cf. figure 1). Savoir que le problème appartient à la classe *répartition* n'est pas suffisant pour résoudre ce problème. Il faut continuer à descendre dans le graphe afin d'affiner le classement. Pour cela, il faut déterminer la valeur de l'attribut discriminant, qui est le **type de l'objet formé**. SYRCLAD utilise une des règles attachées à cet attribut. La règle pertinente pour ce problème indique qu'un mot doit être considéré comme une liste et non comme un ensemble de lettres.

Nous sommes donc à présent dans la classe *répartition-liste*. L'attribut discriminant de cette classe est la **remise**. Un choix d'éléments a lieu avec remise si l'on peut choisir plusieurs fois le même élément, sans remise dans le cas contraire. Aucune indication n'est donnée à ce sujet dans l'énoncé de l'exercice. SYRCLAD ne trouve pas de règle déclenchable, il choisit alors une règle s'appliquant par défaut qui indique que faute de précisions, on considère que l'on peut répéter plusieurs fois la même lettre dans un mot. Le choix des éléments dans le problème a donc lieu avec remise. Nous descendons donc dans la sous-classe *répartition-liste-avec*.

SYRCLAD essaie d'affiner encore le classement, le problème est-il un produit cartésien ? Pour cela il essaie de déclencher une des règles permettant de conclure que l'ensemble caractéristique est un cas particulier. Or comme ce n'est pas le cas, aucune règle ne se déclenche, et le classement est terminé.

Le classement se termine dans la classe *répartition-liste-avec*. Cette classe est assez spécifique pour que l'on puisse envisager une méthode de résolution, c'est une classe *opérationnelle*. Le classement se termine donc de manière satisfaisante. En même temps que le classement, SYRCLAD a déterminé la valeur de plusieurs attributs, construisant ainsi un nouveau modèle du problème, que nous appelons *modèle opérationnel*.

5 Le modèle opérationnel

Un des résultats du classement est un nouveau modèle du problème, tourné vers la résolution, et dont les attributs correspondent aux critères de la classification. Ce modèle opérationnel contient aussi des attributs non discriminants, qui permettent de compléter le modèle opérationnel en vue d'une résolution.

Pour l'exemple traité, des attributs discriminants ont été calculés pendant le classement :

Type du problème : *répartition*
Type de l'objet formé : *liste*
Remise : *avec*

Ces trois attributs, avec leurs valeurs, forment les critères (attribut = valeur) qui définissent la classe du problème.

Le modèle opérationnel est complété par d'autres attributs, non discriminants, qui seront utiles pour la résolution. Les valeurs de ces attributs sont propres à l'exercice, elles ont été déterminées lors du calcul des attributs discriminants.

```
Taille de l'ensemble où l'on choisit : 26
Taille de l'objet formé : 5
Ensemble caractéristique[1] : ((1, 2, b),(6, 2, voyelle))
```

En utilisant le modèle opérationnel, on pourrait reformuler le problème :

> « Soit un ensemble E de taille 26 divisé en 2 catégories disjointes d'effectifs 1 et 6. On tire 5 éléments dans E successivement, avec remise, et on forme une liste. Combien y a-t-il de résultats contenant exactement 2 éléments de la première catégorie et 2 éléments de la seconde catégorie ? »

Pour produire cette formulation, nous avons utilisé les trois attributs discriminants afin d'avoir une formulation-type (cf. § 6). Nous avons adapté cette formulation-type au problème traité grâce aux trois autres attributs.

6 Choisir une méthode de résolution

Pour choisir une méthode de résolution à employer, il faut savoir quelle méthode est adaptée pour un problème. Dans chaque domaine, la classification utilisée a été construite par l'expert en vue de la résolution. Certaines classes sont assez spécifiques pour que l'on sache associer à la classe une méthode de résolution adaptée aux problèmes de cette classe. Nous les appelons *classes opérationnelles*. L'expert attache donc à chaque classe opérationnelle du graphe de classification une méthode de résolution. Ces méthodes de résolution ont des formes différentes suivant les domaines.

On écrira :

```
employer_résolution(classe_opérationelle):- méthode de résolution
                                              (utiliser une règle par exemple)
```

En dénombrement, on associe à chaque classe opérationnelle ce que l'on appelle un *plan de résolution type* pour les exercices de cette classe, ainsi qu'une formule permettant de calculer la solution numérique.

Un plan de résolution est une description de type algorithmique des actions à effectuer pour former un objet de l'ensemble à dénombrer. L'analyse de ce plan permet de calculer la solution numérique. S'il n'a pas le statut d'une démonstration de la solution, il est assez détaillé pour associer à chaque

[1] taille de la catégorie, nombre d'éléments à choisir dans la catégorie, description de la catégorie

action le nombre de possibilités que l'on a pour réaliser cette action, et permet de calculer en multipliant ces nombres la solution numérique de l'exercice. C'est ce type de plan que l'on demande aux élèves de fournir dans une copie pour justifier le résultat numérique.

Pour la classe *répartition-liste-avec*, on peut écrire un problème type :

> « Soit un ensemble E de taille e divisé en m catégories disjointes d'effectifs c1, …, cm. On tire p éléments dans E successivement, avec remise, et on forme une liste. Combien y a-t-il de résultats contenant exactement x_1 éléments de catégorie 1, …, x_n éléments de catégorie n ? ».

Comme le type de l'objet formé est "liste", le plan de résolution type associé à cette classe introduit l'ensemble des places dans le mot :

> « On choisit x_1 places parmi p pour les <description catégorie 1>, puis on choisit un <description catégorie 1> parmi c_1 pour chacune de ces places, *
> puis on choisit x_2 places parmi p - x_1 pour les <description catégorie 2>, puis on choisit un <description catégorie 2> parmi c_2 pour chacune de ces places, *
> etc.,
> puis on complète les p - $(x_1+x_2+\ldots+x_n)$ places restantes en choisissant p - $(x_1+x_2+\ldots+x_n)$ éléments * parmi les e - $(c_1+c_2+\ldots+c_n)$ éléments restants. »

La solution numérique est obtenue par la formule suivante :

$$\frac{p!{c_1}^{x_1}\ldots{c_n}^{x_n}(e-(c_1+\ldots+c_n))^{p-(x_1+\ldots+x_n)}}{x_1!x_2!\ldots x_n!(p-(x_1+\ldots+x_n))!}$$

On peut remarquer que pour la classe *répartition-liste-sans_remise*, le plan de résolution sera presque le même, excepté l'ajout de ", tous différents, " à la place des *. Dans la formule numérique, les puissances seront remplacées par des arrangements. En effet, le plan-type et la formule associés à une classe opérationnelle sont liés à la classe, donc liés aux valeurs des attributs discriminants qui la définissent.

Les plans de résolution type et les formules associées aux autres classes du graphe de classification des problèmes de dénombrement sont donnés en annexe.

7 Appliquer la méthode de résolution et donner la solution

Pour donner une solution au problème, SYRCLAD applique sur le modèle opérationnel la méthode de résolution associée à la classe du problème. On n'a plus besoin pour appliquer cette méthode du modèle descriptif donné initialement à SYRCLAD, les informations utiles sont contenues dans le modèle opérationnel.

En dénombrement, SYRCLAD adapte le plan de résolution type à l'exercice, et il calcule la solution numérique grâce à la formule. En utilisant le plan et la formule associés à la classe *répartition-liste-avec* (cf. § 6), SYRCLAD produit une solution à l'exercice posé dans ce chapitre :

```
Plan de résolution : On choisit 2 places parmi 5 pour les b,
```

```
puis on choisit un b parmi 1 pour chacune de ces places
puis on choisit 2 places parmi 3 pour les voyelle, puis on choisit un
voyelle parmi 6 pour chacune de ces places
puis on complète les 1 places restantes en choisissant 1 éléments parmi les
19 éléments restants.
Solution : 20520        /*    C52*1*1*C32*6*6*19    */
```

Pour appliquer le plan-type et la formule, SYRCLAD a utilisé les attributs du modèle opérationnel qui ne sont pas discriminants : la taille de l'ensemble où l'on choisit (26), la taille de l'objet formé (5), et l'ensemble caractéristique ((1, 2, b) , (6, 2, voyelle)).

Ce plan de résolution est peu différent de celui présenté en début de chapitre :

> « Je vais d'abord placer une voyelle, j'ai 6 voyelles possibles, et j'ai 5 façons de la placer. Je dois ensuite placer une deuxième voyelle, j'ai toujours 6 possibilités pour la choisir, mais je n'ai plus que 4 places disponibles. A présent, je dois placer un b, il y a 3 places possibles, puis un deuxième b en choisissant sa place parmi deux. J'ai enfin 19 lettres possibles pour remplir la dernière place disponible. Il y a donc 6*5*6*4*3*2*19 = 82080 possibilités. »

En effet, la seule différence provient du fait que ce plan considère les places l'une après l'autre alors que celui de SYRCLAD choisit en une seule fois les places pour une catégorie.

Dans les deux cas la modélisation du problème est correcte. Pourtant, nous sommes sûrs que c'est le plan de SYRCLAD qui est juste, car l'expertise qui associe à une classe un plan-type et une formule peut être validée par une démonstration mathématique [Le Calvez et al 97].

8 Conclusion

A travers l'exemple que nous avons présenté, nous pouvons discerner trois niveaux de connaissance ou d'activité dans SYRCLAD.

• Tout d'abord, il y a le noyau de SYRCLAD, indépendant du domaine. Ce noyau est conçu et implémenté par le concepteur du système. Il fournit un cadre conceptuel dans lequel on doit représenter les connaissances et un mécanisme d'utilisation de ces connaissances.

• Ensuite, l'expert d'un domaine X fournit, dans le cadre conceptuel exigé par le noyau, les connaissances de classification, de reformulation et de résolution spécifiques au domaine. Le noyau et ces connaissances forment un résolveur de problèmes du domaine X, qu'on peut appeler SYRCLAD-X.

• Enfin, un utilisateur soumet à SYRCLAD-X un problème à résoudre, sous la forme d'un modèle descriptif de ce problème exprimé dans un langage de la logique du premier ordre. Dans ce chapitre, nous avons étudié la résolution d'un exercice de dénombrement par SYRCLAD-dénombrements.

Pour classer ce problème, SYRCLAD-dénombrements utilise un graphe de classification de problèmes de dénombrement que l'expert lui a donné. Pour cela, il a besoin de règles permettant de déterminer la valeur des attributs discriminants intervenant dans la classification. Ces règles sont les connaissances de reformulation, également fournies par l'expert.

Le noyau de SYRCLAD effectue une *opérationnalisation* du problème, obtenant ainsi d'une part une classe (si possible opérationnelle) et d'autre part un nouveau modèle du problème, appelé modèle opérationnel. Une expertise fournie par l'expert associe à chaque classe du graphe de classification une méthode de résolution adaptée. Pour obtenir une solution du problème, SYRCLAD applique cette méthode de résolution au modèle opérationnel.

Nous commençons à distinguer l'architecture de SYRCLAD (cf. figure 2) :

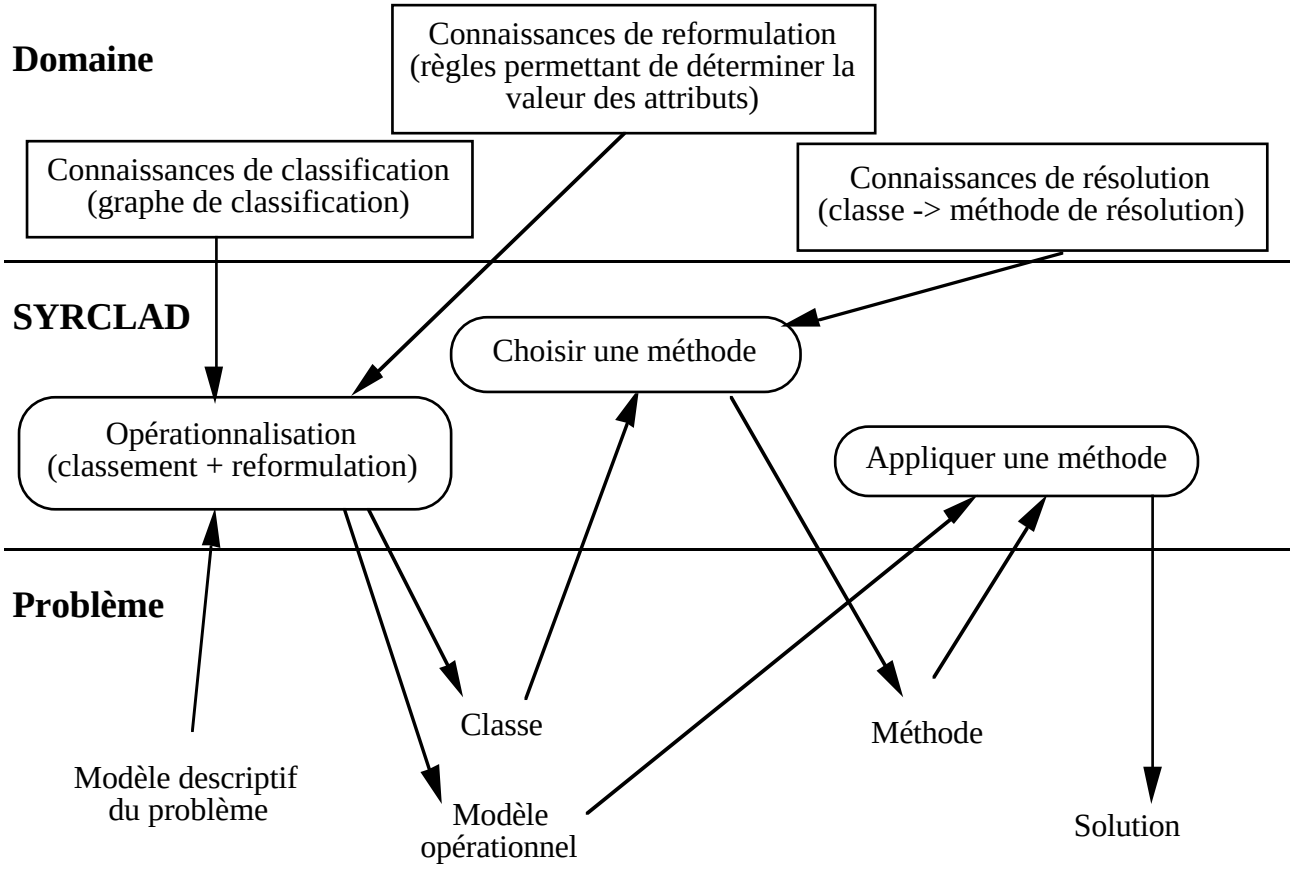


Figure 2 : l'architecture de SYRCLAD

Le niveau "domaine" contient des *bases de connaissances*, spécifiques au domaine, que nous avons représentées par des rectangles. Le niveau "SYRCLAD" contient des *processus* qui utilisent les bases de connaissances du domaine ; nous avons représenté les processus en arrondi.

Dans ce chapitre, nous avons essayé de donner au lecteur une première idée de ce qu'est le système SYRCLAD. Nous étudierons plus en détail son architecture et le fonctionnement de son noyau au chapitre 3. Nous présenterons d'abord dans le chapitre 2 les travaux qui nous ont conduit à réaliser ce système, et nous comparerons SYRCLAD à d'autres systèmes d'intelligence artificielle utilisant des classifications.

Chapitre 2

Classification et résolution de problèmes : différentes approches

Dans un premier temps, nous évoquerons dans ce chapitre des travaux de psychologie cognitive et de didactique qui ont motivé et justifié l'approche de SYRCLAD. Nous essaierons de montrer qu'utiliser une classification permet de modéliser et de résoudre efficacement des problèmes. Nous soulignerons l'intérêt d'une telle approche dans le cadre de l'EIAO.

Dans un deuxième temps, nous comparerons SYRCLAD à des travaux d'intelligence artificielle utilisant des classifications. Nous essaierons de souligner l'originalité de SYRCLAD et les capacités spécifiques de ce système.

Plan du chapitre

1 Travaux de psychologie cognitive

Dans l'enseignement, l'apprentissage passe souvent par la présentation aux élèves de problèmes corrigés. De nombreuses expérimentations de psychologie cognitive consistent à proposer aux élèves des problèmes corrigés qu'ils doivent analyser et expliquer à haute voix, leurs verbalisations faisant l'objet d'une étude. Les travaux sur l'analogie issus de [Gick et Holyoak 80, 83] montrent qu'il faut présenter plusieurs exemples pour que l'élève puisse construire une structure abstraite de connaissances, appelé *schéma*. A. Renkl établit dans [Renkl 97] que les élèves qui profitent le plus de l'analyse des exemples sont ceux qui explicitent les principes du domaine utilisés et les sous-buts à atteindre, et ceux qui anticipent les étapes du raisonnement. A. Didierjean et E. Cauzinille-Marmèche montrent dans [Didierjean et Cauzinille-Marmèche 97] l'existence de différences individuelles parmi les sujets analysant des corrigés-types.

Certains élèves cherchent, pendant l'analyse de ces corrigés, à anticiper les étapes de la résolution et créent ainsi des *attentes déçues*[1] lorsque le corrigé ne correspond pas à leurs prévisions. Ils détectent ainsi comment résoudre certaines étapes spécifiques et construisent des *cas,* en mémorisant la résolution d'un problème spécifique. Ces élèves résolvent bien les problèmes proches des exemples, mais n'arrivent pas à résoudre des problèmes dont l'aspect contextuel est éloigné. D'autres élèves ont des verbalisations qui font apparaître qu'ils comparent les exemples entre eux. Ces élèves construisent des schémas et résolvent bien les problèmes éloignés des exemples. A. Didierjean et E. Cauzinille-Marmèche observent également que certains des élèves qu'ils étudient utilisent à la fois des schémas et des cas. Ces élèves obtiennent les meilleurs résultats au moment de résoudre, mais la charge mentale requise semble plus importante. Cette charge mentale est mesurée par une deuxième tâche (mémoriser une liste de mots) que les élèves doivent accomplir simultanément. Enfin, il apparaît qu'après quelque temps, les élèves ayant précédemment utilisé des cas ont construit des schémas. Ce résultat se retrouve dans [Ross et Kennedy 90], où les performances de certains sujets ayant préalablement analysé des exemples sont sensibles à l'aspect contextuel des problèmes à résoudre lors d'un premier post-test, et ne le sont plus lors d'un deuxième post-test. Ceci permet de supposer que raisonner par cas est une première étape, la généralisation des cas amenant par la suite à construire une classification.

M. Chi a montré dans [Chi et al 81] que les experts se distinguent des novices par la classification qu'ils donnent d'un problème. Lorsque l'on demande de classer des problèmes de mécanique, les novices parlent de "plan incliné" ou de "ressort", c'est-à-dire des traits de surface des problèmes. Les experts parlent de "conservation de l'énergie" ou de "troisième loi de Newton". Ils ont directement une idée de la méthode de résolution qu'il faudra appliquer. Pour M. Chi, la classification est l'activation d'un schéma qui contient des connaissances. Cela débouche d'une part sur une représentation du problème (il s'agit de l'aspect déclaratif des connaissances) et d'autre part sur une

[1] À propos des attentes déçues, on pourra consulter [Hammond 90].

solution (il s'agit de l'aspect procédural des connaissances). Il se peut qu'un sujet, lorsqu'il a assimilé et compilé une classification de problèmes, possède des schémas.

Dans SYRCLAD, nous explicitons les classifications de problèmes dans un but pédagogique. Les classes opérationnelles peuvent être considérées comme des schémas. On retrouve bien dans SYRCLAD l'aspect déclaratif des connaissances d'un schéma, car il utilise des connaissances de reformulation qui permettent (avec le graphe de classification) de trouver une représentation du problème. L'aspect procédural des connaissances d'un schéma est représenté dans SYRCLAD par les connaissances de résolution, qui associent une méthode de résolution à une classe de problème.

Dans [Chi 87], M. Chi distingue dans le raisonnement humain les connaissances procédurales que l'on représente par des règles de production, des connaissances déclaratives que l'on représente par des graphes. Elle ajoute que cette distinction est également valable pour les métaconnaissances. Les connaissances de classification sont des connaissances de niveau méta puisqu'elles permettent de raisonner sur l'énoncé du problème [Pitrat 90]. Ce sont des connaissances déclaratives, que l'on représente dans SYRCLAD à l'aide d'un graphe.

Dans [Robert et Robinet 96], les auteurs estiment que les métaconnaissances peuvent servir à rendre opérationnelles les connaissances enseignées. Nous pensons que l'association dans SYRCLAD d'un graphe de classification et de méthodes de résolution est une représentation opérationnelle de connaissances de résolution. Les auteurs pensent qu'il n'y a pas lieu de comparer les conceptions et les pratiques des experts et des élèves. Nous souhaitons avec SYRCLAD expliciter non les connaissances de l'expert, mais celles de l'élève-expert au sens de J.R. Anderson : un élève "idéal" [Anderson et al 87]. Dans SYRCLAD, les connaissances explicitées sont les connaissances du niveau d'un élève qui maîtrise le domaine en appliquant une méthode de niveau méta.

Dans [Cauzinille-Marmèche et Mathieu 88], les auteurs proposent d'envisager pour un EIAO une base de connaissances structurée en différents niveaux :

> « Il s'agirait donc de représenter explicitement les concepts du domaine et leurs relations (qui peuvent être de nature différente), les catégories de problèmes et les plans d'actions associés, les procédures élémentaires qui interviennent dans la résolution de problèmes, ainsi que les relations entre ces différents plans de connaissances. »

L'architecture de SYRCLAD permet ainsi de distinguer ces types de connaissances : les concepts du domaine et leurs relations sont représentés dans le graphe de classification. Les classes opérationnelles représentent les catégories de problèmes, et des "plans d'actions" leur sont associés. Toutes ces connaissances sont liées au domaine d'application, elles appartiennent donc au niveau "domaine" de l'architecture de SYRCLAD.

2 Travaux de didactique et EIAO[1]

Nous indiquons ici en quoi l'approche de SYRCLAD est appropriée aux EIAO, tout d'abord du point de vue de la modélisation de problèmes concrets, puis vis-à-vis de l'enseignement de méthodes, et enfin en vue de donner des explications.

2.1 Résoudre un problème concret

Nous nous intéressons à des domaines où les énoncés des problèmes correspondent à des situations concrètes alors que le cours est donné sous forme de connaissances théoriques. Les élèves sont en difficulté face à ce type de problèmes parce qu'ils n'arrivent pas à modéliser ces situations afin qu'elles entrent dans le cadre du cours théorique [Antibi 88]. Dans Modélis [Tisseau 90], G. Tisseau a explicité les connaissances nécessaires à la modélisation de problèmes de thermodynamique, cette modélisation étant une étape préalable à l'utilisation des connaissances de résolution d'un cours théorique.

Dans SYRCLAD, nous avons choisi d'utiliser une classification pour construire un nouveau modèle du problème considéré, qui correspond mieux aux connaissances théoriques, et auquel il est possible d'appliquer les méthodes de résolution du cours.

2.2 Enseigner une méthode

Dans le domaine des EIAO et de la didactique, des études d'enseignement de méthodes ont été menées par A. Robert et al. [Robert et al 87] et M. Rogalski [Rogalski 90]. M. Rogalski propose :

> « Pourquoi ne pas enseigner explicitement les méthodes qui, dans chaque domaine relativement limité, permettent d'amorcer la résolution de problèmes de ce domaine ? Ne serait-ce pas un moyen de combler un manque évident de l'enseignement, de le rendre plus efficace, de mieux dominer les concepts mathématiques en montrant explicitement comment ils servent ? » [Rogalski 90].

Mais qu'est-ce qu'une méthode ?

> « Une méthode ou un ensemble de méthodes sur un champ donné est la description d'un ensemble d'activités du sujet, portant sur l'analyse et le classement de problèmes à résoudre dans un domaine assez précis, l'utilisation des outils et des techniques disponibles, les stratégies et tactiques possibles, la gestion dans le temps des choix, des stratégies et de leur déroulement, la conscience de ces choix, les moyens de contrôle et de retour arrière pour procéder à d'autres choix... Un algorithme produit une réponse, une méthode fournit des questions : quoi, pourquoi, comment, par quels moyens... et donne des outils pour générer et contrôler la recherche des réponses » [Rogalski 90].

---

[1] EIAO signifie depuis 1992 et sur l'initiative de M. Baron : Environnement Interactif d'Apprentissage avec Ordinateur

Une méthode permet d'organiser une résolution de problèmes en se basant sur un classement des problèmes et des outils de résolution. Elle permet ainsi d'expliciter les métaconnaissances associées à certaines connaissances du domaine et à certaines classes de problèmes. En accord avec G. Paquette [Paquette 91], M. Rogalski énonce pluiseurs thèses [Rogalski 94] :

> Thèse n° 1 :
> « L'enseignement de méthodes de résolution de problèmes dans un domaine mathématique bien défini est une utilisation privilégiée d'un tuteur intelligent (dans le cadre de l'enseignement des mathématiques). »
>
> Thèse n° 2
> « Le résolveur d'un tuteur *donneur de leçons de méthodes* doit fonctionner selon les méthodes qu'il veut enseigner, et non pas selon les méthodes expertes les plus générales du domaine concerné. »

Il poursuit ainsi :

> « Cette thèse n°2 peut servir de point de départ pour commencer à répondre à une question de Martial Vivet : "quand on envisage un tuteur intelligent, faut-il partir de l'apprenant ou des connaissances ?" La réponse est peut-être qu'**il faut partir des connaissances telles qu'on veut qu'elles fonctionnent réellement après l'apprentissage**. Cet état qu'on veut obtenir chez l'apprenant n'est pas celui qui correspondrait aux connaissances expertes les plus générales sur le domaine étudié » .

Selon M. Rogalski, certaines méthodes proposées sont "algorithmisables" comme la méthode de recherche de primitives selon la forme des intégrandes [Rogalski 88] qui est implémentée dans l'environnement ELISE [Delozanne 92]. "Algorithmisable" signifie ici que l'importance de la classification y est déterminante, que les techniques sont en nombre limité et que les consignes correspondent à des actions techniques modifiant la forme du problème afin de se rapprocher de la solution.

D'autres méthodes sont plus exploratoires, comme celles qui concernent la convergence des suites numériques [Rogalski 94], l'enseignement de l'algèbre linéaire [Rogalski 91], et la résolution de problèmes de géométrie en terminale scientifique [Robert et Tenaud 89].

La réalisation d'un résolveur utilisant une méthode exploratoire semble difficile. Mais pour les méthodes plus algorithmisables, SYRCLAD semble être bien adapté, ces méthodes étant basées sur la classification des problèmes et des outils de résolution associés.

2.3 Donner des explications

Un de nos objectifs est de représenter des classifications de problèmes dans des domaines variés, en utilisant un mode de représentation général. Nous souhaitons ainsi, dans chaque domaine et dans un cadre d'enseignement à un niveau donné, pouvoir expliquer à un élève la démarche de résolution d'un élève-expert, et les métaconnaissances qu'il utilise. Les explications concernant la résolution de ce problème seront d'autant plus faciles à donner que les connaissances seront explicitées. C'est pourquoi nous avons distingué dans SYRCLAD les différents types de connaissances intervenant dans une résolution : les connaissances de classification, les connaissances de reformulation et les

connaissances de résolution ; ces différents types de connaissances seront exprimées de manière déclarative.

Représenter les connaissances dans un système pour résoudre n'implique pas nécessairement que ce système puisse construire des explications. En effet, W.J. Clancey a conçu le tuteur GUIDON [Clancey 87] à partir du système expert MYCIN [Shortliffe 76], les résolutions de MYCIN servant de référence à GUIDON pour construire des explications. Les étudiants ont jugé que les capacités d'explication de GUIDON étaient inadaptées et insuffisantes. En effet, dans MYCIN, les connaissances relatives à la stratégie de raisonnement étaient implicites. Après une étude approfondie sur les exigences d'une modélisation des connaissances du domaine en EIAO [Clancey 83], W.J. Clancey a conçu NEOMYCIN [Clancey et Letsinger 84], nouveau système dans lequel les connaissances et les stratégies de raisonnement sont explicites. M. Baron tire les enseignements de GUIDON dans [Baron 84, 93]. Elle souligne, en particulier, le lien entre d'une part la modélisation du domaine et de la stratégie de raisonnement et d'autre part les capacités d'explication d'un module de résolution de problèmes pour un EIAO. Elle ajoute qu'il est nécessaire d'expliciter des métaconnaissances pour utiliser les connaissances du domaine. En réalisant SYRCLAD, nous avons souhaité construire des résolveurs de problèmes dans lesquels les métaconnaissances pour utiliser les connaissances soient explicites, afin que ces résolveurs puissent servir de référence pour construire des explications dans un EIAO[1].

M. Vivet distingue trois niveaux d'explications [Vivet 87, 88]. Le niveau le plus bas consiste à *commenter* une trace de résolution d'un système expert en utilisant les aspects syntaxiques des règles utilisées. *Expliquer* consiste à analyser le contenu sémantique de ces règles de manière à expliciter les connaissances utilisées par le système pour produire la résolution. Enfin, *justifier* consiste à détailler les connaissances stratégiques utilisées par le système pour élaborer la résolution du problème. Un système résolveur de problèmes doit donc, s'il a pour but de donner des explications, posséder des métaconnaissances explicites pour exprimer les choix stratégiques de résolution. Dans SYRCLAD, les choix stratégiques de résolution sont déterminés par l'utilisation du graphe de classification, où les (méta)connaissances stratégiques sont explicites et déclaratives.

J.-F. Nicaud exprime dans [Nicaud 94] la nécessité de présenter en EIAO des modèles à un *niveau connaissance* [Newel 82]. Il s'agit d'un « niveau de modélisation dans lequel les connaissances sont détaillées, sans pour autant être exprimées dans des langages informatiques ». L'une des caractéristiques des modèles à niveau connaissance est d'être communicables entre les êtres humains, ce qui est particulièrement important dans la communauté pluridisciplinaire des EIAO. J.-F. Nicaud insiste également sur le fait que le module résolveur d'un EIAO doit être adapté à un niveau d'apprentissage donné et ainsi servir de *résolveur de référence* ayant une plus forte plausibilité cognitive qu'un système expert. « Le comportement de ce résolveur doit pouvoir être fourni comme exemple à un apprenant du domaine ». La compétence d'un résolveur de référence est donc supposée adaptée à un niveau donné de l'apprentissage.

---

[1] Mais il n'existe actuellement aucun EIAO qui utilise SYRCLAD pour produire des explications.

SYRCLAD s'inscrit dans cette démarche : chaque résolveur SYRCLAD-X a pour objectif d'être le résolveur de référence d'un EIAO sur le domaine X. La déclarativité de la représentation des connaissances dans SYRCLAD permet d'expliciter une démarche de résolution au "niveau connaissance". En effet, le graphe de classification et les règles (si...alors) de reformulation sont exprimées indépendamment d'un langage informatique. Les connaissances de résolution qui associent une méthode de résolution à une classe de problèmes également. Si nous reprenons le problème m10 du chapitre 1, nous avons utilisé (entre autres) les connaissances :

- "un mot est une liste d'éléments" (connaissance de reformulation)
- "si l'on forme une liste, il faut savoir si c'est avec ou sans remise" (connaissance de classification)
- "dans un mot on peut utiliser plusieurs fois la même lettre" (connaissance de reformulation)

Nous avons également explicité au chapitre 1 le plan de résolution type et la formule associés au problème-type de la classe "répartition-liste-avec_remise" (§6) (connaissances de résolution).

Le projet ELMER [Choquet et al 96] vise à construire des systèmes à base de connaissances qui explicitent une démarche de raisonnement fondée sur les éléments cognitifs et pédagogiques du domaine. Ces systèmes permettent d'une part de résoudre des problèmes et d'autre part, ils servent de référence pour aider un élève. Le système doit pour cela expliciter et justifier sa résolution. SYRCLAD s'inscrit dans cette démarche. En effet, l'architecture de SYRCLAD permet d'expliciter une démarche de résolution à travers trois bases de connaissances (connaissances de classification, de reformulation et de résolution) ; le noyau de SYRCLAD utilise les bases de connaissances fournies par l'expert pour résoudre des problèmes suivant cette démarche.

Les systèmes du projet ELMER ont pour objectif, lorsque l'élève résout un problème, d'analyser ses actions et de le guider dans sa résolution. SYRCLAD ne possède pas encore ces capacités, mais a été conçu afin de faciliter leur mise en place. EMMA [Choquet et al 97] est un système réalisé dans le cadre du projet ELMER. Le domaine étudié dans EMMA est celui des mathématiques appliquées, et en particulier la programmation linéaire. Sa réalisation est basée sur le concept tâche-méthode. Selon ce concept, résoudre un problème consiste à décomposer cette tâche en sous-tâches et à choisir la meilleure méthode pour réaliser une tâche. Les problèmes à résoudre sont des problèmes concrets, les premières tâches à réaliser pour résoudre un problème étant "analyser le problème", puis "formaliser la situation mathématique". Nous avons choisi dans SYRCLAD d'expliciter les connaissances nécessaires pour réaliser ces deux tâches dans une base de connaissances de reformulation.

E. Delozanne propose dans ELISE [Delozanne 92] une interaction explicative implémentée par une maquette pour laquelle elle a simulé un résolveur de problèmes utilisant la méthode de M. Rogalski pour le calcul de primitives. E. Delozanne insiste sur le fait que la *tâche d'explication* doit être prise en compte dès la phase de conception d'un résolveur de problèmes. Il faut, en particulier, identifier les connaissances stratégiques de résolution du domaine, dont les métaconnaissances qui permettent d'évoquer les connaissances, de faire des choix raisonnés, et donc de justifier ces choix [Delozanne 94]. Les explications produites par ELISE sont détaillées et adaptées à chaque étape de résolution. L'architecture de SYRCLAD pourrait être utilisée pour réaliser le résolveur de problèmes utilisant la méthode de calcul de primitives qu'enseigne ELISE et qui a pour l'instant été simulé.

Nous avons essayé de montrer que SYRCLAD est une architecture pour réaliser des résolveurs de problèmes bien adaptée aux EIAO. En effet, elle permet d'exprimer des connaissances de classification, de reformulation et de résolution de manière explicite et déclarative. Un SYRCLAD-X est ainsi destiné à servir de résolveur de référence à un EIAO en modélisant la démarche de résolution et les connaissances telles qu'on voudrait qu'elles fonctionnent chez un élève après l'apprentissage. La déclarativité et le découpage en trois bases de connaissances devraient à notre avis faciliter par la suite la production d'explications.

Nous allons à présent comparer SYRCLAD à des résolveurs de problèmes utilisant la classification.

3 Travaux d'Intelligence Artificielle[1]

Comme le souligne J. Pitrat : « il est intéressant de changer la représentation d'un problème afin de se ramener à un problème pour lequel on a des outils efficaces ». Il considère que la classification des problèmes du domaine est une connaissance de niveau méta qui permet de classer un problème et ainsi de choisir une bonne représentation et une bonne méthode de résolution. Classer un problème peut également donner une idée de sa difficulté [Pitrat 90]. Dans SYRCLAD, pour résoudre un problème, la démarche consiste à se ramener à un problème connu, pour lequel on connaît une méthode de résolution efficace. Pour ce faire, nous utilisons la classification pour changer la représentation d'un problème afin de se ramener à une représentation plus opérationnelle.

3.1 Résolveurs de problèmes

3.1.1 GEOMUS

GEOMUS [Bazin 93] est un résolveur d'exercices de géométrie du niveau de la classe de quatrième. Il a été conçu pour "résoudre des exercices de géométrie en utilisant les connaissances de l'élève, tout en mettant en œuvre des stratégies, des heuristiques utilisées par le professeur, mais qui restent compréhensibles par l'élève". C'est également le choix que nous avons fait pour SYRCLAD, en explicitant une méthode de résolution de l'élève-expert.

GEOMUS extrait des sous-figures de la figure correspondant à un exercice et associe une étiquette à chaque sous-figure caractéristique. Il mobilise alors les connaissances attachées à ces étiquettes. Par exemple dans une figure complexe, il extrait tous les triangles dans lesquels on a nommé les milieux de deux côtés, et sélectionne ensuite les connaissances sur la droite des milieux (dans un triangle (MNP), la droite qui joint les milieux de [MN] et [MP] est parallèle à [NP]).

Même si GEOMUS n'utilise pas de classification de problèmes explicite, sa démarche en est proche. En effet, les étiquettes correspondent aux titres de chapitres du cours de géométrie, on peut ainsi considérer qu'elles forment une classification des connaissances du cours. Les connaissances attachées à ces étiquettes peuvent également être comparées aux méthodes de résolutions de

[1] autres que les EIAO

SYRCLAD. Les critères de cette classification sont les sous-figures présentes dans la figure de l'exercice. En géométrie, les objets manipulés dans les exercices correspondent aux théorèmes du cours, on n'a pas besoin des connaissances de reformulation de SYRCLAD. Néanmoins, en étiquetant les sous-figures d'un problème et en mobilisant les connaissances pertinentes pour ce problème, GEOMUS réalise un *enrichissement du problème*. Le problème enrichi est un nouveau modèle du problème, orienté vers la résolution ; on peut le comparer aux modèles opérationnels construits par SYRCLAD. Dans SYRCLAD, la classification est explicitée sous forme d'un graphe et elle est manipulée par le système. De plus, l'architecture de SYRCLAD est générale et a permis d'appliquer le système à plusieurs domaines.

3.1.2 Progé

Progé [Schreck 93] est un système qui résout des problèmes de construction de figures géométriques à la règle et au compas. Il est proche de SYRCLAD par certains aspects. En effet, un expert en géométrie doit définir un univers géométrique et une base de règles et les fournir comme donnée à l'architecture de Progé. Ces deux bases de connaissances associées à l'architecture du système forment un résolveur de problèmes de construction de figures géométriques. L'expert a aussi la possibilité d'agir sur une base de règles et un univers géométrique déjà définis. L'utilisateur présente au résolveur ainsi défini un problème sous la forme d'un énoncé Prolog, et le système produit un plan de construction de la figure.

Les différences avec SYRCLAD sont d'une part que le raisonnement utilisé par le moteur du système est un moteur d'inférence d'odre 1, et pas un raisonnement par classification, et d'autre part que l'architecture ainsi définie n'a pas été appliquée à un autre domaine que la construction de figures géométriques à la règle et au compas. Un des intérêts de Progé vient d'une spécification algébrique de l'univers géométrique fourni au système, et d'une méta-spécification détaillant la mise en œuvre de cette spécification lors de la résolution de problèmes, ce qui permet de décrire rigoureusement les mécanismes utilisés dans ce système.

Nous évoquerons en partie II deux systèmes : Bateleur et Modélis. Bateleur [Nigro 95] est un système qui joue au tarot. Il doit en début de partie choisir un plan de jeu et utilise pour cela une base de règles, mais pas de classification explicite. Modélis [Tisseau 90] modélise des problèmes de thermodynamique énoncés en langage naturel, puis les résout, sans utiliser de classification. Nous pensons qu'utiliser une classification peut rendre cette phase de résolution plus efficace.

Nous allons maintenant présenter des systèmes d'intelligence artificielle qui utilisent une classification explicite des problèmes du domaine.

3.2 Raisonnement par cas

Le raisonnement pas cas [Riesbeck et Schank 89] [Kolodner 91] est une forme de raisonnement par analogie. Pour résoudre un problème, la démarche consiste à utiliser des problèmes déjà résolus et mémorisés appelés cas. On cherche un cas similaire au problème à résoudre, et on adapte la méthode de résolution employée pour le cas sélectionné au problème à résoudre. Pour pouvoir

rechercher un cas et comparer ce cas au problème, il est nécessaire de définir des indices de recherche caractérisant un problème. La recherche de bons indices est un problème difficile. L'ensemble des cas est souvent structuré (grâce aux indices) en un graphe qui est comparable à une classification. Le nouveau cas résolu par adaptation d'un ancien cas est parfois ajouté à la base de cas, ce qui peut être considéré comme une méthode d'apprentissage.

La figure 1 illustre les principales étapes du raisonnement par cas. Tous les systèmes utilisant le raisonnement par cas n'effectuent l'ensemble du processus illustré par cette figure. Certains réalisent uniquement la recherche d'un cas, d'autres effectuent également l'étape d'adaptation.

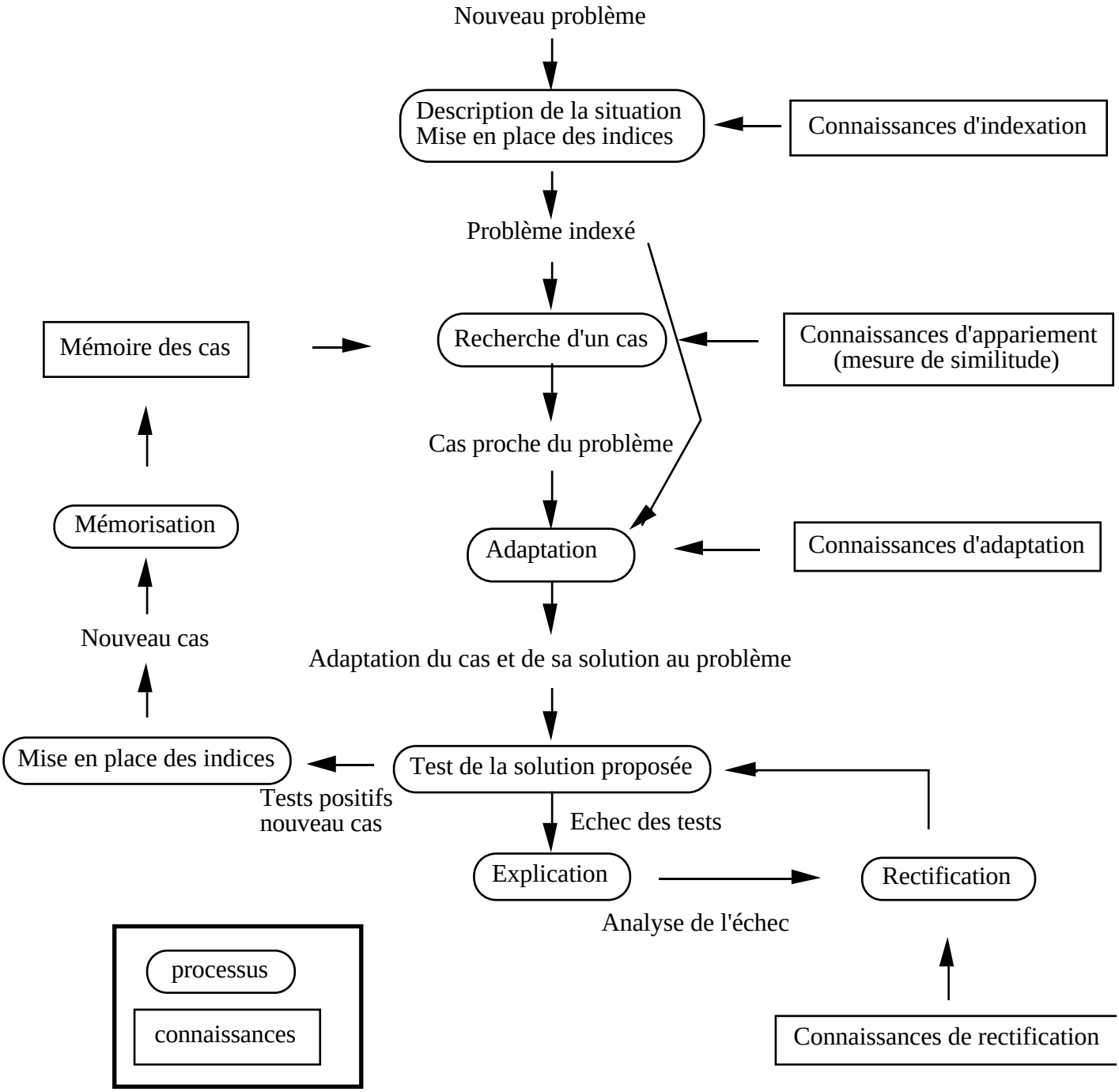


Figure 1 : raisonnement par cas

SYRCLAD utilise une démarche qui est plus proche du raisonnement par classification que du raisonnement par cas. Pourtant, chercher à quelle classe de problèmes connue appartient un problème à résoudre relève d'une démarche analogue à la comparaison du problème à résoudre avec des problèmes déjà résolus.

A. Napoli compare le raisonnement par classification et le raisonnement par cas [Napoli 92a] :

> « Leur fonctionnement de base est identique : ils sont guidés par l'expérience, comme le suggère le principe de remémoration. Toutefois, à l'image des scripts, une expérience dans une mémoire de cas a le plus souvent un rôle d'"épisode temporel" à adapter à une situation courante, que n'ont pas forcément des objets d'une hiérarchie sur laquelle opère le raisonnement par classification. [...] Un parallèle peut être fait entre le raisonnement par classification et les deux premières phases du raisonnement par cas, à savoir la description de la situation courante et la recherche d'un cas. Les phases suivantes d'apprentissage et d'acquisition de connaissances ne font pas partie du cycle standard du raisonnement par classification. »

On peut en effet comparer la première partie du raisonnement par cas, qui cherche un cas proche du problème à résoudre, avec un raisonnement par classification. Nous avons vu au paragraphe 1 qu'il semble que le raisonnement pas cas soit une première étape de raisonnement dans l'acquisition d'une expertise. La généralisation des cas donnerait ensuite lieu à la construction de schémas ou de classes.

3.3 Classification heuristique

SYRCLAD utilise une stratégie proche de la classification heuristique [Clancey 85]. A partir de *données spécifiques* (l'énoncé du problème), il construit des *données abstraites* (une reformulation du problème) durant une *phase d'abstraction*. Ces données abstraites sont associées de manière *heuristique* à une *solution générique* de la hiérarchie préétablie (ici une classe de problèmes à laquelle est associée une méthode de résolution). Cette solution est spécialisée pour convenir au problème posé.

SYRCLAD réalise la reformulation du problème (abstraction de données) et le classement (association heuristique) en même temps, en utilisant le graphe de classification. De plus, la solution n'est pas obtenue par raffinement d'une solution générique, mais par l'application d'une méthode de résolution qui peut être complexe. Cette méthode peut en effet impliquer la *construction* de solutions, que Clancey oppose à la *sélection* de solutions (comme la classification heuristique).

Il faut noter que dans la plupart des systèmes basés sur la classification, et en particulier dans le cadre de la classification heuristique, les valeurs des attributs du problème qui permettent la classification sont connues ; on peut considérer les données abstraites comme un problème "classifiable". Dans de nombreux systèmes, si les valeurs des attributs manquent, elles peuvent être demandées à l'utilisateur ou calculées par des réflexes procéduraux. Il est important de remarquer que, contrairement à ces systèmes, dans SYRCLAD les valeurs des attributs du problème à classer ne sont pas connues ; SYRCLAD ne sait même pas quels attributs seront pertinents pour le problème qu'il doit classer.

3.4 Systèmes à subsomption : organiser des classes

L'utilisation d'un graphe de classification hiérarchique rappelle le problème du classement d'une description, traité par les travaux issus de KL-ONE [Brachman et Schmolze 85], que l'on regroupe souvent sous le terme de systèmes à subsomption, et dont font partie les logiques terminologiques [Napoli et Volle 93]. Ces systèmes définissent un langage qui offre la possibilité de construire des descriptions de concepts, et une relation de subsomption qui permet d'ordonner hiérarchiquement ces descriptions. On distingue les descriptions de concepts génériques (analogues aux classes) et les descriptions d'individus (analogues aux instances). Deux problèmes de classification peuvent alors apparaître, suivant que l'on considère un concept ou un individu.

Dans le premier cas il s'agit d'insérer le concept dans une hiérarchie. Il y a alors une construction incrémentale d'une hiérarchie. Par exemple le système YCHEM [Napoli et Laurenço 93, Napoli 92b] construit une structure moléculaire M en insérant M dans une hiérarchie de structures réactives H. Cette hiérarchie H est construite au fur et à mesure que d'autres structures sont insérées dans H.

Dans le second cas, il s'agit de trouver le concept le plus spécifique dont l'individu est une instance. Notre système est proche du second cas : SYRCLAD classe des instances, il ne construit pas de graphe de classification par insertion de nouvelles classes. Nous allons décrire quelques systèmes qui font partie de cette approche.

3.5 Classer des instances

Utiliser BACK pour classer des sons

P.-Y. Rolland et F. Pachet ont utilisé BACK [Hoppe et al 93] pour représenter une classification de sons établie en fonction des transformations qu'ils peuvent subir [Rolland et Pachet 95]. Un module Smalltalk demande à BACK de classer une instance de son par rapport à cette classification, et BACK lui transmet le résultat de ce classement, avec les transformations envisageables. Un système en Smalltalk gère l'interface utilisateur et applique les transformations grâce à des méthodes.

SHIRKA

SHIRKA [Rechenman et al 90] est un système de représentation par objets ayant l'approche classe-instance, il permet la classification d'instances. SHIRKA permet la manipulation d'instances incomplètes, la relation d'appartenance à une classe peut être sûre, possible, ou impossible. B. Rousseau et F. Rechenmann proposent dans [Rechenmann et Rousseau 92] de construire des environnements de résolution de problèmes pour aider un utilisateur à choisir dans une bibliothèque de calcul scientifique la méthode la mieux adaptée à une tâche donnée. Le mécanisme d'exploitation envisagé alterne des phases de classification permettant de spécialiser une tâche et des phases de décomposition en sous-tâches. Les valeurs des attributs des objets à classer sont demandées à l'utilisateur ou calculées par des réflexes procéduraux. Le but de ces environnements est de guider l'utilisateur à travers un grand graphe de classification-décomposition, afin de choisir une méthode parmi les nombreuses proposées.

Dans les domaines que nous abordons avec SYRCLAD, il n'y a qu'un nombre limité de méthodes de résolution ; la difficulté ne vient pas d'un choix parmi un grand nombre de méthodes mais d'une modélisation nécessaire du problème afin de savoir quelle méthode est applicable. Les graphes de classification utilisés par SYRCLAD ne sont pas très importants puisqu'ils doivent pouvoir être mémorisés par un être humain.

SCARP

SCARP [Willamovski 94] est un système coopératif d'aide à la résolution de problèmes qui utilise SHIRKA pour représenter les connaissances. SCARP est centré sur l'interaction avec l'utilisateur qui peut contrôler une phase de résolution, fournir des valeurs de paramètres manquantes ou intervenir sur un choix stratégique. SCARP utilise alternativement des phases de classification et des phases de planification hiérarchique. Les domaines considérés appartiennent au calcul scientifique, comme le traitement du signal ou l'analyse de données. La classification est utilisée pour guider la résolution suivant la forme des données.

TRAM

TRAM [Chaillot et Crochon 94] est un environnement de résolution de problèmes qui gère l'exécution de modules algorithmiques dans l'utilisation de bibliothèques de codes de calcul scientifique. A une tâche à exécuter est associé un ensemble de méthodes qui réalisent cette tâche. Ces méthodes sont organisées sous forme de hiérarchie de spécialisation. Les méthodes associées à une tâche élémentaire sont les modules algorithmiques qui effectuent cette tâche avec des performances différentes en fonction des valeurs des entrées de la tâche. Le système est utilisé pour assurer le contrôle d'un robot autonome mobile, il acquiert des données sur les modifications de l'environnement par l'intermédiaire de capteurs. Un objet décrivant l'état courant de la résolution est créé, les attributs de cet objet étant affectés par les valeurs d'entrée de la tâche à résoudre. Cet objet est classé afin que le système sélectionne la méthode activable la plus performante. M. Chaillot et E. Crochon définissent l'utilité d'une méthode pour réaliser une tâche à partir d'un état, en fonction de critères de temps, précision et place mémoire, c'est-à-dire des valeurs numériques. La classification a pour objectif d'améliorer la sélection courante d'une méthode, le gain espéré par une meilleure méthode doit être significatif par rapport au coût en temps de la classification. TRAM utilise la classification pour contrôler le temps de réponse d'un système autonome et réactif.

SYRCLAD et le classement d'instances

La tâche de SYRCLAD relève du classement d'instance, il n'ajoute pas de classe à une hiérarchie. Cependant, sa tâche ne consiste pas seulement à classer une instance.

En effet, le classement en analyse de données ou en logique terminologique prend en entrée une description D et produit en sortie une classe C (la plus spécifique possible) telle que D soit instance de C.

Supposons que nous devons classer le problème m10 (chapitre 1) à l'aide du graphe de classification des problèmes de dénombrement. La description D serait alors la suivante :

attribut	valeur
type du problème	répartition
type de l'objet formé	liste
remise	avec

Le classement de D utilise le graphe de classification et trouve la classe C : "répartition-liste-avec_remise", sans avoir besoin d'utiliser de connaissances de reformulation.

Dans SYRCLAD, étant donné un problème P (un individu au sens les logiques terminologiques) décrit par un modèle descriptif D (cf. chapitre 1 §1), le système doit trouver la classe opérationnelle la plus spécifique C telle qu'il existe une instance M de C représentant un problème équivalent à P. La différence avec le paragraphe précédent vient du fait que le problème n'est pas décrit au départ dans le langage de description des individus associés à une classe.

> Si nous reprenons le problème m10 (chapitre 1), D est ici le modèle descriptif du problème P (§1), et M le modèle opérationnel de P (§5). M est instance de la classe C : "répartition-liste-avec". Le système a dû utiliser des connaissances de reformulation pour construire M.

Par conséquent, dans SYRCLAD, la donnée d'entrée D n'est pas directement instance de la classe de sortie C. Elle est seulement équivalente[1] à une autre description M qui, elle, est instance de C. SYRCLAD classe uniquement des problèmes, et il utilise la notion particulière d'*équivalence* entre deux problèmes. L'introduction de ce classement modulo l'équivalence des problèmes rend d'une part la tâche plus difficile, car cela introduit un quantificateur existentiel ("il existe M") et pose le problème de l'unicité du résultat. En effet, deux problèmes équivalents n'appartiennent pas nécessairement à la même classe.

> Par exemple, soit P le problème : « Dans un jeu de 32 cartes, combien y a-t-il de mains de 3 cartes contenant 3 piques ? ».
>
> Soit M1 : « Soit un ensemble E de taille 32 divisé en 4 catégories de taille 8. Combien y a-t-il d'ensembles de 3 éléments dont 3 sont pris dans une catégorie de E et 0 dans les autres catégories ? »
>
> Soit M2 : « Soit E un ensemble de taille 8, combien y a-t-il d'ensembles de trois éléments pris dans E ? ».
>
> P, M1 et M2 sont des problèmes équivalents, mais M1 est instance de la classe "répartition-ensemble" alors que M2 est instance de la classe "combinaison". En fait "combinaison" est une sous-classe de "répartition-ensemble", mais il est possible également de considérer que m10 est un problème de la classe "spectre-ensemble", cette classe faisant partie d'une autre branche du graphe de classification (chapitre 1 §2). On voit donc qu'un problème peut être modélisé de plusieurs manières et donc être équivalent à plusieurs modèles appartenant à des classes différentes. Nous verrons dans le chapitre 3 que SYRCLAD choisit *une* modélisation, suivant les connaissances que l'expert lui a données.

L'introduction de ce classement modulo l'équivalence des problèmes augmente d'autre part le champ des possibilités (il y a beaucoup de problèmes équivalents à un problème donné, puisqu'un problème peut être modélisé de plusieurs manières).

Dans les domaines abordés par SYRCLAD, les classes font intervenir un vocabulaire adapté aux méthodes de résolution, basées sur des principes abstraits, tandis que la représentation initiale du problème fait intervenir un vocabulaire décrivant une situation concrète. Le problème ne peut donc pas être classé directement à partir de sa formulation initiale, le système doit construire une autre représentation pour pouvoir classer le problème. SYRCLAD construit cette représentation et classe le problème en même temps. En fait, dans SYRCLAD, on ne s'intéresse pas au classement pour lui-

---

[1] Deux problèmes sont équivalents s'ils ont la même solution.

même, mais parce qu'il fournit une méthode de résolution : **on classe pour résoudre**. Et pour cela, on a besoin non seulement de la classe C, mais aussi de l'instance M. Ce sont deux problèmes liés, puisqu'une façon de prouver l'existence d'un M est justement d'en exhiber un. Le classement au sens de SYRCLAD implique pratiquement la génération d'une instance (ce n'est pas nécessaire car on pourrait affirmer une existence à l'aide de théorèmes d'existence non constructifs). On peut ainsi considérer qu'engendrer une instance M, c'est "modéliser" ou "opérationnaliser" le problème de départ. La signification que nous attribuons au mot "classement" éclaire le fait que "classer" et "modéliser" sont deux processus simultanés dans SYRCLAD.

3.6 Construction du graphe de classification

Dans SYRCLAD, un graphe de classification est donné par l'expert pour chaque domaine d'application. Nous avons vu (§3.3) que certains systèmes peuvent ajouter des classes à un graphe existant, afin d'augmenter ce graphe. D'autres systèmes construisent eux-mêmes un graphe de classification, à partir d'exemples de problèmes résolus.

Ces systèmes construisent des hiérarchies d'objets qui sont appelés classes, catégories, prototypes ou MOPs. Ces classes sont décrites avec des termes qui apparaissent dans l'énoncé du problème ou dans la résolution de celui-ci. Dans SYRCLAD, les classes sont des objets de niveau méta par rapport au domaine. Les attributs intervenant dans ces classes peuvent aussi être de niveau méta, comme le nombre de contraintes présentes dans l'énoncé, ou le nombre d'objets d'un certain type.

Nous voulons ici mentionner deux systèmes, ALEX [Séroussi et Morice 93] et EUREKA [Elio et Scharf 90], qui construisent des graphes de classification en utilisant une méthode analogue au raisonnement par cas.

ALEX apprend à partir d'exercices résolus, puis utilise les prototypes construits pour résoudre de nouveaux exercices par analogie ; le domaine est celui du calcul de limite de suites numériques. B. Séroussi et V. Morice estiment qu'une résolution de problèmes est divisé en deux étapes : la *compréhension du problème* pour obtenir une formulation (appelée *élément significatif*) à partir de laquelle la *construction de la solution* est procédurale. Pour les novices, l'élément significatif est le dernier pas avant la solution alors que pour les experts, c'est la formulation initiale du problème. Quand on propose un exemple de problème résolu à ALEX, il mémorise la première situation (non déjà apprise) qui le mène à la solution par une méthode déjà apprise, c'est-à-dire l'élément significatif. En généralisant le problème[1], il acquiert ainsi une nouvelle règle situation -> méthode, la méthode pouvant résumer un enchaînement de pas élémentaires. Cette règle est représentée en mémoire comme un cas ou un prototype. Ces prototypes sont ensuite utilisés pour résoudre d'autres problèmes par analogie.

[1] Les problèmes étant des calculs de limite de suite numériques, ALEX généralise le terme de la suite 3n+1 par Kn+K'.

EUREKA est appliqué aux exercices de mécanique sur les forces et les énergies, il résout des exercices grâce à un système à base de règles, et construit des prototypes par comparaison avec des exercices résolus. Une résolution d'exercice est stockée dans un P-MOP[1]. Chaque exercice résolu est comparé au P-MOP à travers des indices. Les P-MOP sont indexés les uns par rapport aux autres par leurs indices de même valeur (ressemblance) et leurs indices de valeurs différentes (différence). Les étapes de résolutions communes à plusieurs exercices sont généralisées.

Les prototypes représentés par les P-MOP sont d'abord basés sur des traits de surface du problème, puis le nombre d'exercices résolus augmentant, ce sont des traits liés aux concepts fondamentaux de la physique qui apparaissent. EUREKA modélise ainsi le comportement d'un novice qui s'améliore avec l'expérience. Les index des prototypes d'EUREKA sont des attributs du problème présents dans les énoncés des problèmes ou apparaissant dans la résolution, EUREKA ne peut donc pas découvrir d'attributs de niveau méta.

Ces systèmes modélisent comment un novice construit des schémas à partir de problèmes résolus. Nous avons vu que cette démarche est peut-être une première étape avant d'être expert et d'utiliser une classification comme celles qu'utilise SYRCLAD. En effet, SYRCLAD ne modélise pas la construction d'une classification, mais utilise des classifications fournies par un expert du domaine. Ces classifications permettent d'expliciter des méthodes de modélisation et de résolution qui sont des métaconnaissances. Elles peuvent en particulier faire intervenir des attributs de niveau méta qu'il est difficile de découvrir. Le novice qui s'est amélioré, tel qu'il est modélisé par EUREKA, construit des schémas contenant des méthodes de résolution, mais n'utilise pas ces méthodes de résolution pour résoudre de nouveaux problèmes. En effet, les schémas construits ne sont pas assez opérationnels puisqu'ils ne permettent pas de résoudre directement. Au contraire, SYRCLAD applique les méthodes de résolution qu'il sait adaptées aux problèmes d'une certaine classe, ainsi que le fait l'élève-expert.

4 Conclusion

Nous avons évoqué dans ce chapitre des travaux de psychologie cognitive et de didactique qui ont motivé la réalisation de SYRCLAD. En effet, la littérature montre que les experts se distinguent des novices parce qu'ils possèdent une bonne expertise de classification des problèmes qui leur permet d'être efficace dans la modélisation et la résolution de ces problèmes. De plus, si l'on veut utiliser SYRCLAD comme résolveur de référence dans un EIAO, il est important que la démarche de résolution utilisée par le système soit explicite et déclarative, de manière à faciliter (pour un futur EIAO) la production d'explications. Nous avons choisi dans SYRCLAD de distinguer trois types de connaissances : les connaissances de classification, de reformulation et de résolution. Cette représentation des connaissances et le fonctionnement du noyau de SYRCLAD qui les utilise permettent de résoudre des problèmes concrets en construisant un nouveau modèle du problème, pour se ramener aux connaissances de la théorie du domaine.

[1] Problem Memory Organization Packet

Nous avons souligné que SYRCLAD se distingue d'autre systèmes d'IA utilisant des classifications pour résoudre les problèmes. En effet, ces systèmes sont souvent utilisés pour choisir une méthode de résolution parmi de nombreuses méthodes possibles, en utilisant un graphe de classification de grande taille. Au contraire, SYRCLAD utilise des graphes de classification de petite taille, mémorisables par un être humain. La difficulté vient par contre du fait que les problèmes à classer ne sont pas exprimés dans les termes qui permettent de les classer. Par conséquent, il est nécessaire de procéder en même temps que le classement à une modélisation du problème. En fait, le graphe de classification permet de construire un problème équivalent au problème posé pour lequel on connaît une bonne méthode de résolution.

Le chapitre 3 est destiné à détailler l'architecture de SYRCLAD et à expliquer le fonctionnement de son noyau.

Chapitre 3
Architecture et fonctionnement de SYRCLAD

Dans ce chapitre, nous présentons l'architecture de SYRCLAD et le fonctionnement de son noyau, qui sont indépendants de tout domaine d'application. Nous expliciterons les connaissances que l'expert doit donner à SYRCLAD, et comment elles sont organisées ; nous étudierons également la démarche qu'il emploie pour classer un problème et construire son modèle opérationnel ; enfin nous évoquerons le langage utilisé pour poser des problèmes à SYRCLAD et pour exprimer les résultats obtenus.

Plan du chapitre

Nous avons vu au chapitre 1 que l'on peut distinguer trois niveaux dans SYRCLAD.

Tout d'abord le noyau de SYRCLAD, indépendant du domaine. Ce noyau est conçu et implémenté par le concepteur du système. Il fournit un cadre conceptuel destiné à représenter les connaissances. Nous verrons dans ce chapitre qu'il impose un découpage des connaissances en différentes bases, suivant des syntaxes obligées, et qu'il fournit de plus un vocabulaire pour relier les différentes bases (attachement de règles à un attribut, attachement d'une méthode à une classe). Le noyau fournit également un mécanisme d'utilisation de ces connaissances.

Ensuite, l'expert d'un domaine X fournit, dans le cadre conceptuel exigé par le noyau, les connaissances de classification, de reformulation et de résolution spécifiques au domaines. Il définit également un langage de la logique du premier ordre en précisant le vocabulaire utilisable pour parler des problèmes du domaine. Le noyau et les bases de connaissances forment un résolveur de problèmes du domaine X, que l'on peut appeler SYRCLAD-X.

Enfin, un utilisateur soumet à SYRCLAD-X un problème à résoudre, sous la forme d'un *modèle descriptif* de ce problème exprimé dans le langage défini par l'expert. Pour classer ce problème, SYRCLAD-X utilise un *graphe de classification* de problèmes que l'expert lui a donné. Ce graphe explicite les connaissances de classification. Pour utiliser le graphe de classification, SYRCLAD-X a besoin de règles permettant de déterminer la valeur des *attributs discriminants* intervenant dans la classification. Ces règles sont les *connaissances de reformulation*, également fournies par l'expert.

Le noyau de SYRCLAD effectue l'"opérationnalisation" du problème. Il produit d'une part une classe (si possible opérationnelle) et d'autre part un nouveau modèle du problème, appelé *modèle opérationnel*. L'expert associe à chaque classe du graphe de classification une méthode de résolution adaptée. Cette association classe -> méthode explicite les *connaissances de résolution*. Pour obtenir une solution du problème, SYRCLAD applique la méthode de résolution associée à la classe du problème sur le modèle opérationnel.

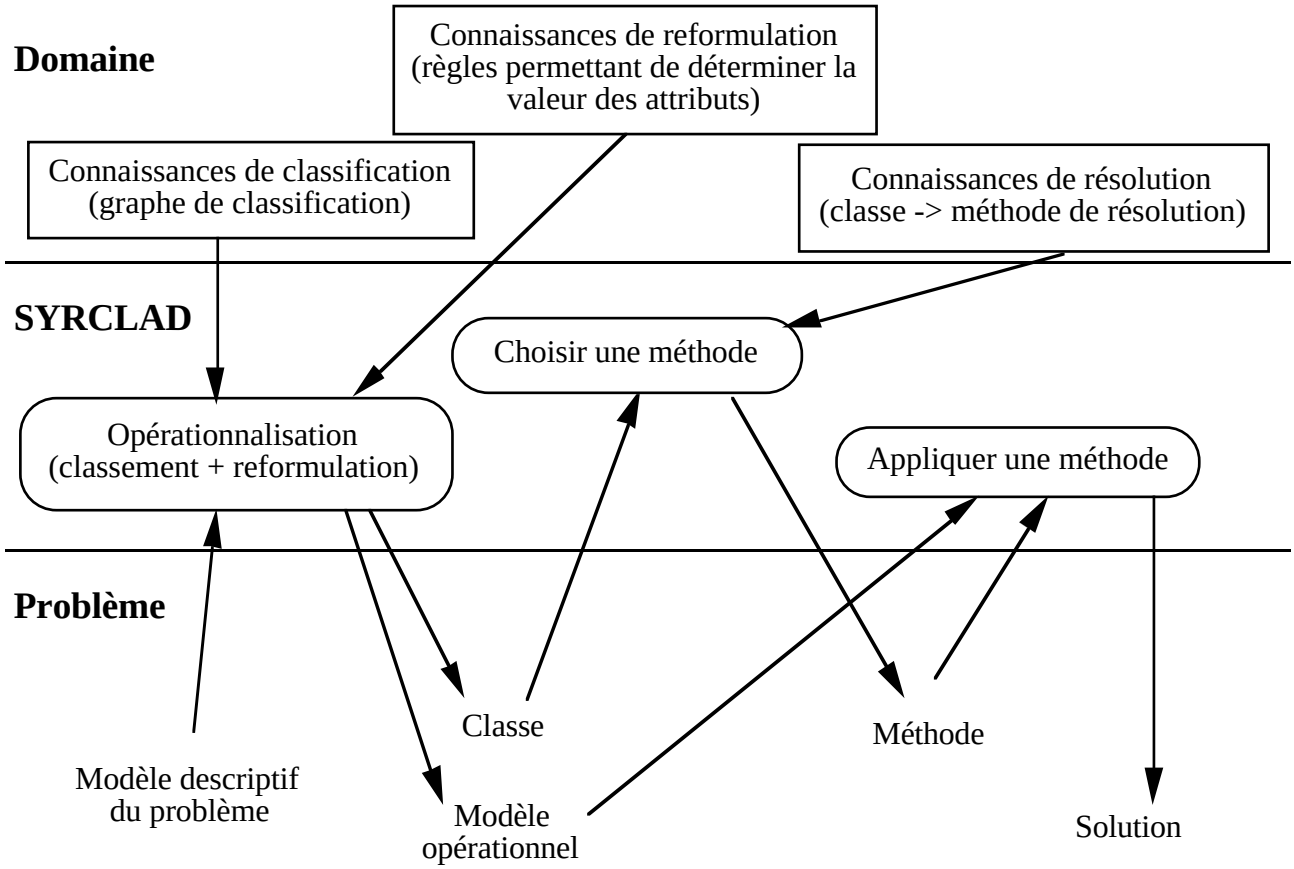


Figure 1 : l'architecture de SYRCLAD

Le niveau "domaine" contient des *bases de connaissances*, spécifiques au domaine, que nous avons représentées par des rectangles. Le niveau "SYRCLAD" contient des *processus* qui utilisent les bases de connaissances du domaine ; nous avons représenté les processus en arrondi.

L'architecture de SYRCLAD et le processus de résolution dans son ensemble sont indépendants du domaine. Le fonctionnement de SYRCLAD est le même dans les quatre domaines, seul le contenu des trois bases de connaissances change suivant le domaine. L'algorithme de classement et de construction du modèle opérationnel est le même quel que soit le domaine. Les bases de connaissances de classification, de reformulation et de résolution doivent être données par l'expert dans un formalisme unique quel que soit le domaine. Nous avons défini ce formalisme afin que ces bases de connaissances soient utilisables par le noyau de SYRCLAD. Les modèles descriptifs et opérationnels doivent être exprimés dans un langage de logique du premier ordre ; les prédicats pouvant être utilisés sont eux spécifiques au domaine.

Nous allons reprendre, comme dans le chapitre 1, chaque étape du déroulement du processus de résolution d'un exercice, en insistant sur ce qui est indépendant du domaine.

1 Modèle descriptif du problème

Les problèmes considérés représentent des *situations concrètes*. On y parle de mains de cartes, de parties de billes, du jeu de tarot, de gazomètres. Les exercices ne sont pas donnés à SYRCLAD en langue naturelle, mais sous forme de modèle descriptif dans un formalisme logique. Le modèle descriptif du problème décrit une situation concrète, qui est celle présentée dans l'énoncé. Les prédicats utilisés dans ce modèle descriptif décrivent parfois des *traits de surface* du problème posé [Chi et al 81]. Considérons par exemple les deux exercices suivants :

> Dans un jeu de 32 cartes, combien y a-t-il de mains de 5 cartes contenant 2 piques et deux carreaux ?
>
> Un sac contient 8 jetons rouges, 8 jetons verts, 8 jetons bleus et 8 jetons jaunes. On tire simultanément 5 jetons dans le sac. Combien y a-t-il de tirages contenant 2 jetons rouges et deux jetons jaunes ?

Ces deux exercices sont différents par leurs traits de surface (des cartes pour l'un, des jetons pour l'autre), mais il ont la même structure profonde. Il est possible de trouver une même formulation abstraite qui soit équivalente aux deux problèmes à la fois. SYRCLAD est justement capable de trouver cette formulation, et de déterminer une classe de problèmes dont elle est une instance. Il résoudra les deux exercices de la même façon. Les traits de surface ne modifieront pas la résolution.

Dans SYRCLAD, qui est implémenté en Prolog, les modèle descriptifs sont exprimés dans un langage de logique du premier ordre. Un modèle descriptif est représenté par un ensemble de prédicats. Dans chaque domaine, l'expert définit les prédicats utilisables pour représenter un modèle descriptif de problème. Ces prédicats décrivent des objets qui ont des propriétés et sont reliés par des relations. Il y a aussi une question à laquelle il faut répondre pour résoudre le problème. Les prédicats sont spécifiques au domaine du problème.

Le seul prédicat indépendant du domaine présent dans tous les modèles descriptifs est le prédicat *problème(nom_du_problème)*. Il permet de nommer le problème, ce qui permet d'en parler, et d'y attacher des informations déduites pendant le processus d'opérationnalisation. De plus, dès que l'on décompose un problème en sous-problèmes, il est essentiel de savoir de quel problème on parle.

On introduit souvent les objets présents dans les modèles descriptifs à l'aide du prédicat *est_un(objet, type)*, affirmant qu'un objet (le premier argument) est instance d'un type d'objet (le second argument).

Dénombrements

```
est_un(a,action_tirer)./* soit a une action de tirer des éléments */
est_un(e,ensemble).   /* soit e un ensemble d'éléments */
est_un(s,sac).        /* soit s un sac */
```

Problèmes additifs

```
est_un(p,partie_de_billes). /* soit p une partie de billes */
est_un(isabelle,personne).  /* soit isabelle une personne */
```

Tarot

```
est_un(p,partie_de_tarot).  /* soit p une partie de tarot */
```

Thermodynamique

```
est_un(gazo,gazomètre).     /* soit gazo un gazomètre*/
```

Le prédicat "est_un" n'est jamais utilisé par le noyau de SYRCLAD ; donc même s'il est utilisé dans plusieurs domaines, il est dans chacun d'entre eux du niveau "domaine"[1]. Le deuxième argument du prédicat "est-un" est aussi de niveau "domaine" ; action_tirer, partie_de_billes, partie_de_tarot, gazomètre sont des concepts spécifiques aux domaines. Quant au premier argument du prédicat "est_un" (a, e, s, p, isabelle, gazo), il est de niveau "problème".

D'autres prédicats permettent d'exprimer des relations entre ces objets, décrivant ainsi une situation. Ces prédicats sont tous de niveau "domaine", leurs arguments étant de niveau "problème".

Dénombrements

```
récipient(a,s). /* pour l'action-tirer a, le récipient dans lequel on tire
est le sac s (a et s sont définis plus faut) */

contenu(s,e).
/* le contenu du sac s est l'ensemble e défini plus haut */
```

Problèmes additifs

```
participants(p,isabelle,yves).
/* les participant de la partie p (définie plus haut) sont isabelle
(définie plus haut) et yves (également une personne) */
possède(isabelle,ens).
/* isabelle possède ens, qui sera défini comme un ensemble de billes */
```

Tarot

```
contrat(p,j,c).
/* dans la partie p, le joueur j a choisi le contrat c */

main(p,j,m).
/* dans la partie p, le joueur j possède la main de cartes m */

écart(p,j,é).
/* dans la partie p, le joueur j a fait l'écart é */
```

Thermodynamique

```
trans_gp(sys,i,f).          /* le système gazeux sys subit une
transformation entre les instant i et f */
```

---

[1] Cela illustre le fait que les vocabulaires des différents domaines peuvent avoir des symboles en commun. Comme on traite les différents domaines dans des systèmes différents, on ne peut pas avoir de conflits de noms. Il n'est pas alors obligatoire que le même symbole utilisé dans des domaines différents ait la même sémantique . Il se trouve que est_un a bien la même sémantique dans les différents domaines.

```
équi_gazomètre(gazo,sys,atm,i).
/* le gazomètre gazo établit un équilibre entre les deux systèmes gazeux
sys et atm (l'atmosphère) à l'instant i */
```

Le modèle est enrichi par des propriétés sur les objets. De même que pour les prédicats exprimant les relations entre objets, les prédicats exprimant les propriétés sont tous de niveau "domaine", leurs arguments étant de niveau "problème".

Dénombrements

```
simultanément(a).
/* dans l'action-tirer a, les éléments sont tirés simultanément */

effectif_attribut(m,2,couleur,pique).
/* l'objet formé m doit respecter la contrainte suivante : il doit contenir
2 éléments de couleur pique */
```

Thermodynamique

```
isotherme(sys,i,f).
/* le système gazeux sys est isotherme entre les instants i et f */
```

Il faut enfin savoir quelle est la question à laquelle il faut répondre. Il y a pour chaque domaine un unique prédicat permettant de poser la question ; ce prédicat est de niveau "domaine" et ses arguments sont de niveau "problème", le premier d'entre eux est le problème.

Dénombrements

```
à_dénombrer(pb,ens).      /* il faut dénombrer l'ensemble ens */
```

Problèmes additifs

```
à_calculer(pb,entier1).   /* il faut calculer l'entier "entier1" */
```

Tarot

```
à_déterminer(pb,plan_de_jeu).
/* il faut déterminer un plan de jeu pour la partie pb */
```

Thermodynamique

```
on_demande(pb,volume(sys,i)).
/* il faut calculer le volume du système sys à l'instant i */
```

En partie II seront donnés de nombreux exemples de modèles descriptifs pour les quatre domaines. Le lecteur trouvera en annexe la liste des prédicats utilisables dans chaque domaine d'application. Dans un domaine, la liste de ces prédicats de la logique du premier ordre définit le langage qui permet d'exprimer des modèles descriptifs pour ce domaine.

Pour proposer un nouvel exercice à SYRCLAD-X, l'utilisateur du système doit utiliser le langage de la logique du premier ordre défini par l'ensemble des prédicats utilisables dans le domaine X. Si SYRCLAD-X était utilisé dans le cadre d'un EIAO, l'utilisateur de SYRCLAD-X ne serait pas l'élève : on ne peut pas demander à un élève de formuler un exercice dans le langage de la logique du premier ordre défini pour le domaine. On peut éventuellement passer par une interface (inspirée par exemple

de [Le Calvez et al 97]) qui demanderait à l'élève de spécifier certaines caractéristiques du problème ; le système construirait ainsi un modèle du problème composé de prédicats. Sans cette interface, l'utilisateur potentiel de SYRCLAD-X dans le cadre d'un EIAO est le professeur. On peut en effet demander au professeur qui veut ajouter un exercice à la base d'exercices de SYRCLAD-X de formuler cet exercice dans le langage de la logique du premier ordre défini pour le domaine X.

Dans TALC [Desmoulins 94, Desmoulins 95], C. Desmoulins utilise un langage de construction géométrique (SCL) issu de l'utilisation de Cabri-Géomètre [Bellemain 92] par l'élève et un langage de spécification d'une figure géométrique (CDL) utilisé par l'enseignant. La construction de l'élève et la spécification du professeur sont traduites en un langage de la logique du premier ordre (LDL) afin d'être comparées.

2 Graphe de classification

Dans chaque domaine d'application, l'expert donne à SYRCLAD un graphe de classification de problèmes. La classification représentée par ce graphe est évidemment dépendante du domaine. Mais la structure de représentation du graphe et son utilisation sont les mêmes pour les quatre domaines.

2.1 Une hiérarchie de classes

2.1.1 Structure du graphe

Nous avons choisi de représenter une classification de problèmes par une hiérarchie de classes, suivant la relation de spécialisation / généralisation. Une classe C2 est une sous-classe d'une classe C1 si tout problème de classe C2 est aussi de classe C1. On définit la racine du graphe comme la classe la plus générale au sens du classement, les feuilles étant les classes les plus spécifiques au sens du classement.

2.1.2 Discrimination

Pour qu'une classification soit utilisable par SYRCLAD, il faut que les sous-classes d'une classe soient différenciées par la valeur d'un seul attribut du problème. Soient C1 et C2 deux sous-classes[1] d'une classe C. Un problème P appartenant à la classe C appartiendra à la classe C1 si l'attribut A de P prend la valeur V1. Il appartiendra à la classe C2 si l'attribut A de P prend la valeur V2. On appellera A l'*attribut discriminant* de la classe C, et V1 et V2 les *valeurs discriminantes* de cet attribut.

> **Ce choix d'un *unique* attribut discriminant les sous-classes par une *égalité* de valeurs peut être considéré comme très restrictif**. Nous pensons cependant qu'il facilite l'expression des connaissances et l'implémentation du processus d'opérationnalisation, sans pour autant contraindre démesurément l'expert. Imaginons que l'expert ait établi que pour discerner si un objet, instance d'une classe C, est instance d'une sous-classe S, il faut examiner non pas la valeur d'un attribut, mais le type de cette valeur ou une propriété de cette valeur. Par exemple une instance de la classe

[1] Une classe peut avoir plus de deux sous-classes.

Être_Humain appartient à la sous-classe Mineur si la valeur de l'attribut âge de cette instance est inférieure à 18 ans. Cette comparaison de valeurs peut se ramener à une égalité de valeur si l'on définit un (méta)attribut âge_inférieur_à_18, à valeurs booléennes. D'autre part, un attribut peut traduire une conjonction de propriétés du problème, et exprimer ainsi une condition sur la valeur de plusieurs attributs. Par exemple, une instance de la classe Candidat appartient à la classe Admis si sa note de math est supérieure à 10 *et* si sa note de français est supérieure à 10. On peut alors définir l'attribut deux_notes_supérieures_à_10, à valeurs booléennes[1]. En fait, tout se passe comme si l'attribut discriminant de la classe C traduisait la propriété : "sous-classe de C à laquelle j'appartiens" (ou "sous-classe de C suivant laquelle le problème peut être modélisé, sachant qu'il est déjà modélisable suivant la classe C") dont les valeurs seraient exactement les différentes sous-classes de C.

La valeur de cet attribut n'est pas obtenue par simple lecture des données, mais par calcul, inférences ou décision de modélisation. Dans le classement au sens de l'analyse de données, les valeurs des attributs sont des données (résultats de mesure par exemple) et les attributs sont imposés par les contraintes expérimentales (un attribut, c'est ce qui est mesurable). Dans ce cas, il n'y a pas de raison pour qu'il existe toujours un attribut discriminant. Ici, au contraire, les attributs sont inventés par l'expert et ne s'appliquent pas directement à l'objet de départ (le modèle descriptif) mais à un autre objet que l'on construit progressivement (le modèle opérationnel). De plus, si l'expert définit un attribut dans une classe, c'est le système qui détermine sa valeur. Cette valeur n'est qu'un résumé (condensé en un seul symbole) d'une propriété que possède le modèle descriptif. Pour étudier cette propriété, une expertise parfois complexe est nécessaire, la propriété pouvant être définie indirectement par des propriétés auxiliaires et des objets auxiliaires. Nous avons en effet remarqué au chapitre 1 que pour déterminer le type d'un problème de dénombrement, il fallait déterminer les valeurs de plusieurs autres attributs du problème. En effet, dans les domaines que nous avons abordés, la difficulté vient justement du fait qu'il n'est pas possible de classer un problème uniquement avec les informations données par le modèle descriptif. Les attributs discriminants peuvent se situer à un niveau méta, c'est-à-dire qu'ils peuvent se rapporter non seulement aux objets intervenant dans le problème, mais aussi à la structure même du problème. C'est le cas lorsque l'on parle en dénombrement du *nombre* de contraintes portant sur l'objet formé, ou du *type* des attributs[2] intervenant dans les contraintes.

La restriction d'un unique attribut discriminant peut sembler contraignante à l'expert. Néanmoins elle peut s'avérer utile, surtout lorsque l'expert (moi) est encore en apprentissage. En effet, en construisant le graphe de classification des problèmes de thermodynamique, j'avais construit la classe deux_systèmes-deux_instants. Je savais que les problèmes de gazomètres, de systèmes de surface ou d'ambiance[3] devaient constituer des sous-classes de cette classe. Mais existait-il un seul attribut permettant de discriminer ces trois sous-classes ? J'ai trouvé un attribut : la manière de relier deux systèmes. En effet, lorsque l'on a deux systèmes gazeux, on peut les relier par un gazomètre, par un système de surface, ou en faisant évoluer l'un dans l'ambiance de l'autre. Ce concept explicitant les différents types de liens entre deux systèmes peut-il s'avérer intéressant au moment de fournir des explications ? Seul un véritable expert pourrait le dire. De même, la classification sur papier que j'ai utilisée pour les problèmes additifs ne donnait que les classes terminales. J'ai donc dû construire des classes intermédiaires et des attributs discriminants pour ces classes. Ainsi sont apparues des questions comme : les opérateurs sont-ils de même type ou de types différents ? quel est l'opérateur manquant ? Il est peut-être intéressant d'introduire ces questions dans une démarche explicative.

---

[1] C'est également le cas en thermodynamique lorsqu'on souhaite repérer un cas particulier de la classe un_système-deux_instants. Cet attribut traduit la conjonction de plusieurs contraintes sur une grandeur invariante pendant la transformation, des grandeurs connues à l'instant initial et final, et une grandeur à déterminer.

[2] Attention, il s'agit ici des attributs de l'objet formé : hauteur, couleur, parité. Ces attributs-là sont spécifiques aux dénombrements et n'ont rien à voir avec les attributs du problème qui forment le modèle opérationnel.

[3] Lorsqu'un système gazeux évolue dans l'ambiance de l'autre système (souvent l'atmosphère).

2.1.3 Utilisation

Certaines classes sont assez spécifiques pour que l'on puisse leur associer une méthode de résolution. On les appelle les *classes opérationnelles*. D'autres classes sont des classes intermédiaires du classement, pas assez spécifiques pour qu'une méthode de résolution soit envisageable (tous les attributs ne sont pas déterminés). On les appelle les classes non-opérationnelles. L'expert doit donner pour chaque classe du graphe son *statut* : opérationnelle ou non-opérationnelle.

Si l'on veut que le graphe de classification soit utile pour trouver une méthode de résolution, il faut que les classes terminales du graphe soient opérationnelles, mais les classes opérationnelles ne sont pas forcément terminales. En effet, une classe opérationnelle peut avoir une sous-classe décrivant un problème plus spécifique pour lequel il existe une autre méthode de résolution, plus pertinente. Par exemple, pour résoudre une équation du second degré en x (cas général), on peut toujours calculer "delta" et appliquer les formules (méthode générale), mais dans le cas où le terme constant est nul (cas particulier), on peut mettre x en facteur et résoudre directement (méthode particulière).

2.2 Représentation d'une classe du graphe de classification

Une classe est un nœud du graphe de classification.
Une classe est représentée par (figure 2) :

- son nom ;
- son *statut* (opérationnelle / non-opérationnelle) ;
- l'ensemble des attributs d'instances
  - ceux hérités de la super-classe,
  - les nouveaux attributs d'instances, spécifiques de la classe ;
- l'ensemble des *critères* qui définissent la classe
  (ensemble d'égalités attribut = valeur).
- l'attribut qui permet de discriminer les différentes sous-classes.

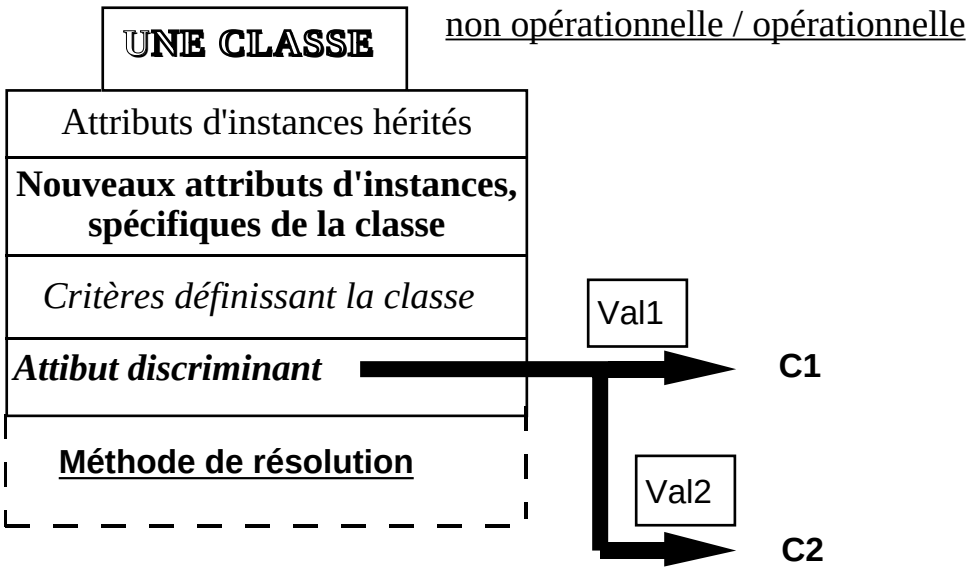


Figure 2 : une classe du graphe de classification

La figure 3 donne un exemple de classe :

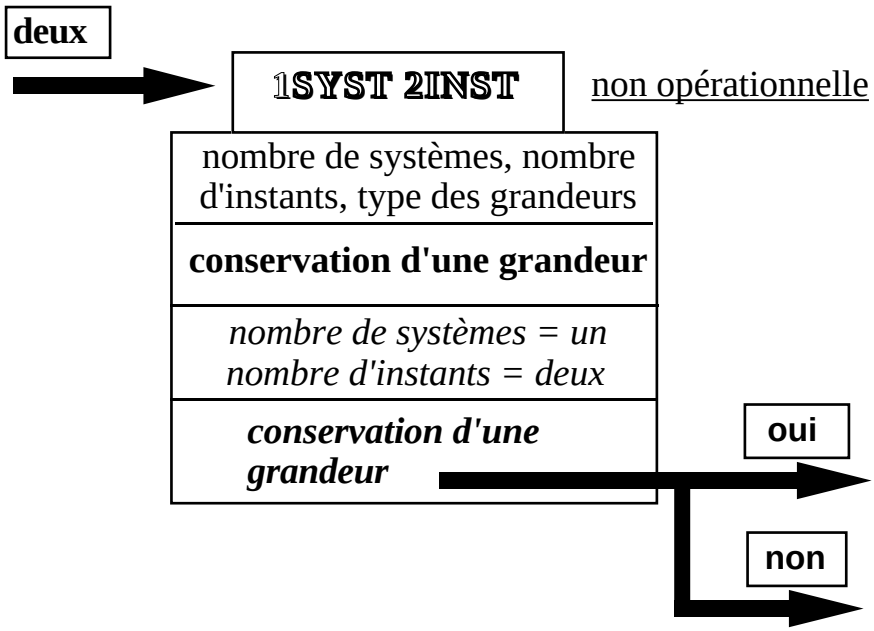


Figure 3 Il s'agit d'une classe du graphe de classification en thermodynamique. C'est la classe "un système, deux instants", sa sur-classe est la classe "un système". L'attribut qui discrimine cette classe par rapport à sa sur-classe est le nombre d'instants, égal à deux dans cette classe. Il y a trois attributs hérités de la sur-classe. Un nouvel attribut, la conservation d'une grandeur, apparaît dans cette classe ; en effet, on ne peut parler de conservation d'une grandeur que lorsque l'on sait qu'il y a deux instants dans le problème, cela n'aurait pas de sens avant. Les critères définissant la classe sont les valeurs des deux attributs discriminant cette classe par rapport à la racine du graphe. L'attribut qui discrimine les deux sous-classes est la conservation d'une grandeur. Cette classe n'est pas une classe opérationnelle, elle n'est pas assez spécifique pour que l'on puisse lui associer une méthode de résolution.

L'implémentation des graphes de classification est la même dans les quatre domaines. Sont attachés au nom de la classe, le statut de celle-ci, les nouveaux attributs spécifiques de cette classe, et l'attribut discriminant ainsi que les sous-classes qu'il détermine suivant ses valeurs.

```
statut(c_1s_2i,non_opérationnelle).
/* la classe un_système-deux_instants est non-opérationnelle */

attribut(c_1s_2i,P,conservation(P,B),...(§3)...):-
      eval_prem(problème(P)).
/* dans la classe un_système-deux_instants, un nouvel attribut est défini pour
le problème P : la conservation d'une grandeur [1]*/

test_fils(c_1s_2i,P,conservation(P,B)):- eval_prem(problème(P)).
/* l'attribut discriminant est la conservation d'une grandeur */

valeur_fils(c_1s_2i,P,c_1s_2i_cons):-
      eval_prem(problème(P)),eval_prem(conservation(P,oui)).
/* s'il y a conservation d'une grandeur, le problème P appartient à la sous-
classe un_système-deux_instants-conservation */
```

[1] eval_prem(fait) est une lecture dans la base de faits.

```
valeur_fils(c_1s_2i,P,c_1s_2i_sans):-
      eval_prem(problème(P)),eval_prem(conservation(P,non)).
/* s'il n'y a pas conservation d'une grandeur, le problème P appartient à la
sous-classe un_système-deux_instants-sans_conservation */

père(c_1s_2i,c_1s).
/* le père de la classe un_système-deux_instants est la classe un_système */
```

On ne précise pas les attributs hérités, puisqu'il existe un mécanisme d'héritage :

```
attribut(Classe,P,Att,...):-
      père(Classe,Père), attribut(Père,P,Att,...).
```

On ne donne pas non plus les critères définissant la classe, puisque pendant l'utilisation du graphe, les attributs correspondants d'un problème appartenant à cette classe ont forcément les valeurs de ces critères.

Les graphes de classification des quatre domaines étudiés seront donnés en partie II ainsi que dans les annexes.

3 Connaissances de reformulation

Les attributs discriminants intervenant dans les graphes de classification sont des attributs relevant de la théorie du domaine, ou des attributs de niveau méta[1]. La plupart du temps, ces attributs ne sont pas présents dans les modèles descriptifs des problèmes. Il est donc nécessaire d'avoir des connaissances permettant de déterminer la valeur de ces attributs. Ces connaissances permettent alors de passer d'un modèle descriptif à un modèle que nous appelons opérationnel, plus adapté au choix d'une méthode de résolution.

Pour que SYRCLAD puisse utiliser ces connaissances de reformulation, l'expert doit les exprimer par des règles si-alors. Dans une classe donnée, l'expert définit un attribut du problème si cet attribut a un sens pour les problèmes de cette classe. Il précise les règles qui permettent de trouver la valeur de cet attribut :

```
attribut(c_1s_2i,P,conservation(P,B),[isochore,isotherme,isobare,
isobare2,adiabatique,cons_inv1,cons_loi1],...):- eval_prem(problème(P))
/* l'attribut conservation est défini dans la classe un_système-
deux_instants, les règles qui permettent de déterminer sa valeur sont les
règles désignées par les symboles isochore, isotherme...*/
```

La complexité de ces règles varie beaucoup d'un domaine à l'autre et au sein d'un domaine donné. On a vu dans le premier chapitre qu'il faut déclencher 21 règles afin de déterminer la valeur du type du problème m10. Cela est dû au fait que les règles permettant de conclure sur la valeur de cet attribut (type du problème) font intervenir plusieurs autres attributs du problème, dont il faut alors déterminer

[1] Un attribut est de niveau méta lorsqu'il parle d'autres attributs du problème : le *nombre* de contraintes, le *type* des opérateurs, la *nature* du lien entre deux systèmes...

la valeur. Ensuite, pour déterminer le type de l'objet formé[1], SYRCLAD n'a eu besoin de déclencher qu'une seule règle ; cela est dû au fait que cette règle est plus simple, et fait intervenir moins d'attributs du problème, et également au fait que la valeur d'un attribut comme l'objet formé, intervenant dans cette règle, avait déjà été déterminée.

> **Un moteur d'inférence permet d'utiliser les règles de reformulation**. Ce moteur fait partie du noyau de SYRCLAD. Lorsqu'il faut déterminer la valeur d'un attribut du problème, ce moteur essaie de déclencher une des règles associées à l'attribut. Toutes ces règles concluent sur la valeur de l'attribut, il suffit donc d'en déclencher une. Le moteur essaie une règle après l'autre, dans l'ordre dans lequel elles sont données. Dans l'exemple ci-dessus, il essaiera d'abord de déclencher la règle "isochore", s'il échoue il essaiera la règle "isotherme"...On considère une base de faits, contenant (avant toute déduction) les faits du modèle descriptif du problème. Les prémisses des règles peuvent être soit des recherches de faits de la base de faits, soit des prédicats prolog permettant de faire des calculs ou de construire des objets. Les actions de la partie conclusion des règles peuvent être soit des modifications de la base de faits (on ajoute la valeur de l'attribut) soit des prédicats prolog permettant de faire des calculs ou de construire des objets. Lorsque le moteur essaie de déclencher une règle, il examine chacune des prémisses dans l'ordre où elles sont données. Si une prémisse fait intervenir la valeur d'un attribut et que cette valeur n'a pas encore été déterminée, le moteur utilise les règles associées à l'attribut pour déterminer sa valeur. Si toutes les prémisses ont été vérifiées, le moteur exécute la partie action de la règle.

Pour certains attributs, c'est un paquet de règles qui permet de déterminer la valeur de l'attribut. C'est le cas pour les listes d'éléments, qu'il faut construire en saturant le paquet de règles.

```
     attribut(c_problème,P,contraintes(P,Lc),paquet(contraintes)):-
            eval_prem(problème(P)).
/* l'attribut contraintes(pb,liste_des_contraintes) est défini dans la classe
"problème", les règles qui permettent de déterminer sa valeur sont les règles du
paquet désigné par le symbole "contraintes"...*/
```

> **Pour saturer un paquet de règles**, le moteur d'inférence utilise la méthode d'évaluation des prémisses et de la conclusion décrite ci-dessus pour une règle, mais il ne se contente pas de déclencher une seule règle. Par un échec Prolog, il essaiera de déclencher toutes les règles du paquet, et pour chaque prémisse il essaiera toutes les instanciations possibles. Il construira ainsi une liste de tous les éléments recherchés dans le modèle descriptif du problème (par exemple en dénombrement toutes les contraintes sur l'objet formé).

On a vu dans le chapitre 1 que SYRCLAD peut avoir besoin d'utiliser des règles par défaut. Les règles par défaut sont aussi des règles si-alors qui sont attachées à l'attribut dont elles permettent de déterminer la valeur. Elles sont peu nombreuses et sont examinées uniquement quand aucune des autres règles attachées à l'attribut n'a pu être déclenchée.

Dans l'exemple précédent :

```
     attribut(c_1s_2i,P,conservation(P,B),[isochore,isotherme,isobare,
     isobare2,adiabatique,cons_inv1,cons_loi1],[sans]):-
            eval_prem(problème(P)).
```

La règle "sans" est une règle par défaut : si l'on n'a trouvé aucun signe de conservation d'une grandeur avec les règles précédentes, on suppose qu'il n'y en a pas.

---

[1] En dénombrement, le type de l'objet formé peut être *ensemble* ou *liste*. Par exemple, une main de cartes est un ensemble de cartes, un mot ou un nombre sont des listes de lettres ou de chiffres.

Attachement procédural ou "détachement fonctionnel" ? Soit un attribut Att défini dans une classe C, et E1 l'ensemble des règles associées à Att dans C. On pourrait tout à fait dans le formalisme de SYRCLAD définir dans une sous-classe S de C le même attribut Att, et lui associer un autre ensemble de règles E2. Les règles de l'ensemble E2 seraient plus spécifiques, étant donné que S est plus spécifique que C. Grâce au mécanisme d'héritage, lorsque pour un problème d'une sous-classe de S on doit calculer la valeur de l'attribut Att, ce sont les règles de l'ensemble E2 qui seraient appliquées. On se trouverait alors dans le cadre des représentations à objets, avec un attachement de méthodes respectant l'héritage. F. Rechenmann a souligné qu'il pouvait y avoir contradiction entre une méthode et la méthode de sa classe père. Il a donc dissocié les méthodes du graphe de classification, en préférant le "détachement fonctionnel" à l'attachement procédural. Dans les domaines auxquels nous avons appliqué SYRCLAD, nous n'avons jamais eu besoin de définir un deuxième ensemble de règles dans une sous-classe. Nous pouvons donc considérer les connaissances de reformulation comme dissociées du graphe de classification. En effet, le seul lien entre les règles et le graphe est que les règles sont attachées à un attribut, et que cet attribut n'a un sens que dans une partie du graphe. On peut donc considérer que SYRCLAD utilise le "détachement fonctionnel".

Dans SYRCLAD, il n'y a pas de choix quant à la détermination d'une valeur pour un attribut du problème. Le système ne considère pas plusieurs possibilités pour finalement choisir entre elles d'après des préférences. Déterminer la valeur d'un attribut du problème ne relève pas ici d'une déduction objective résultant uniquement des données mais plutôt d'une décision subjective du modélisateur. L'expert a décidé a priori, statiquement, des choix à faire en donnant les connaissances de reformulation. Les règles actuelles ne disent pas : "tel problème *peut* se modéliser de telle façon" (avec des possibilités concurrentes), mais : "tel problème *doit* se modéliser de telle façon" (une seule possibilité). Cela n'empêche pas qu'un autre expert puisse coder les connaissances du domaine autrement. Dans l'architecture actuelle, une fois qu'une valeur a été choisie pour un attribut, le moteur ne la remet pas en cause. SYRCLAD prend des décisions de modélisation irrévocables, il choisit une voie et s'y tient, sans retour arrière. Si l'on veut par la suite faire du diagnostic dans le cadre d'un EIAO, il faudra accepter que l'élève propose une autre modélisation que celle qu'aurait choisi le système si cette modélisation est correcte. Il faudra donc donner plus de souplesse à SYRCLAD, afin qu'il ne donne pas uniquement la modélisation que l'expert a jugé être la meilleure, mais plusieurs modélisations possibles.

Nous avons vu que l'expert fournit à SYRCLAD des connaissances de classification sous la forme d'un graphe et des connaissances de reformulation sous la forme de règles. Voyons à présent comment le noyau de SYRCLAD utilise ces connaissances afin de classer le problème et de construire un nouveau modèle du problème.

4 Opérationnalisation

Pour réaliser un résolveur SYRCLAD-X, l'expert du domaine X fournit à SYRCLAD un graphe de classification de problèmes du domaine et des connaissances de reformulation. Un utilisateur de SYRCLAD-X lui présente alors un modèle descriptif du problème à résoudre. SYRCLAD-X doit classer ce problème et en construire un nouveau modèle. Ces démarches de classement et de construction du modèle opérationnel ont lieu en même temps, le graphe de classification permettant le classement, et les règles permettant la construction du modèle opérationnel. Mais les règles sont nécessaires au classement et le graphe est nécessaire pour guider la reformulation du problème. Il y a donc un seul processus, que l'on appellera *opérationnalisation* du problème. Nous détaillerons cet unique algorithme d'abord du point de vue du classement et ensuite du point de vue de la modélisation du problème.

4.1 Classement

Pour classer un problème, SYRCLAD utilise le graphe de classification comme un guide. Il part de la racine du graphe, et affine le classement du problème petit à petit, en passant à une classe plus spécifique. Il essaie d'aller le plus loin possible vers les feuilles, au moins jusqu'à une classe opérationnelle. Il obtient ainsi la classe la plus spécifique selon laquelle les connaissances permettent de modéliser le problème.

Descendre dans une sous-classe revient à déterminer la valeur de l'attribut discriminant. Les règles permettant de déterminer sa valeur se déclenchent par nécessité. Lorsque l'on arrive dans une classe opérationnelle, on essaie quand même de descendre vers une sous-classe pour avoir un classement le plus précis possible. Il se peut que l'on ne puisse plus descendre, soit parce que l'on est arrivé à une feuille, soit parce que la valeur de l'attribut discriminant n'est pas de celles qui permettent de descendre vers les cas particuliers.

De manière synthétique, on peut décrire un algorithme :

Algorithme de descente dans le graphe de classification :
Essayer de descendre dans une sous-classe
= *trouver la valeur* de l'attribut discriminant.
Si l'on ne peut pas descendre :
Si l'on est dans une classe non-opérationnelle : la classe n'est pas assez spécifique, on possède cependant certaines informations sur le problème.
Si l'on est dans une classe opérationnelle : terminé avec succès.

Trouver la valeur d'un attribut :
lire la valeur connue ou *appliquer une règle*.

Appliquer une règle :
valeurs des prémisses connues,
ou *trouver la valeur* de la première prémisse inconnue.

4.2 Construction du modèle opérationnel

La construction du modèle opérationnel du problème a lieu pendant le classement de celui-ci. En effet, on a vu que pour classer un problème, on détermine la valeur d'attributs du problème, ces nouveaux attributs sont ceux qui définissent le modèle opérationnel. Au début du classement, le modèle opérationnel est vide, il est enrichi à chaque fois que l'on descend dans une classe plus spécifique.

Un modèle est une description d'un problème suivant un certain point de vue, ici les points de vue sont des classes du graphe de classification.

Soit M le modèle descriptif du problème, et soit D une description de M suivant une classe ; au départ D est vide, et à la fin ce sera le modèle opérationnel du problème. Nous appelons C la classe courante, G le graphe de classification. Le même algorithme qu'au paragraphe 4.1 peut s'écrire :

Algorithme de construction du modèle opérationnel :

D <- Ø, C <- racine de G

Répéter tant que c'est possible :

Trouver dans G une classe C' plus spécifique que C (une sous-classe de C) telle que M puisse se décrire suivant C'.

D' <- description de M suivant C' = D + valeur de l'attribut qui spécialise C' par rapport à C.

D <- D', C <- C'

Fin répéter

Si C est une classe opérationnelle, D est le modèle opérationnel du problème décrit par M.

Cet algorithme est illustré par la figure 4.

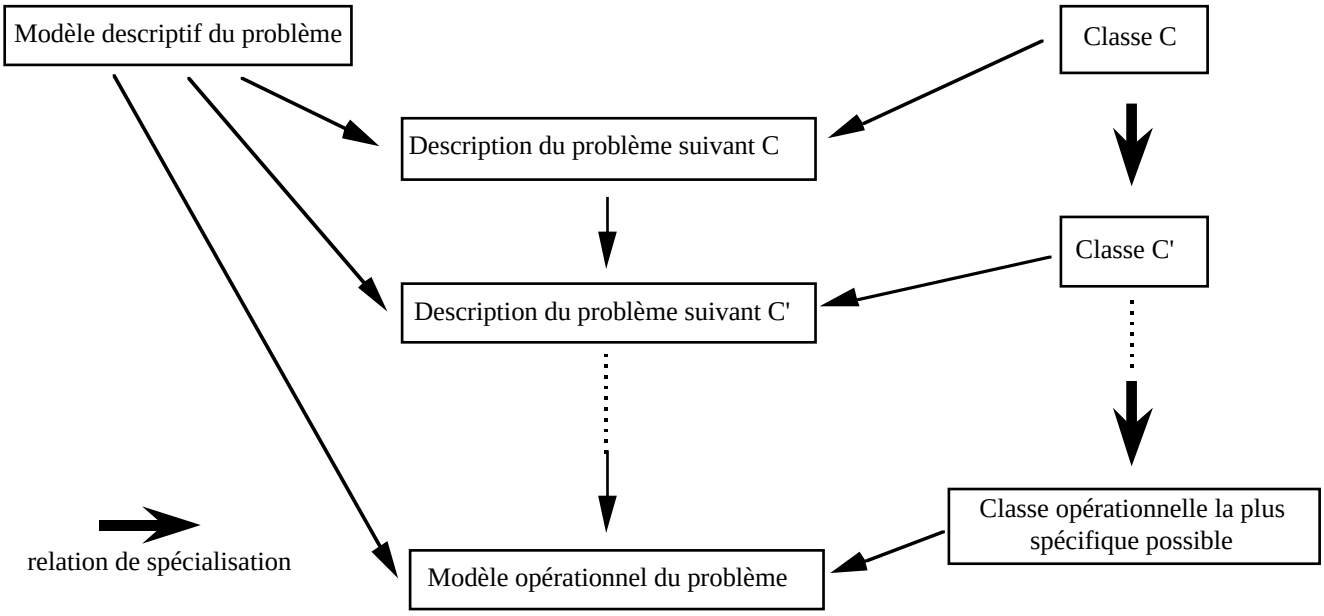


Figure 4 : Construction du modèle opérationnel

On a vu que l'algorithme qu'utilise SYRCLAD pour classer un problème est simple. Pourquoi ne pas avoir utilisé un outil déjà existant ?

Le processus d'opérationnalisation consiste à engendrer une instance d'une classe à partir d'un modèle descriptif. Pour engendrer une instance complète dans un langage de programmation objet, il faut deux sortes d'informations : la classe et les valeurs des attributs, avec la classe fixée définitivement. Dans un tel langage, il est impossible de créer une instance avant de connaître sa classe et il est impossible de changer la classe d'un objet. On ne peut donc pas adopter la stratégie qui consisterait à créer d'abord un objet d'une classe inconnue puis à compléter ensuite ses attributs.

On peut alors envisager l'inverse : déterminer les valeurs des attributs de l'objet à créer avant même de le créer, puis analyser le résultat pour en déduire la classe, puis enfin créer l'instance et la compléter. L'étape d'analyse correspond exactement à un thème des logiques terminologiques : étant donnée une description d'un objet, déterminer sa classe la plus spécifique dans un graphe de classes. On pourrait alors envisager d'utiliser un système fondé sur ces logiques. Mais pour cela, il serait nécessaire de disposer d'abord d'une description formée avec les concepts et le vocabulaire définis dans la taxonomie. La question se pose donc d'engendrer d'abord une telle description à partir du modèle descriptif. C'est une tâche de modélisation permettant de passer d'une description du problème suivant un point de vue (le modèle descriptif) à une description sous un autre point de vue (le modèle opérationnel). Le problème

consiste ici à savoir quels attributs déterminer et dans quel ordre. Ce genre de tâche se retrouve dans les systèmes de compréhension de textes écrits en langue naturelle : à partir d'une description initiale (le texte), il faut construire une autre description (dans un formalisme logique par exemple) représentant le sens du texte suivant un certain point de vue.

Deux types d'approche peuvent être envisagées : une approche "montante" (dirigée par les données, en chaînage avant) et une approche "descendante" (dirigée par les concepts cibles, en chaînage arrière). Dans l'approche montante, il est parfois difficile d'interpréter certaines données sans avoir au départ un cadre de référence, un point de vue suivant lequel les interpréter.

Il y a donc deux approches extrêmes :

- déterminer d'abord la classe définitive puis compléter les attributs

- déterminer d'abord les attributs puis en déduire la classe

SYRCLAD utilise une méthode intermédiaire entre les deux extrêmes, méthode que l'on pourrait appeler "génération descendante d'un modèle par raffinements successifs". A chaque étape, on dispose d'un modèle partiel du problème, ce modèle étant sous la forme d'une classe et d'une liste attributs-valeurs (les attributs faisant partie de ceux qui sont déclarés pour la classe). Raffiner un modèle consiste à remplacer la classe par une sous-classe en gardant les attributs déjà trouvés et en en déterminant d'autres. Au départ, on part avec la classe la plus générale (problème) et aucun attribut. L'avantage d'une telle méthode est que l'on ne remet jamais en question les attributs déjà déterminés et que l'on ne considère à chaque étape que des attributs pertinents (ceux de la classe). Mais la méthode suppose que l'on puisse déterminer les valeurs des attributs des classes les plus générales avant celles des attributs des classes particulières.

Certaines connaissances de classement, que l'on peut appeler connaissances de spécialisation, ont la forme suivante :

Si une description représente un objet que l'on peut voir sous le point de vue C (classe, concept) et si, suivant ce point de vue, l'attribut A vaut a, alors l'objet peut aussi être vu sous le point de vue plus spécialisé C1 (sous-classe de C).

Une telle connaissance suggère l'heuristique suivante, qui permet d'orienter la modélisation :

Si l'on a déjà réussi à représenter partiellement un objet sous le point de vue C, alors il peut être intéressant de déterminer la valeur de l'attribut A pour raffiner la description.

L'existence de connaissances de spécialisation et de l'heuristique associée suggèrent alors fortement d'employer une méthode descendante, aussi bien pour le classement que pour la modélisation et permettent même de fusionner ces deux processus : commencer par décrire partiellement le problème sous le point de vue le plus général (la classe racine de la hiérarchie, ici la classe "problème"), puis déterminer une sous-classe suivant laquelle on peut raffiner la description. Recommencer avec cette sous-classe, et ainsi de suite. On pourrait appeler ce procédé : modélisation hiérarchique, ou modélisation par raffinements successifs.

Nous avons vu que SYRCLAD construit pendant la phase d'opérationnalisation un modèle opérationnel du problème posé. Nous allons maintenant étudier la forme des modèles opérationnels ainsi construits.

5 Modèle opérationnel

Le résultat du classement est un couple (Cop, Dop) : une classe à laquelle appartient le problème posé, et un nouveau modèle du problème, qui est adapté à la théorie du domaine. Le modèle opérationnel du problème est en fait un nouveau problème, équivalent à celui présenté par l'énoncé, mais qui peut être considéré de manière indépendante. C'est un problème qui se suffit à lui-même, et qui contient assez d'informations pour que l'on puisse le résoudre. On n'a plus besoin du modèle descriptif (c'est-à-dire de l'énoncé) pour terminer la résolution.

Nous avons choisi de représenter le modèle opérationnel du problème dans un formalisme de type liste attributs-valeurs. La liste des attributs possibles pour un problème est définie pour chaque classe de chaque domaine. Les attributs du modèle opérationnel sont d'une part les attributs discriminants de la classification, dont les valeurs ont été déterminées pendant le classement[1], et d'autre part des attributs non discriminants, qui seront utiles pour la résolution. Ces derniers permettent de caractériser le problème en tant qu'instance. On a vu au chapitre 1 que le problème[2] m10 est de classe "répartition-liste-avec", mais l'attribut "ensemble caractéristique" est spécifique à m10 et permettra d'appliquer au problème m10 la méthode de résolution associée à sa classe.

On voit donc qu'une partie du modèle opérationnel est liée à la classe du problème, l'autre partie étant liée au problème qui en est instance.

Voici quelques prédicats discriminants présents dans des modèles descriptifs :

Dénombrements

```
type_problème(pb,répartition). /* le problème est une répartition*/
```

Problèmes additifs

```
à_trouver(pb,opérande).  /* l'inconnue à déterminer pour résoudre le
problème est l'opérande */
```

Tarot

```
nb_perdantes(pb,beaucoup).  /* le joueur attaquant pour ce problème possède
beaucoup de perdantes */
```

Thermodynamique

```
composé(pb,mélange).
/* les deux systèmes gazeux du problème sont liés par un mélange */
```

Voici des prédicats non discriminants qui peuvent aussi figurer dans des modèles opérationnels :

---

[1] Même si la présence de ces attributs est redondante lorsqu'on connaît la classe du problème, elle est nécessaire si on veut considérer le modèle opérationnel comme un problème qui se suffit à lui-même.

[2] modèle descriptif §1 et modèle opérationnel §5

Dénombrements

```
nombre_catégories(pb,4).          taille_commune_catégories(pb,8).
/* dans le problème, il y 4 catégories de taille 8 */
```

Problèmes additifs

```
donnée1(pb,3).  /* la première donnée du problème est 3 */
```

Thermodynamique

```
liste_gamma(pb,sys,i,f,température,pression)).
/* si le problème est dans la classe un_système-deux_instants-
cas_particulier_gamma, il faut savoir quel est le système, quel est
l'instant initial, quel est l'instant final, quelle est la grandeur connue
uniquement à l'instant initial et quelle est la grandeur connue aux deux
instants */
```

Les prédicats des modèles descriptifs[1] décrivent souvent un "attribut d'un objet dont parle le problème". Au contraire, les prédicats des modèles opérationnels ont toujours comme premier argument le problème lui-même. Ils décrivent en effet des "attributs du problème". Dans un modèle opérationnel, toutes les informations ont été ramenées au même niveau, celui du problème. On a réifié le problème et on en parle directement comme d'un objet et non pas indirectement par son contenu. En réifiant un problème, on peut raisonner dessus. En raisonnant sur un problème, on passe au niveau méta.

On peut remarquer que les informations contenues dans les attributs non-discriminants sont souvent numériques, ou décrivent les grandeurs utilisées dans le problème. Ceci permet d'instancier les paramètres nécessaires à l'application de la méthode au problème posé. En effet, les méthodes de résolution consistent par exemple à appliquer une formule ou un plan-type. Ces méthodes sont des procédures auxquelles on doit fournir des paramètres d'entrée pour le problème étudié.

Dans les modèles opérationnels que l'on rencontre au tarot, il n'y a que des attributs discriminants, et ce parce que les méthodes de résolutions n'ont aucun paramètre ; dans ce domaine, tous les problèmes de la même classe ont la même solution.

---

[1] par exemple :

```
est_un(a,action_tirer).  /* soit a une action de tirer des éléments */
récipient(a,s).          /* pour l'action-tirer a, le récipient dans lequel on tire est le sac s */
simultanément(a).        /* dans l'action-tirer a, les éléments sont tirés simultanément */
```

6 Connaissances de résolution

Pour choisir la méthode de résolution à employer, il faut savoir quelle méthode est adaptée pour un problème d'une classe donnée. C'est une expertise difficile, M. Chi et al ont montré dans [Chi et al 81] que les novices possèdent parfois de bons critères de classification, mais qu'ils ne savent pas les relier aux différentes méthodes de résolution possibles.

Pour réaliser un résolveur SYRCLAD-X, l'expert en X associe à chaque classe opérationnelle une méthode de résolution (M_{Cop}) :

```
employer_résolution(classe_opérationelle):- méthode de résolution
```

Une méthode peut par exemple consister à utiliser une règle :

```
employer_résolution(c_répartition_ensemble):- utiliser([rep_e]).
```

Ces méthodes de résolution sont spécifiques aux domaines, et il n'y a aucune unité de représentation imposée par l'architecture de SYRCLAD pour les méthodes de résolution. En dénombrement, on associe à chaque classe opérationnelle un plan de résolution type pour les exercices de cette classe, ainsi qu'une formule permettant de calculer la solution numérique. Il en est de même pour les problèmes additifs. Au tarot, un plan de jeu est associé à chaque classe opérationnelle. Dans ce domaine, c'est une solution plus qu'une méthode de résolution qui est associée aux classes opérationnelles.

En thermodynamique, les méthodes de résolution sont de types différents suivant les classes. Elles donnent toutes des solutions qualitatives aux problèmes. Certaines classes sont assez spécifiques pour qu'un plan de résolution type leur soit associé. Pour les autres classes, moins spécifiques, la méthode de résolution associée consiste à utiliser un moteur d'inférence sur certains modules de connaissances.

Certaines méthodes de résolution définissent des sous-problèmes à résoudre pour trouver la solution du problème posé. Une méthode de résolution donnée par l'expert peut demander à SYRCLAD de classer et résoudre ces sous-problèmes comme n'importe quel problème. En dénombrement, la classe "complémentaire" est une classe à laquelle est associée une méthode de résolution qui décompose le problème ; cette méthode consiste à classer et résoudre le problème qui compte les éléments qui *ne* vérifient *pas* les contraintes de l'énoncé (problème complémentaire), puis à classer et résoudre le problème sans contraintes (problème complet) ; la solution de l'exercice est alors la différence entre la solution du problème complet et la solution du problème complémentaire.

```
employer_résolution(c_complémentaire):-
       eval_prem(problème(P)),
/* pour résoudre le problème P */
       eval_prem(liste_complémentaire(P,N,M,Att,Val)),
/* on lit la valeur d'un attribut non-discriminant présent dans le modèle
opérationnel pour instancier P par rapport à la classe "complémentaire" */
       construire_pb_compl(P,N,M,Att,Val,Pc,Pall),
/* grâce aux valeurs récupérées, on construit le problème complémentaire Pc
et le problème complet Pall */
       ajouter_bf(complémentaire(P,Pall,Pc)),
       retirer_bf(problème(P)),ajouter_bf(problème(Pall)),classer(Pall),
       retirer_bf(problème(Pall)),ajouter_bf(problème(Pc)),classer(Pc),
       retirer_bf(problème(Pc),ajouter_bf(problème(P)),
/* on classe Pall et Pc */
       eval_prem(solution(Pall,S1)), /* soit S1 la solution de Pall*/
       eval_prem(solution(Pc,S2)),   /* et S2 celle de Pc */
       -(S1,S2,Sol), /* la solution Sol de P est S2-S1 */
       ajouter_bf(solution(P,Sol)),
       ajouter_bf(plan(P,[['On compte les résultats qui ne vérifient pas la
propriété demandée, puis on les soustrait du nombre total de résultats
possibles.']])),!.
```

Nous reviendrons dans le chapitre 9 sur les méthodes de résolution qui décomposent le problème à résoudre.

7 Produire la solution

Pour obtenir une solution au problème posé, SYRCLAD applique la méthode de résolution correspondant à la classe du problème sur le modèle opérationnel, c'est-à-dire $M_{Cop}(Dop)$. La méthode de résolution est appliquée au problème en utilisant les attributs du modèle opérationnel qui instancient le problème par rapport à Cop. De nombreux exemples de solutions seront donnés dans la partie II.

8 Conclusion

Nous avons présenté l'architecture de SYRCLAD, en décrivant la forme des bases de connaissances qu'il faut lui donner et le processus de résolution qu'il utilise. Si un expert du domaine X souhaite utiliser SYRCLAD pour réaliser un SYRCLAD-X, il faut que cet expert puisse définir :

- un langage de la logique du premier ordre qui permette de décrire les problèmes du domaine X,
- un graphe de classification des problèmes du domaine X qui permette d'associer une méthode de résolution à certaines classes de problèmes,
- des règles de reformulation permettant de modéliser un problème concret afin d'obtenir un modèle plus proche de la théorie du domaine X,
- des méthodes de résolution efficaces pour certaines classes de problèmes.

SYRCLAD est particulièrement bien adapté aux domaines où les problèmes sont concrets et où une résolution de tels problèmes nécessite une phase de modélisation de manière à se replacer dans la théorie du domaine. Cette phase de modélisation constitue la principale étape de la résolution, qui se termine par une étape plus algorithmique où une méthode de résolution adaptée au nouveau modèle du problème est appliquée. SYRCLAD facilite l'expression des connaissances dans de tels domaines en permettant d'exprimer déclarativement une classification des problèmes, et de distinguer connaissances de reformulation et connaissances de résolution. De plus, SYRCLAD sait utiliser ces connaissances pour résoudre un problème concret à partir d'un modèle descriptif de ce problème.

En utilisant l'architecture de SYRCLAD décrite dans ce chapitre, nous avons mis au point quatre applications. Ce sont ces applications qui permettent d'apprécier les résultats obtenus. La partie II de ce mémoire est consacrée à décrire ces quatre applications. Étudier le fonctionnement de SYRCLAD-X sur un domaine X devrait permettre au lecteur de mieux appréhender les modules qui le constituent.

Partie II : les quatre domaines d'application

La deuxième partie du mémoire présente les quatre applications de SYRCLAD qui ont abouti à la réalisation de quatre résolveurs de problèmes. Les quatre chapitres peuvent être lus de manière indépendante, mais toutefois après la lecture de la première partie. Le lecteur pourra ainsi faire un choix suivant ses centres d'intérêt. Le domaine des dénombrements (chap. 4) est peut-être le plus représentatif des capacités du système SYRCLAD, mais ceux des problèmes additifs (chap. 5) et de la thermodynamique (chap. 7) sont également révélateurs de ses capacités. Par contre, le domaine du tarot (chap. 6) ne permet qu'une exploitation limitée des possibilités du système SYRCLAD et le chapitre correspondant est une illustration qui ne suffit pas à donner un éclairage suffisant sur SYRCLAD.

Chapitre 4

Dénombrements : améliorer un système expert pour plus de déclarativité

Nous présentons dans ce chapitre l'application de SYRCLAD au domaine des dénombrements, c'est-à-dire SYRCLAD-dénombrements. Nous nous intéresserons tout d'abord aux spécificités de ce domaine ; nous indiquerons pourquoi il était intéressant d'appliquer SYRCLAD aux dénombrements. Nous détaillerons ensuite les connaissances données à SYRCLAD-dénombrements, et en particulier la classification. Enfin nous donnerons des exemples de résolutions d'exercices par SYRCLAD-dénombrements afin d'en évaluer les résultats.

Plan du chapitre

1 Pourquoi les dénombrements ?

1.1 Un domaine à part et difficile

Dans le programme de mathématiques de la classe de terminale scientifique, les dénombrements forment un domaine complètement à part : les exercices de dénombrement posés ne demandent pas aux élèves de connaissances préalables, il n'est pas nécessaire pour les aborder d'avoir une grande culture mathématique. Les élèves résolvent souvent de manière intuitive. Mais comme nous l'avons vu dans le chapitre 1, on s'aperçoit vite que la méthode intuitive est insuffisante : il faut justifier les raisonnements par l'application de principes rigoureux sous peine d'erreurs.

En outre, l'hypothèse fondamentale d'une recherche récente sur le raisonnement combinatoire [Fischbein et Grossman 97] est que les raisonnements intuitifs des élèves sont toujours basés sur certaines capacités structurelles. Dans cette expérimentation portant sur les permutations, arrangements avec ou sans remise, combinaisons, les auteurs constatent que, même quand elles s'expriment par des estimations, en réalité, les intuitions sont issues d'une mise en oeuvre de schémas (corrects ou erronés). En effet, un questionnaire portant sur l'estimation d'un résultat numérique pour des problèmes variés a été proposé à des individus de tous niveaux (12 ans, 14 ans, 17 ans, enseignants en formation initiale et adultes). On observe une sous-estimation du résultat pour les problèmes portant sur des permutations, une estimation correcte pour les problèmes d'arrangements avec remise, et une surestimation pour les arrangements sans remise, allant jusqu'à 50 % dans le cas des problèmes de combinaison. Ce questionnaire était suivi d'un entretien où les sujets devaient justifier leurs estimations. Les auteurs ont observé que ces "réponses intuitives" ne sont pas dûes au hasard et qu'elles sont reliées à des schémas de base (schéma de classification par exemple), les sujets étant cohérents dans leurs explications. E. Fischbein et A. Grossman insistent, par conséquent, sur la nécessité d'un point de vue didactique, d'identifier ces schémas afin de pouvoir les faire évoluer : le processus didactique doit donc prendre en compte ces capacités structurales et favoriser le développement d'organisations hiérarchiques des connaissances.

D. Grenier [Grenier 94] évoque l'intérêt d'introduire les dénombrements dans l'enseignement secondaire : « beaucoup de problèmes de combinatoire tout en étant d'un abord facile permettent de poser des questions fondamentales pour les mathématiques, en particulier celles de la modélisation et de la preuve ». Elle regrette cependant que l'enseignement des dénombrements laisse souvent à penser qu'il ne faut aucune connaissance pour résoudre de tels problèmes.

L'ENCYCLOPÆDIA UNIVERSALIS définit l'analyse combinatoire comme « l'ensemble des techniques qui servent, en mathématiques, à compter (ou dénombrer) certaines structures finies, … Le plus souvent on est conduit à chercher une correspondance biunivoque entre ces structures et les ensembles finis. … Jusqu'ici, il n'existe pas de théorie pour construire ces correspondances. Tout est question d'ingéniosité et de patience. ». L'activité de dénombrement est effectivement liée à la

modélisation. Si l'on ne possède pas des trésors d'ingéniosité et de patience, il peut être intéressant d'avoir une méthode permettant de guider cette modélisation.

A. Antibi a consacré un chapitre de sa thèse d'état [Antibi 88] aux problèmes de dénombrements. Il a proposé un exercice de dénombrement à 55 enseignants, du supérieur et du secondaire, et à 37 étudiants de Deug, de Math Sup, d'une école d'ingénieur et d'une préparation au CAPES. Les résultats montrent que le nombre de réponses justes est faible : 16% pour les enseignants et 11% pour les étudiants. Il apparaît également que personne ne s'est déclaré sûr de sa réponse. Les enseignants ont tous déclaré qu'il est souvent très difficile dans ce domaine de faire comprendre à un élève pourquoi il s'est trompé. Les solutions aux exercices de dénombrement sont souvent courtes, et il apparaît qu'il serait très long et très compliqué de rédiger une démonstration détaillée de telles solutions ; ces démonstrations utiliseraient souvent des notions qui ne sont pas connues par les élèves. Cela explique peut-être pourquoi dans ce domaine, on est rarement sûr de sa réponse et pourquoi les enseignants ont du mal à expliquer aux élèves leurs erreurs. Pour expliquer la difficulté des exercices de dénombrement, A. Antibi propose plusieurs raisons : on ne donne pas, contrairement à d'autres domaines mathématiques, la marche à suivre et les étapes intermédiaires ; on indique très rarement la réponse, ce qui permettrait de vérifier sa solution ; enfin, la spécificité de ces problèmes souvent signalée par les enseignants et les étudiants : il est difficile de trouver un modèle mathématique permettant de traiter le problème.

J. Briand [Briand 93] a étudié des copies de baccalauréat contenant la résolution d'un exercice de dénombrement. Cette étude montre que les étudiants n'arrivent pas à passer de la compréhension de la situation au dénombrement de la collection. Ce passage engendre une opération fausse pour plus de la moitié d'entre eux. Il constate que les étudiants ne disposent pas d'instruments permettant de passer d'une conception des objets à dénombrer à une autre qui faciliterait l'énumération. Nous proposons une méthode de résolution des exercices de dénombrement qui utilise justement un changement de représentation du problème, cette nouvelle représentation permettant un dénombrement plus aisé.

1.2 Pourquoi appliquer SYRCLAD aux dénombrements ?

Une des principales difficultés rencontrées par les élèves en dénombrement réside dans le fait que les exercices sont posés de manière concrète, alors que le cours utilise des notions abstraites (fonctions, ensembles). Un travail préalable de modélisation est donc nécessaire, et c'est toute la difficulté de la résolution. Il faut reformuler le problème sous une forme telle que l'on puisse appliquer les théorèmes du cours. Un système capable de résoudre les problèmes de dénombrements doit donc posséder des capacités de modélisation importantes pour engendrer un modèle à partir d'un texte en langage naturel, puis pour résoudre le problème. Un tutoriel doit de plus être capable d'expliquer ses solutions. Nous pensons qu'utiliser SYRCLAD permet d'expliciter cette phase de modélisation des problèmes.

Dans les problèmes de dénombrement, il s'agit de compter le nombre des solutions sans les énumérer. L'énumération n'est d'ailleurs pas toujours possible : il suffit de penser à des problèmes de plaques d'immatriculation ! Les ensembles sont définis en compréhension, on utilise donc, plutôt qu'une méthode par énumération, les propriétés de structure de l'ensemble des solutions. La méthode

de résolution se base sur des connaissances analogues à celles qu'un expert humain emploierait pour la résolution.

Le groupe *Combien?*[1] avait élaboré des classes basées sur la théorie du domaine et sur une longue expérience d'enseignement des dénombrements en terminale scientifique. Le système expert réalisé pendant mon DEA [Guin 94, Guin et al 95] utilisait une base de règles divisée en paquets. Chaque paquet permettait de déterminer la valeur d'un des attributs distingués par ces classes.

L'élaboration de l'architecture de SYRCLAD et son application aux dénombrements a permis une réification des classes et des relations entre elles (sous-classe-de), ainsi que des relations entre les classes et les méthodes de résolution. L'architecture de SYRCLAD a permis de distinguer différentes sortes de connaissances, et le système ainsi obtenu permet une représentation plus déclarative des connaissances nécessaires à la résolution des exercices de dénombrement ; la résolution est plus claire et plus générale, et les résultats sont plus satisfaisants.

2 Les bases de connaissances de SYRCLAD-dénombrements

2.1 Les exercices rencontrés

Nous nous intéressons aux exercices de dénombrement de niveau terminale scientifique. Les exercices traitent de situations concrètes variées. Les objets qui apparaissent dans les énoncés constituent souvent des traits de surface de l'exercice. Comme nous l'avons vu au chapitre 3, le modèle descriptif est composé de prédicats définissant des objets ; ces objets sont liés par des relations afin de former une situation, et il y a des propriétés et des contraintes sur ces objets. On trouvera en annexe une liste des prédicats définissant le langage pour exprimer les modèles descriptifs des exercices de dénombrement. Vingt-six d'entre eux définissent des objets, 22 construisent des relations et 21 précisent des propriétés.

Une particularité des exercices de dénombrement est que des énoncés très ressemblants peuvent avoir des méthodes de résolutions très éloignées. De même, des exercices très différents a priori s'avéreront faire partie de la même classe. Pour que le lecteur ait un aperçu des exercices résolus par SYRCLAD-dénombrements, nous donnons l'énoncé en langage naturel de certains de ces exercices. Ils ont pour l'occasion été classés en fonction de leurs traits de surface. Une telle classification n'est pas adaptée à la résolution.

Chiffres

On dispose de six chiffres : 2, 4, 5, 6, 8, 9. Combien de nombres distincts peut-on former en les utilisant tous une fois et une seule fois ?

[1] H. Giroire et G. Tisseau de Paris6, F. Le Calvez et M. Urtasun de Paris 5 et J. Duma du Lycée Jacquard.

Un enfant forme des nombres sans répétitions avec des chiffres de 1 à 7. Combien peut-il former de nombres de sept chiffres dont le chiffre des unités est 5 ?

Combien de nombres de cinq chiffres peut-on constituer avec les chiffres pairs ?

Choix de personnes dans des groupes

Dans une classe de 36 élèves, combien de façons a-t-on de choisir deux délégués ?

Dans une classe de 36 élèves, combien de façons a-t-on de choisir un responsable de cahier de texte et son suppléant ?

Jetons

Un sac contient 9 jetons numérotés de 1 à 9. On tire 3 jetons successivement, en remettant à chaque fois le jeton tiré dans le sac avant de tirer le suivant. On écrit côte à côte chacun des trois chiffres tirés, dans l'ordre du tirage, formant ainsi un nombre de 3 chiffres. Combien peut-on obtenir de résultats différents ?

Un sac contient 9 jetons numérotés de 1 à 9. On procède au tirage de 3 jetons simultanément. Combien peut-on obtenir de résultats différents ?

Dans un sac, se trouvent 5 jetons verts numérotés de 1 à 5 et 4 jetons rouges numérotés de 1 à 4. On tire au hasard et sans remise 3 jetons dans le sac. Combien y a-t-il de tirages contenant au plus 2 jetons verts ? Combien y a-t-il de tirages contenant exactement 2 jetons verts et 2 jetons numéroté 2 ?

Un sac contient 19 jetons numérotés de 1 à 19. Combien peut-on former d'ensembles de 4 jetons dont un jeton au moins porte un numéro pair ?

Mots

On considère l'alphabet réduit constitué des 5 lettres e, a, s, n, i . Combien de mots de 3 lettres peut-on composer avec cet alphabet sachant qu'un mot peut contenir plusieurs fois la même lettre ? (on ne tient pas compte du sens des mots ainsi formés).

On appelle "mot" toute permutation de lettres données. Avec les lettres du mot BUNGALOW, combien peut-on former de mots de 8 lettres commençant par une consonne et finissant par une voyelle ?

Combien y a-t-il de mots de 6 lettres commençant par une voyelle et dont le reste du mot est composé avec deux lettres, l'une présente 3 fois, et l'autre présente 2 fois ?

Cartes

Une "main" est un ensemble de 8 cartes d'un jeu de 32 cartes. Combien existe-t-il de mains contenant exactement 3 piques, 2 carreaux et 3 trèfles ? Combien existe-t-il de mains contenant au moins 3 valets ?

On dispose d'un jeu de 32 cartes. On en tire 5 simultanément. Quel est le nombre de tirages de 5 cartes qui comportent exactement 2 valets et 2 cœurs ?

On considère les mains de 5 cartes que l'on peut constituer avec un jeu de 32 cartes à jouer. Déterminer le nombre de mains de 5 cartes qui contiennent un full, c'est-à-dire 3 cartes de même hauteur, et 2 autres cartes de même hauteur.

Énoncé générique

On peut formuler un énoncé générique pour les exercices de dénombrement que nous considérons :

> On dispose d'un ensemble (alphabet, chiffres, jeu de cartes) où l'on choisit des éléments (lettre, carte, jeton) pour former un objet (mot, nombre, main de cartes). Il existe des *contraintes* sur cet objet (lettres toutes différentes, un seul chiffre pair, deux piques et deux dames). Il s'agit de dénombrer le nombre d'objets que l'on peut former de cette manière et qui respectent les contraintes.

2.2 Les connaissances de classification

Le domaine des dénombrement est intéressant et pose des difficultés d'enseignement. Il peut exister plusieurs classifications de problèmes de dénombrement, suivant le point de vue. Notre point de vue est celui de la résolution par plan des exercices les plus courants posés en terminale scientifique. Parmi d'autres points de vue, on peut citer une classification établie par J.-G. Dubois [Dubois 84] qui distingue plusieurs types de rangements simples d'objets dans des cases. Il établit une correspondance entre ces classes de rangements simples et des classes de sélection d'objets parmi un ensemble d'objets. Il étend cette correspondance à des classes de séparations en tas et des classes de partages d'entiers. Il associe des formules aux classes de rangements introduites et expose des méthodes de déduction générales. Nous nous intéressons uniquement aux problèmes de sélection, qui sont les plus fréquents. Nous étudions en particulier des exercices où il existe des contraintes sur les objets choisis, ces contraintes peuvent être compliquées.

Au sein du groupe *Combien?* , G. Tisseau a déterminé les informations nécessaires pour produire le plan de résolution, c'est-à-dire les caractéristiques définissant la classe d'un exercice. La classification obtenue est le résultat de son expérience de quinze années d'enseignement en terminale. Cette classification a été établie en analysant conjointement les différentes formes de solutions rencontrées (proposées par les élèves et les manuels) et les types de contraintes présents dans les énoncés. Des classes spécifiques ont ainsi été dégagées, et elles ont ensuite été généralisées et

organisées en hiérarchie. Il ne semble pas qu'une telle classification ait déjà été établie, et les noms des classes ont été inventés pour l'occasion. On s'est limité aux classes auxquelles on peut associer des formules directes de résolution. Ces formules ne sont pas toujours au programme de terminale, mais les élèves peuvent les retrouver sur des cas particuliers.

Nous allons étudier trois attributs qui permettent de classer un problème de dénombrement :

- le *type du problème*, qui dépend de la forme des contraintes qui portent sur l'objet formé ;
- le *type de l'objet formé*, suivant que l'objet formé est une liste ou un ensemble ;
- la *remise* , suivant que l'on peut ou non utiliser plusieurs fois le même élément pour former l'objet ;

a) Le type du problème

Il existe trois types de base : la répartition, la liste avec conditions de position et le spectre. Nous étudierons ensuite trois types qui débouchent sur des résolutions composées : la disjonction, le complémentaire et le produit.

Répartition

Voici un exemple d'exercice de type répartition : « Combien y a-t-il de mots de cinq lettres contenant (exactement) un A et (exactement) deux consonnes ? ». Il peut se réécrire : « On tire 5 éléments (les lettres) dans un ensemble E de taille 26 (l'alphabet), dans lequel on distingue la *catégorie* des A et celle des consonnes. Combien y a-t-il de résultats contenant exactement un élément dans la catégorie des A et deux dans la catégorie des consonnes ? ».

L'exercice type correspondant à la répartition est : « On tire p éléments dans un ensemble E de taille e divisé en m *catégories* disjointes, combien y a-t-il de résultats contenant exactement x_1 éléments de la catégorie 1, ..., x_n éléments de la catégorie n ? ».

Un nouvel attribut est défini pour les problèmes de type répartition : l'ensemble caractéristique. Il permet de préciser les catégories dans lesquelles on répartit les éléments. Il est défini par un ensemble de triplets, un triplet (taille de la catégorie, nombre d'éléments à choisir dans cette catégorie, description de la catégorie) pour chaque catégorie. Dans l'exemple ci-dessus l'ensemble caractéristique est :

((1, 1, a) ; (20, 2, consonne)).

Nous verrons au paragraphe 3 la résolution par SYRCLAD-dénombrements d'un problème de type répartition. Il s'agit de l'exercice j12.

Les combinaisons, arrangements, permutations, produits cartésiens sont des cas particuliers de répartition. Nous verrons par la suite les attributs qui permettent de discriminer ces cas particuliers.

Liste avec des conditions de position

L'exercice type s'écrit : « On tire p éléments successivement dans un ensemble E à e éléments, partitionné en m catégories disjointes d'effectifs c_1, ..., c_m. On forme une liste. Combien y a-t-il de listes telles que i_1 éléments à des positions données appartiennent à la catégorie 1, ..., i_n éléments à des positions données appartiennent à la catégorie n (pas de restriction pour les autres) ? ».

Un nouvel attribut est défini pour les problèmes de type liste avec des conditions de position : l'ensemble caractéristique. Ce n'est pas le même que pour les problèmes de type répartition. Il permet de préciser les catégories dans lesquelles on choisit les éléments à positionner. Il est défini par un ensemble de triplets, un triplet (taille de la catégorie, liste des positions à remplir avec des éléments de cette catégorie, description de la catégorie) pour chaque catégorie.

L'exercice suivant : « Combien y a-t-il de mots de huit lettres commençant par une consonne et terminant par deux voyelles ? » est de type liste avec des conditions de position. L'ensemble caractéristique est ((20, [1], consonne) ; (6, [7,8], voyelle)).

Nous verrons au paragraphe 3 la résolution par SYRCLAD-dénombrements d'un problème de type liste avec des conditions de position. Il s'agit de l'exercice m14.

Spectre

L'exercice type s'écrit : « On tire p éléments dans un ensemble E divisé en m catégories disjointes de même effectif c. On s'intéresse à la partition de cet ensemble en classes d'éléments de même catégorie. Combien y a-t-il de résultats contenant k_1 classes de i_1 éléments, ..., k_n classes de i_n éléments ? ($k_1i_1+...+k_ni_n=p$) ».

Quatre nouveaux attributs sont définis pour un problème de type spectre : l'attribut définissant les catégories, le nombre de catégories (m), la taille commune à ces catégories (c), et l'ensemble des couples (nombre de classes, taille des classes). Cet ensemble est appelé ensemble caractéristique de l'exercice.

L'exercice suivant : « Dans un jeu de 32 cartes, combien de mains de cinq cartes peut-on former qui contiennent deux fois deux cartes de même couleur et une carte d'une autre couleur ? Par exemple : dame de cœur, 8 de cœur, 10 de trèfle, valet de trèfle, roi de carreau. » est de type spectre. L'attribut définissant les catégories est la couleur, il y a 4 catégories de taille 8, et l'ensemble caractéristique est :

((2, 2) ; (1, 1)).

Nous verrons au paragraphe 3 la résolution de ce problème par SYRCLAD-dénombrements. Il s'agit de l'exercice tc56.

Certains problèmes s'expriment comme des combinaisons de problèmes plus simples, leur classe est alors une classe composée. On reconnaît qu'un problème est composé grâce aux contraintes qui

portent sur l'objet formé. Il existe trois sortes de classes composées : le produit, le complémentaire et la disjonction, chacune ayant ses propres attributs. Il s'agit en fait de trois autres valeurs possibles pour le type d'un problème.

Produit

Il s'agit d'un problème que l'on peut considérer comme un produit cartésien de sous-problèmes indépendants. Les contraintes portent sur des parties disjointes de l'objet formé. On considère alors des sous-problèmes consistant à former une partie de l'objet. Il faut ensuite faire le produit des solutions des sous-problèmes considérés.

Un nouvel attribut des problèmes de type produit est la liste des problèmes dont on fait le produit. Chaque sous-problème peut appartenir à une classe de base ou une classe composée.

Voici un exemple de problème produit : « Combien y a-t-il de mots de 6 lettres commençant par une voyelle et dont le reste du mot est composé avec deux lettres, l'une présente 3 fois et l'autre présente 2 fois ? ».

On considère d'une part le problème de choisir une voyelle pour la première lettre, problème de classe produit cartésien, et d'autre part le problème de former un mot de 5 lettres composé avec deux lettres, l'une présente 3 fois et l'autre présente 2 fois, problème de classe spectre.

Nous verrons au paragraphe 3 la résolution de ce problème par SYRCLAD-dénombrements. Il s'agit de l'exercice m13.

Disjonction

Un problème de type disjonction est un problème où la contrainte définit des cas disjoints. Un nouvel attribut des problèmes de cette classe est la liste des problèmes disjoints. La classe de chacun de ces problèmes peut être une classe de base ou une classe composée. La solution du problème est la somme des solutions des différents sous-problèmes. Il existe deux types de disjonction, la disjonction linéaire et la disjonction "dame de pique".

Disjonction linéaire

Voici un exemple de problème de classe disjonction linéaire : "Dans un sac se trouvent 5 jetons verts et 4 jetons rouges. On tire au hasard et sans remise 5 jetons dans le sac. Combien y a-t-il de tirages contenant au moins 3 jetons verts ? ".

On considère trois sous-problèmes, celui où il faut exactement 3 jetons verts (et 2 rouges), celui où il faut exactement 4 jetons verts (et 1 rouge), et celui où il faut exactement 5 jetons verts. En faisant la somme des solutions de ces trois problèmes, on obtient la solution du problème composé.

Disjonction "dame de pique"

Prenons un exemple de classe disjonction "dame de pique" : « Dans un sac se trouvent 5 jetons verts numérotés de 1 à 5 et 4 jetons rouges numérotés de 1 à 4. On tire au hasard et sans remise 5

jetons dans le sac. Combien y a-t-il de tirages contenant exactement 2 jetons verts et exactement 1 jeton numéroté 2 ? ».

On s'aperçoit vite que le jeton vert numéroté 2 va poser problème, c'est lui qui est la "dame de pique". Il s'agit en effet de problèmes où deux attributs (couleur et numéro) définissent des catégories (vert et 2) dont l'intersection est réduite à un élément.

Pour résoudre ce type de problèmes, il faut considérer deux problèmes disjoints : celui où on prend le jeton 2 vert (il faut encore choisir un vert différent du 2), et celui où on ne prend pas le jeton 2 vert (il faut choisir deux jetons verts non 2 et le jeton 2 rouge).

Nous verrons au paragraphe 3 la résolution par SYRCLAD-dénombrements d'un problème de type disjonction "dame de pique". Il s'agit de l'exercice tc7bis.

Complémentaire

On considère le complémentaire du problème posé, en niant la contrainte portant sur l'objet formé. On considère d'autre part le problème complet (sans contraintes). La solution du problème posé est la différence entre la solution du problème complet et la solution du problème complémentaire. Les nouveaux attributs des problèmes appartenant à cette classe sont donc le problème complet et le problème complémentaire. La classe du problème complet est une classe de base, celle du problème complémentaire est une classe de base ou une classe composée.

Voici un exemple de classe complémentaire : « Dans un sac, se trouvent 5 jetons verts et 4 jetons rouges. On tire au hasard et sans remise 3 jetons dans le sac. Combien y a-t-il de tirages contenant au plus 2 jetons verts ? ».

L'exercice se résout en comptant le nombre de tirages contenant exactement 3 jetons verts (c'est là le complémentaire du problème posé), et en retranchant ce nombre à celui comptant tous les tirages possibles.

On remarque que cet exercice aurait aussi pu se résoudre (mais de manière plus "coûteuse") en faisant une disjonction : on considère les trois exercices où on veut exactement 0, exactement 1, et exactement 2 jetons verts. Un exercice peut être classé de différentes manières, SYRCLAD choisit la classe la plus spécialisée, et dans le cas présent la classe la plus "économique" en terme de calculs au moment de la résolution.

Nous verrons au paragraphe 3 la résolution par SYRCLAD-dénombrements d'un problème de type complémentaire. Il s'agit de l'exercice j18.

b) Le type de l'objet formé

Le type de l'objet formé peut être soit liste soit ensemble. Cela dépend si l'ordre des éléments compte ou non. Si l'ordre compte, il y a plus d'objets à dénombrer. C'est cet ordre qui différencie les combinaisons des arrangements par exemple.

Un mot, un nombre sont de type liste. Une main de carte est de type ensemble.

c) La remise

Un choix d'éléments a lieu avec remise (on remet l'élément dans le récipient après l'avoir tiré) si l'on peut choisir plusieurs fois le même élément. Il est sans remise sinon. Avec remise signifie donc qu'il peut y avoir des répétitions dans l'objet formé. S'il ne peut pas y avoir de répétitions, c'est sans remise.

L'attribut remise n'est considéré que lorsque l'objet formé est de type liste. En effet, lorsque l'objet formé est un ensemble, on choisit les éléments simultanément, la remise n'a pas de sens.

Ces trois attributs (type du problème, type de l'objet formé, remise) et leurs valeurs possibles définissent des classes de problèmes. Considérons l'exercice du chapitre 1 : « Combien peut-on former de mots de cinq lettres contenant exactement deux voyelles et deux "b" ? ».

Le type du problème est une répartition entre la catégorie des voyelles, celle des b et les autres lettres. Le type de l'objet formé est une liste, puisque l'ordre compte dans un mot. Le choix des éléments se fait avec remise, puisque l'on peut utiliser plusieurs fois la même lettre.

Ce problème appartient donc à la classe répartition-liste-avec.

REPARTITION LISTE AVEC	opérationnelle
ensemble où l'on choisit, type du problème, objet formé, liste des contraintes, ensemble caractéristique	
type du problème = répartition *type de l'objet formé = liste* *avec remise*	
plan3 **formule3**	

Figure 1 : la classe répartition-liste-avec

Cette classe est opérationnelle, un plan-type et une formule permettent d'obtenir le plan de résolution et la solution numérique. On utilise pour cela la valeur de l'ensemble caractéristique de l'exercice : ((20, 2) ; (1, 2)).

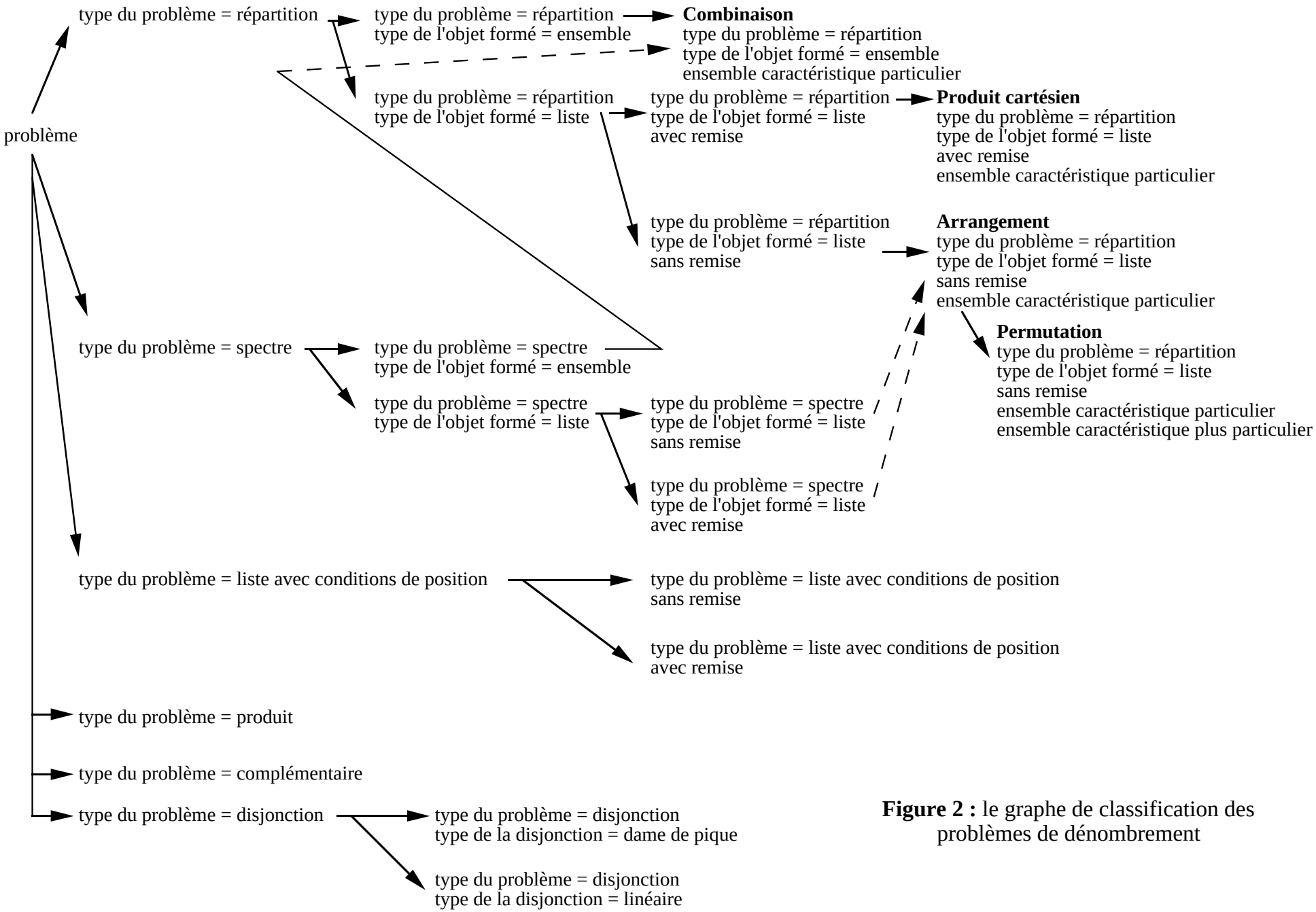


Figure 2 : le graphe de classification des problèmes de dénombrement

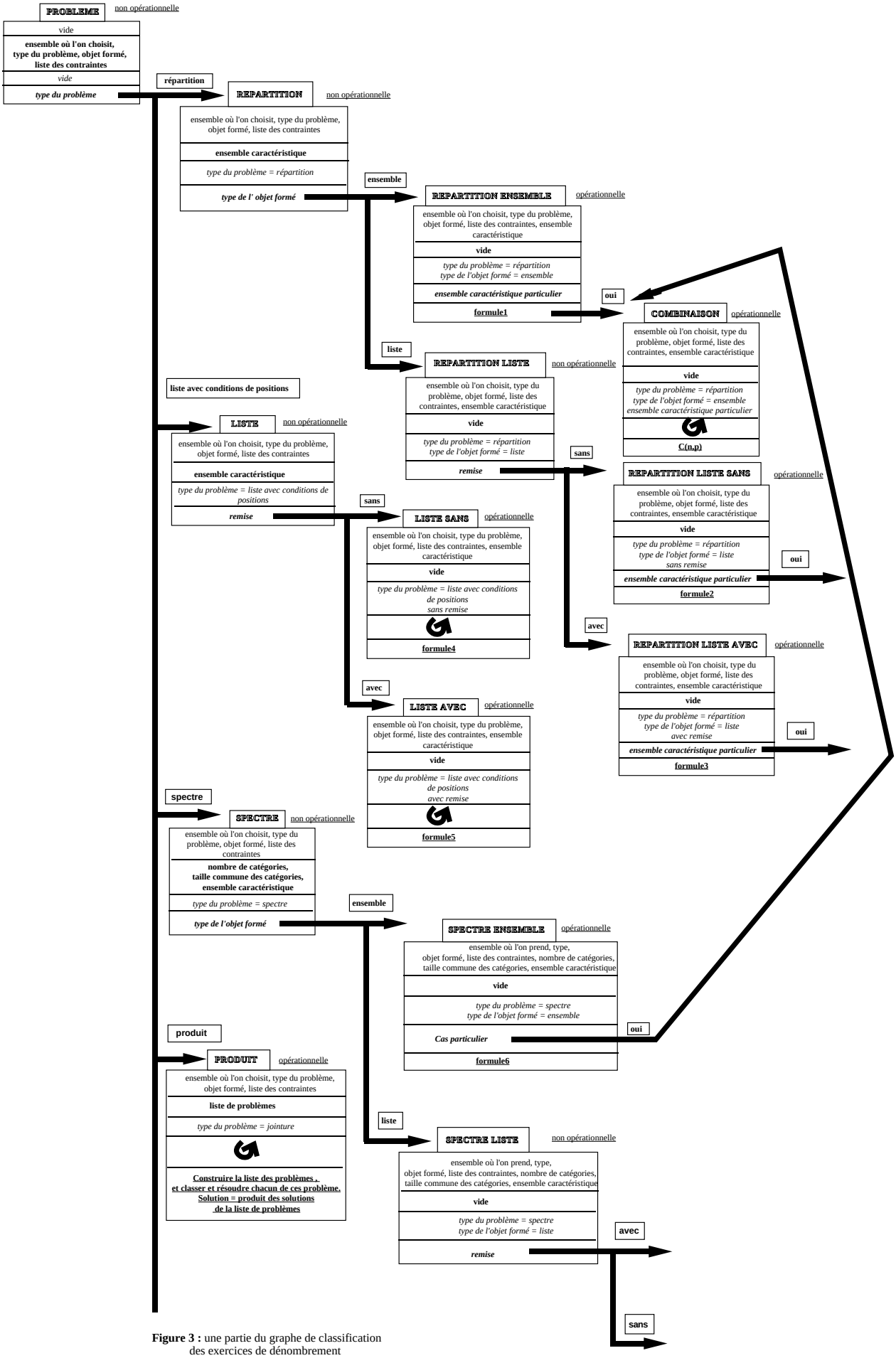


Figure 3 : une partie du graphe de classification des exercices de dénombrement

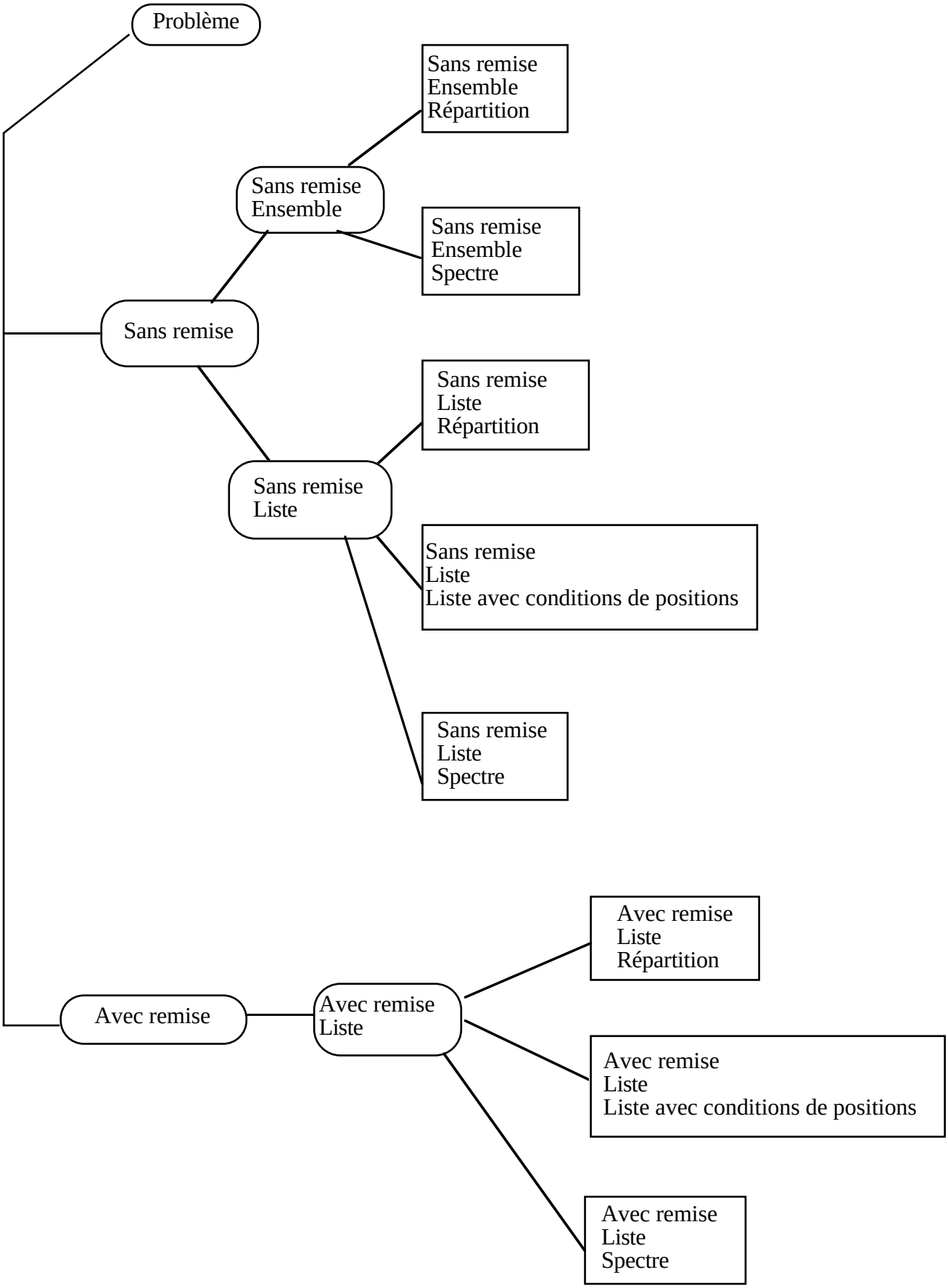


Figure 4 : un autre graphe de classification possible

Nous avons organisé les classes définies par les trois attributs (type du problème, type de l'objet formé et remise) en une hiérarchie. Nous présentons en figure 2 la classification complète des problèmes de dénombrement ainsi établie, en donnant pour chaque classe uniquement les critères qui la définissent. Nous donnons également en figure 3 une partie de ce graphe de classification de manière plus détaillée.

Dans ce graphe de classification, nous avons choisi de discriminer un problème d'abord sur le type du problème, ensuite sur le type de l'objet formé et enfin sur la remise. On pourrait construire un graphe qui examine d'abord la remise, puis le type de l'objet formé et enfin le type du problème (cf. figure 4). En tenant compte de ces trois attributs, il y a six graphes de classification possibles. Nous avons choisi le plus logique. En effet, il est illogique de déterminer la remise avant de déterminer le type de l'objet formé : il ne peut pas y avoir de remise quand l'objet formé est un ensemble. Si l'on détermine le type de l'objet formé avant le type du problème, c'est redondant dans le cas d'une liste avec conditions de position.

Autres attributs discriminants

D'autres attributs discriminants apparaissent dans le graphe de classification (Figure 2).

On a vu qu'il y a deux valeurs possibles pour le type de la disjonction.

Dans la branche type du problème = répartition, un attribut discriminant apparaît : ensemble caractéristique particulier. On dit que l'ensemble caractéristique est particulier lorsqu'il est réduit à un triplet (taille catégorie, nombre d'éléments à choisir dans la catégorie, description de la catégorie), avec le nombre d'éléments à choisir égal à la taille de l'objet formé. Cela signifie que l'on choisit tous les éléments dans la même catégorie, ce qui correspond aux cas particuliers de la combinaison, de l'arrangement ou du produit cartésien (selon la valeur du type de l'objet formé et de la remise).

Par exemple : « Combien y a-t-il de mots de cinq lettres contenant exactement 5 voyelles ? ». C'est un exercice de classe répartition-liste-avec, l'ensemble caractéristique est ((6, 5, voyelle)) et l'objet formé est de taille 5. On est donc dans le cas d'un produit cartésien : choisir 5 éléments dans un ensemble de 6, successivement et avec répétitions. La solution est 6^5.

L'attribut qui spécialise la classe permutation par rapport à la classe arrangement est : ensemble caractéristique plus particulier. Dans la classe arrangement, on sait que l'ensemble caractéristique est de la forme ((T, N, desc)), où N est la taille de l'objet formé. L'ensemble caractéristique est *plus* particulier s'il est de la forme : ((N, N, desc)).

Exemple : « Combien y a-t-il de mots de huit lettres formés avec les lettres du mot "bungalow" ? Chaque lettre ne peut être utilisée qu'une fois. ». Ce problème est de classe répartition-liste-sans_remise. Comme il n'y a aucune contrainte sur l'objet formé, l'ensemble caractéristique est : ((taille de l'ensemble où l'on choisit, taille de l'objet formé, _)). L'attribut ensemble-caractéristique-particulier est vrai. Le problème est de classe arrangement. Or cet ensemble caractéristique est ((5, 5, _)), en effet ici la taille de l'objet formé est égale à la taille de l'ensemble où l'on choisit.

L'ensemble caractéristique est *plus* particulier. Le problème est bien une permutation. La solution est 8! .

Sur la figure 2, on remarque trois flèches en pointillé spécialisant des classes de la branche type du problème = spectre vers les classes combinaison et arrangement. Il s'agit aussi d'un cas particulier. On sait que l'on définit pour les problèmes de la classe spectre le nombre de catégories, la taille commune à ces catégories, et l'ensemble caractéristique des couples (nombre de classes, taille des classes). On est dans un cas particulier si les catégories sont de taille 1, et si l'ensemble caractéristique est : ((taille de l'objet formé, 1)).

Exemple : « Un sac contient 9 jetons numérotés de 1 à 9. On tire 3 jetons successivement, en remettant à chaque fois le jeton tiré dans le sac avant de tirer le suivant. On écrit côte à côte chacun des trois chiffres tirés, dans l'ordre du tirage, formant ainsi un nombre de 3 chiffres. Combien peut-on obtenir de nombres formés avec des chiffres tout différents ? ». SYRCLAD-dénombrements analyse la contrainte "valeurs différentes pour numéro" et en déduit que le type du problème est spectre[1] . Les catégories sont définies par l'attribut numéro et donc de taille 1. L'ensemble caractéristique est ((3, 1)). C'est un cas particulier, le problème est donc en fait un arrangement. La solution est A_9^3.

> **Le graphe de classification que nous avons présenté ici peut être augmenté** afin que SYRCLAD-dénombrements puisse résoudre d'autres types de problèmes. Par exemple, le résolveur d'exercices de dénombrement réalisé pendant mon DEA résolvait des problèmes de dés et de roues de loteries. Ces problèmes n'appartiennent pas à la classification que nous avons présentée car il ne s'agit pas dans ces problèmes de choisir des éléments dans un ensemble. Dans le résolveur de DEA, ils étaient donc *normalisés* par un paquet de règles avant d'être traités comme les autres problèmes. En effet, tirer N dés est équivalent à former un nombre de N chiffres pris successivement et avec remise parmi les chiffres de 1 à 6. De même, faire tourner N roues divisées en K secteurs revient à former un nombre de N chiffres pris successivement et avec remise parmi les chiffres de 1 à K.
>
> Pour que SYRCLAD-dénombrements traite ces problèmes, il faut ajouter un frère à la classe "problème" qui ne sera alors plus la racine du graphe. L'attribut à_normaliser(problème, booléen) discriminera les problèmes à normaliser des problèmes qu'il n'est pas nécessaire de normaliser. Pour déterminer la valeur de cet attribut, il faut ajouter des règles : les problèmes de dés et de roues de loteries sont à normaliser ; par défaut, les autres problèmes ne le sont pas. Il faut également ajouter une méthode de résolution pour les problèmes de la nouvelle classe. Cette méthode consistera à normaliser le problème en un problème de formation de nombres, puis à demander à SYRCLAD-dénombrements de classer et résoudre ce nouveau problème ; c'est une méthode composée.
>
> Nous avons vu ici que l'on peut augmenter un graphe de classification, en ajoutant également les connaissances de reformulation et les connaissances de résolution associées. Grâce à la déclarativité de l'architecture SYRCLAD, ces ajouts se font aisément. Par souci de clarté, nous ne parlerons dans la suite que du graphe de classification présenté sur la figure 2.

Nous avons vu plusieurs attributs discriminants qui apparaissent dans le graphe de classification. Nous avons également donné les diverses valeurs que ces attributs peuvent prendre. Pour déterminer ces valeurs, SYRCLAD-dénombrements a besoin de connaissances de reformulation.

---

[1] Le lecteur peut penser que pour cet exercice, la contrainte "valeurs différentes pour numéro" signifie qu'on choisit en fait sans remise, et que le problème est un arrangement. Mais SYRCLAD a raison d'associer le type spectre à une contrainte "valeurs différentes pour". En effet, s'il y avait des jetons verts et des jetons rouges numérotés, la contrainte induirait un "vrai" spectre ; les catégories définies par l'attribut numéro seraient de taille 2.

2.3 Les connaissances de reformulation

Nous allons à présent évoquer le contenu des règles qui concluent sur la valeur des attributs discriminants que nous avons décrits au paragraphe précédent.

L'attribut dont la valeur est la plus difficile à déterminer est le type du problème. Seize règles concluent sur cet attribut, mais on a vu au chapitre 1 que SYRCLAD-dénombrements peut avoir à déclencher 21 règles avant de conclure sur le type du problème. Cela est dû au fait que dans ces règles qui concluent sur le type du problème, les prémisses font intervenir d'autres attributs du problème, dont il faut également déterminer la valeur. Nous allons maintenant détailler quels sont ces attributs.

Pour déterminer le type du problème, il faut analyser les contraintes sur les objets à dénombrer. Il faut donc dresser la liste de ces contraintes. Étudions sur quoi portent les contraintes dans ces exercices :

Exercice m11 : « Combien peut-on former de mots de 5 lettres commençant par une voyelle ? ».

```
problème(m11).
action(m11,f).
à_dénombrer(m11,d).
est_un(f,action_former).
type_des_objets_formés(f,mot).
au_moyen_de(f,a).
tout_résultat(f,r).
est_un(r,mot).
taille(r,5).
position(r,1,type_lettre,voyelle).
est_un(a,ensemble).
taille(a,26).
tout_élément(a,xa).
est_un(xa,lettre).
est_un(d,ensemble).
ens_des_résultats(d,f).
```

Exercice tc1 : « Une "main" est un ensemble de 8 cartes d'un jeu de 32 cartes. Combien existe-t-il de mains contenant exactement deux piques ? ».

```
problème(tc1).
on_dispose_de(tc1,j).
à_dénombrer(tc1,d).
est_un(d,ensemble).
tout_élément(d,xd).
est_un(xd,ensemble).
taille(xd,8).
inclus_dans(xd,j).
effectif_attribut(xd,exactement,2,
couleur,pique).
est_un(j,jeu_de_32_cartes).
```

Dans l'exercice m11, la contrainte (position(r,1,type_lettre,voyelle)) porte sur l'objet formé r. Dans l'exercice tc1, la contrainte (effectif_attribut(xd,exactement,2,couleur,pique)) porte sur xd, un élément de l'ensemble à dénombrer. On pourrait imaginer un exercice où certaines contraintes portent sur l'objet formé et d'autres sur les éléments à dénombrer : « On forme des mains de 5 cartes ne contenant que des piques. Combien y a-t-il de mains ainsi formées qui contiennent un roi ? ».

Nous avons choisi de faire porter les contraintes sur les éléments de l'ensemble à dénombrer. Il nous faut donc dans certains cas (comme l'exercice m11) construire l'élément générique de l'ensemble à dénombrer. Il faut également déterminer quel est l'objet formé. L'objet formé est le résultat de l'action du problème ou l'élément générique s'il n'y a pas d'action. Il faut aussi s'assurer lorsqu'il y a une action que l'ensemble à dénombrer est inclus dans le résultat de l'action. SYRCLAD-dénombrements peut alors rapporter toutes les contraintes sur l'élément générique et dresser la liste de ces contraintes ; il utilise pour cela un paquet de règles qu'il sature.

Pour déterminer le type du tirage, il faut analyser la liste des contraintes. On détermine s'il y a une seule contrainte ou plusieurs ; si elles sont de même type ou non. On dresse la liste des attributs (type-lettre, numéro, couleur, parité) intervenant dans les contraintes, grâce à un paquet de règles. On détermine s'il y a un seul attribut ou non.

On peut également avoir besoin du type de l'action (former, tirer, attribuer un rôle), de la taille de l'objet formé (6 règles possibles), de la taille de l'ensemble où l'on choisit. L'ensemble où l'on choisit (7 règles) est un attribut important. On a besoin de l'observer pour reconnaître l'alphabet ou un jeu de cartes. On détermine les attributs possibles, (lettre et type-lettre pour l'alphabet, hauteur et couleur pour un jeu de cartes, etc.), le nombre de catégories qu'il définissent et la taille de ces catégories.

Décrivons à présent les règles qui concluent sur le type du problème et utilisent les attributs que nous venons de citer.

Lorsque l'objet formé est une concaténation de deux objets, le type du problème est un produit de sous-problèmes.

Lorsqu'il y a une seule contrainte de type effectif_attribut(au_plus,...) ou effectif_attribut(au_moins,...), le type du problème est soit disjonction linéaire, soit complémentaire, suivant ce qui est le plus avantageux en terme de calculs.

Lorsqu'il y a deux contraintes de type effectif_attribut portant sur des attributs différents qui définissent des catégories dont l'intersection est réduite à un élément, le type du problème est disjonction "dame de pique".

S'il y a aucune contrainte, le type du problème est une répartition avec comme ensemble caractéristique : ((taille de l'ensemble où l'on choisit, taille de l'objet formé,_)).

S'il y a une seule contrainte valeurs_identiques_pour(attribut), le type du problème est un spectre avec comme ensemble caractéristique ((1,taille de l'objet formé)).

S'il y a une seule contrainte valeurs_différentes_pour(attribut), le type du problème est un spectre avec comme ensemble caractéristique ((taille de l'objet formé,1)).

S'il y a plusieurs contraintes de même type classes_effectif_attribut concernant un seul attribut, c'est un spectre.

S'il y a une seule contrainte de type effectif_attribut(exactement,..), c'est une répartition dont l'ensemble caractéristique est réduit à un triplet.

S'il y a plusieurs contraintes de même type effectif_attribut(exactement,....) portant sur le même attribut, c'est une répartition.

S'il y a plusieurs contraintes de même type effectif_attribut(exactement,....) portant sur des attributs différents. C'est une répartition si les catégories définies sont disjointes.

S'il y a une ou plusieurs contraintes de type position, le type du problème est liste avec conditions de position.

Pour classer un problème, il faut aussi connaître le type de l'objet formé. Un mot, un nombre, une permutation sont des listes. Si l'objet est formé en tirant successivement (resp. simultanément), c'est une liste (resp. un ensemble).

Dans le cas où l'objet formé est de type liste, il faut déterminer la remise. Si l'on forme un résultat où chaque élément apparaît au plus ou exactement une fois, c'est sans remise. Si les répétitions sont possibles dans le résultat de l'action former, c'est avec remise ; si les répétitions ne sont pas possibles dans le résultat de l'action former, c'est sans remise. Si l'on forme une permutation, c'est sans remise puisque les éléments doivent être utilisés une et une seule fois. Par défaut, on forme un nombre ou un mot avec remise.

Nous avons vu au paragraphe 2.2 comment repérer les cas particuliers.

Il y a 85 règles pour déterminer les 27 attributs qui peuvent s'avérer utiles pour le classement. Certaines sont données en annexe sous leur forme Prolog.

2.4 Les modèles opérationnels

Un modèle opérationnel contient les attributs discriminants de la classification, dont les valeurs ont été déterminées pendant le classement. Le modèle opérationnel est complété par d'autres attributs, non discriminants, qui seront utiles pour la résolution. Ils permettent de caractériser le problème en tant qu'instance. On a vu au chapitre 1 que le problème m10 est de classe répartition-liste-avec, mais l'ensemble caractéristique est spécifique à m10 et permettra d'adapter au problème m10 la méthode de résolution associée à sa classe.

Le modèle opérationnel a été construit au fur et à mesure du classement. Si l'on a déterminé qu'un problème est de type répartition, on sait que l'on peut réécrire ce problème de la manière suivante :

> Soit un ensemble E de taille e divisé en m catégories disjointes d'effectifs c1, …, cm. On tire p éléments dans E. Combien y a-t-il de résultats contenant exactement x_1 éléments de catégorie 1, …, x_n éléments de catégorie n ?

Cette reformulation du problème est insuffisante, car ce nouveau problème ne se suffit pas à lui-même et on ne peut pas le résoudre. Si l'on a en plus déterminé que l'objet formé est de type ensemble, alors on peut reformuler le problème ainsi :

> Soit un ensemble E de taille e divisé en m catégories disjointes d'effectifs c1, …, cm. On tire p éléments dans E simultanément, et on forme un ensemble. Combien y a-t-il de résultats contenant exactement x_1 éléments de catégorie 1, …, x_n éléments de catégorie n ?

Cette reformulation du problème est satisfaisante. On n'a pas besoin de plus d'informations, on sait résoudre le problème ainsi posé. En effet, la classe répartition-ensemble est une classe opérationnelle.

En construisant ce modèle opérationnel, nous avons effectué la phase de modélisation qui rend les exercices de dénombrement difficiles. Il faut maintenant savoir quelle méthode appliquer, étant donné que nous connaissons la classe du problème.

2.5 Les connaissances de résolution

On associe à chaque classe opérationnelle de base un plan-type et une formule de calcul de la solution numérique.

Nous avons vu au paragraphe précédant comment formuler un problème de la classe répartition-ensemble. Le plan-type associé à cette classe est :

> On choisit x_1 <description catégorie 1> parmi c1,
> puis on choisit x_2 <description catégorie 2> parmi c_2,
> etc.,
> puis on complète en choisissant p - $(x_1+x_2+\ldots+x_n)$ éléments parmi les e - $(c_1+c_2+\ldots+c_n)$ éléments restants.

Pour calculer la solution numérique, la formule pour la classe répartition-ensemble est :

$$C_{c_1}^{x_1} \ldots C_{c_n}^{x_n} C_{e-(c_1+\ldots+c_n)}^{p-(x_1+\ldots+x_n)}$$

On donne en annexe tous les plans-type et toutes les formules associées aux classes opérationnelles de base. Pour les classes composées, on a vu au paragraphe 2.2 les méthodes de résolution qui leur sont associées.

La validité du plan-type et de la formule pour les problèmes de cette classe est prouvée par la théorie mathématique. Le groupe *Combien?* a montré la validité de cette "méthode constructive" dans [Le Calvez et al 97].

Nous avons étudié dans ce paragraphe 2 les bases de connaissances que l'on a données à SYRCLAD-dénombrements. Nous allons à présent présenter les résultats obtenus.

3 Résultats

Nous donnons ici quelques exemples significatifs d'exercices de dénombrement résolus par SYRCLAD-dénombrements. Nous donnerons un exemple pour chaque type de problème.

Répartition

Exercice j12 : « Un sac contient 19 jetons numérotés de 1 à 19. On tire simultanément 4 jetons dans le sac. Combien peut-on ainsi former d'ensembles contenant exactement un jeton portant un numéro pair ? ».

Voici le **modèle descriptif** correspondant à l'énoncé que l'on donne à SYRCLAD-dénombrements :

```
problème(j12).
action(j12,f).
on_dispose_de(j12,s).
à_dénombrer(j12,d).
est_un(f,action_former).
tout_résultat(f,r).
est_un(r,ensemble).
taille(r,4).
en_effectuant(f,t).
est_un(t,action_tirer).
simultanément(t).
récipient(t,s).
est_un(s,sac).
contenu(s,c).
est_un(c,ensemble).
tout_élément(c,xc).
est_un(xc,jeton).
éléments_numérotés_séquentiellement(c,1,19).
est_un(d,ensemble).
inclus_dans_résultats(d,f).
tout_élément(d,xd).
est_un(xd,ensemble).
effectif_attribut(xd,exactement,1,parité,pai
```

SYRCLAD-dénombrements classe le problème et construit le modèle opérationnel. Il utilise les **règles** suivantes :

```
j12
former obs_ens : élément_générique contraintes : liste_vide contraintes :
établir5 type_action taille_objet une_contrainte résultat_de_action
jeton_former taille_ens numérotation un_effectif type_précisé rep_e
```

SYRCLAD-dénombrements a déterminé que l'objet formé est r, que l'élément générique est xd, il construit la liste des contraintes. Il observe qu'il n'y a qu'une contrainte. Il détermine que l'ensemble où l'on choisit est c, qu'il est de taille 19, et que la catégorie des pairs est de taille 9. Il détermine le type du problème : répartition. Le type de l'objet est précisé : c'est un ensemble.

Il donne la **classe** du problème :

```
Exercice de classe c_répartition_ensemble
```

et le **modéle opérationnel** qu'il a construit en utilisant les règles :

```
j12 :
Type du tirage : répartition
Type de l'objet formé : ensemble
Taille de l'ensemble où l'on choisit : 19
Taille de l'objet formé : 4
Ensemble caractéristique : [[9, 1, pair]]
```

Il applique alors la méthode de résolution associée à la classe répartition-ensemble. Il produit la **solution** :

```
Plan de résolution : On choisit 1 pair parmi 9
puis on complète en choisissant 3 éléments parmi les 10 éléments restants.
Solution : 9 * C(10,3) = 1080
```

Liste avec conditions de position

Exercice m14 : « On appelle "mot" toute permutation de lettres données. Avec les lettres du mot BUNGALOW, combien peut-on former de mots de 8 lettres commençant par une consonne et finissant par deux voyelles ? ».`

Modèle descriptif

```
problème(m14).
action(m14,f).
à_dénombrer(m14,d).
est_un(f,action_former).
type_des_objets_formés(f,permutation)
.
au_moyen_de(f,a).
tout_résultat(f,r).
est_un(r,mot).
taille(r,8).
position(r,1,type_lettre,consonne).
position(r,8,type_lettre,voyelle).
position(r,7,type_lettre,voyelle).
est_un(a,ensemble).
ensemble_des_lettres_de(a,[b,u,n,g,a,
l,o,w]).
est_un(d,ensemble).
ens_des_résultats(d,f).
```

Règles

```
m14
former obs_ens : tout_élément obs_ens : élément_générique
résultat_de_action contraintes : rapporter4 contraintes : rapporter4
contraintes : rapporter4 contraintes : liste_vide contraintes : établir4
contraintes : établir4 contraintes : établir4 type_action taille_objet
même_type attributs : liste_att attributs : cons_att un_att
former_au_moyen_de taille_ens ens_lettres cat_mot positions
former_permutation l_p_s
```

SYRCLAD-dénombrements a déterminé que l'objet formé est r, il construit un élément générique de l'ensemble d. Il rapporte les 3 contraintes à l'élément générique et construit la liste des contraintes. Il observe que les contraintes sont de même type (position). Il construit la liste des attributs intervenant dans les contraintes ; il observe qu'il y a un seul attribut (type-lettre). Il détermine que l'ensemble où l'on choisit est a, qu'il est de taille 8, que la catégorie des voyelles est de taille 3 et celle des consonnes de taille 5. Il détermine le type du problème : liste avec conditions de position. Il n'y a pas remise car on forme des permutations (donc sans répétitions).

Classe

```
Exercice de classe c_liste_position_sans
```

Modèle opérationnel

```
m14
Type du tirage : liste avec conditions imposées à certaines positions
Sans remise
Taille de l'ensemble où l'on choisit : 8
```

```
Taille de l'objet formé : 8
Ensemble caractéristique : [[3, [7,8], voyelle], [5, [1], consonne]]
```

Solution

```
Plan de résolution : On choisit un(e) voyelle parmi 3 pour la position 7,
puis on choisit un(e) voyelle parmi 2 pour la position 8
puis on choisit un(e) consonne parmi 5 pour la position 1
puis on complète les 5 places restantes en choisissant 6 éléments tous
différents parmi les 5 éléments restants.
Solution : 3*2*5*A(5,5) = 3600
```

Spectre

Exercice tc56 : « Dans un jeu de 32 cartes, on tire 5 cartes simultanément pour former une main. Combien de mains peut-on ainsi former qui contiennent deux fois deux cartes de même couleur et une carte d'une autre couleur ? Par exemple : dame de cœur, 8 de cœur, 10 de trèfle, valet de trèfle, roi de carreau. ».

Modèle descriptif

```
problème(tc56).
on_dispose_de(tc56,j).
à_dénombrer(tc56,d).
action(tc56,f).
est_un(f,action_former).
en_effectuant(f,t).
est_un(t,action_tirer).
simultanément(t).
récipient(t,j).
nb_éléments_tirés(t,5).
est_un(d,ensemble).
inclus_dans_résultats(d,f).
tout_élément(d,xd).
est_un(xd,ensemble).
classes_effectif_attribut(xd,2,2,couleur).
classes_effectif_attribut(xd,1,1,couleur).
est_un(j,jeu_de_32_cartes).
```

Règles

```
tc56
obs_ens : élément_générique résultat_de_action objet contraintes :
rapporter4 contraintes : rapporter4 contraintes : liste_vide contraintes :
établir4 contraintes : établir4 type_action taille_former_tirer
former_tirer_jeu jeu_de_32_cartes taille_jeu att_jeu_carte même_type
attributs : liste_att attributs : cons_att un_att cat_jeu classes
type_précisé s_e
```

Classe

```
Exercice de classe c_spectre_ensemble
```

Modèle opérationnel

```
tc56
Type du tirage : spectre
Type de l'objet formé : ensemble
Taille de l'ensemble où l'on choisit : 32
Taille de l'objet formé : 5
Nombre de catégories : 4
Taille commune des catégories : 8
Ensemble caractéristique : [[1, 1], [2, 2]]
```

Solution

```
Plan de résolution : On choisit 1 couleur parmi 4, dans chacune on choisit
1 éléments parmi 8
puis on choisit 2 couleur parmi 3, dans chacune on choisit 2 éléments parmi
8.
Solution : 4*8*C(3,2)*C(8,2)*C(8,2) = 75264
```

Produit

Exercice m13 : « Combien y a-t-il de mots de 6 lettres commençant par une voyelle et dont le reste du mot est composé avec deux lettres, l'une présente 3 fois et l'autre présente 2 fois ? Exemple : abibib. ».

Modèle descriptif

```
problème(m13).
action(m13,f).
à_dénombrer(m13,d).
est_un(f,action_former).
au_moyen_de(f,a).
tout_résultat(f,r).
taille(r,6).
est_un(r,concaténation).
concaténation(r,[m,n]).
est_un(m,mot).
taille(m,1).
effectif_attribut(m,exactement,1,
      type_lettre,voyelle).
est_un(n,mot).
taille(n,5).
classes_effectif_attribut(n,1,2,lettre).
classes_effectif_attribut(n,1,3,lettre).
est_un(a,ensemble).
taille(a,26).
tout_élément(a,xa).
est_un(xa,lettre).
est_un(d,ensemble).
ens_des_résultats(d,f).
```

Règles

```
m13
former résultat_de_action concaténation former_au_moyen_de
```

Classe

```
Exercice de classe c_jointure
```

Solution

```
m13
Plan de résolution : On compte les possibilités pour certains éléments de
l'objet formé suivant une méthode, et les possibilités pour les autres
éléments suivant une autre méthode.
```

Les deux sous-problèmes sont p0 (former un mot de taille 1 contenant une voyelle) et p1 (former un mot de taille 5 contenant deux lettres, l'une présente 3 fois et l'autre présente 2 fois).

Il classe p0 :

```
p0
former obs_ens : Aucune règle exécutée contraintes : rapporter5 contraintes
: liste_vide contraintes : établir5 type_action taille_objet une_contrainte
alphabet taille_alphabet att_alphabet cat_alphabet un_effectif mot
former_défaut ens_carac_part p_c1
```

et donne la solution:

```
p0
Exercice de classe c_produit_cartésien
Solution : 6
```

puis il classe p1 :

```
p1
former obs_ens : Aucune règle exécutée contraintes : rapporter4 contraintes
: rapporter4 contraintes : liste_vide contraintes : établir4 contraintes :
établir4 type_action taille_objet alphabet taille_alphabet att_alphabet
même_type attributs : liste_att attributs : cons_att un_att cat_alphabet
classes mot former_défaut s_l_a
```

et donne le modèle opérationnel :

```
p1
Type du tirage : spectre
Type de l'objet formé : liste
Avec remise
Taille de l'ensemble où l'on choisit : 26
Taille de l'objet formé : 5
Nombre de catégories : 26
Taille commune des catégories : 1
Ensemble caractéristique : [[1, 3], [1, 2]]
```

et la solution :

```
Exercice de classe c_spectre_liste_avec
Plan de résolution : On choisit 1 lettre parmi 26, pour chacun(e) on
choisit 3 places, puis on remplit chacune de ces places avec un élément
pris parmi 1
puis on choisit 1 lettre parmi 25, pour chacun(e) on choisit 2 places, puis
on remplit chacune de ces places avec un élément pris parmi 1.
Solution : 26*C(5,3)*25 = 6500
```

Enfin, SYRCLAD-dénombrements donne la solution de l'exercice m13, produit des solutions de p0 et p1 :

```
m13 : Solution : 6500*6 = 39000
```

Complémentaire

Exercice j18 : « Dans un sac se trouvent 10 jetons verts numérotés de 1 à 10 et 10 jetons rouges numérotés de 1 à 10. On tire simultanément 6 jetons dans le sac. Combien y a-t-il de tirages contenant au moins 2 jetons verts? ».

Le fait que la contrainte contienne "au moins" doit tout de suite nous faire penser à une disjonction de problèmes : au moins 2 verts, c'est 2 verts ou 3 verts ou 4 verts ou 5 verts ou 6 verts. SYRCLAD-dénombrements choisit de passer par le complémentaire et de compter les tirages contenant 0 ou 1 vert.

```
j18
Exercice de classe c_complémentaire
Plan de résolution : On compte les résultats qui ne vérifient pas la
propriété demandée {p12}, puis on les soustrait du nombre total de
résultats possibles {p13}.
```

```
p13 {problème sans contraintes : tous les tirages}
Exercice de classe c_combinaison
Solution : C(20,6) = 38760

p12 {problème complémentaire : au plus 1 vert}
Exercice de classe c_disjonction_linéaire
Plan de résolution : On envisage plusieurs cas disjoints. {p15 et p16}

p15 {exactement 1 vert}
Exercice de classe c_répartition_ensemble
Plan de résolution : On choisit 1 vert parmi 10
puis on complète en choisissant 5 éléments parmi les 10 éléments restants.
Solution : 10*C(10,5) = 2520

p16 {exactement 0 vert}
Exercice de classe c_répartition_ensemble
Plan de résolution : On choisit 0 vert parmi 10
puis on complète en choisissant 6 éléments parmi les 10 éléments restants.
Solution : C(10,6) = 210

p12
Solution : 2520+210 = 2730

j18
Solution : 38760-2730 = 36030
```

Disjonction "dame de pique"

Exercice tc7bis : « On dispose d'un jeu de 32 cartes. On en tire 5 simultanément. Quel est le nombre de tirages contenant exactement 2 valets et 2 cœurs ? ».

SYRCLAD-dénombrements va considérer deux problèmes disjoints : celui où on tire de valet de cœur, et celui où on ne le tire pas.

```
tc7bis
Exercice de classe c_disjonction_ddp
Plan de résolution : On envisage plusieurs cas disjoints. {p17 et p18}

p17 {on a tiré le valet de cœur, on tire encore 4 cartes}
Type du tirage : répartition
Type de l'objet formé : ensemble
Taille de l'ensemble où l'on choisit : 31
Taille de l'objet formé : 4
Ensemble caractéristique : [[7, 1, cœur], [3, 1, valet]]
Exercice de classe c_répartition_ensemble
Plan de résolution : On choisit 1 cœur parmi 7
puis on choisit 1 valet parmi 3
puis on complète en choisissant 2 éléments parmi les 21 éléments restants.
Solution : 7*3*C(21,2) = 4410

p18 {on ne tire pas le valet de cœur}
Type du tirage : répartition
Type de l'objet formé : ensemble
Taille de l'ensemble où l'on choisit : 31
Taille de l'objet formé : 5
Ensemble caractéristique : [[7, 2, cœur], [3, 2, valet]]
Exercice de classe c_répartition_ensemble
```

```
Plan de résolution : On choisit 2 cœur parmi 7
puis on choisit 2 valet parmi 3
puis on complète en choisissant 1 éléments parmi les 21 éléments restants.
Solution : C(7,2)*C(3,2)*21 = 1323

tc7bis
Solution : 4410+1323 = 5733
```

4 Conclusion

Nous avons donné à SYRCLAD-dénombrements un graphe de classification de problèmes de dénombrement. Nous lui avons aussi donné les connaissances pour utiliser ce graphe, sous la forme de 85 règles de reformulation permettant de construire un modèle opérationnel du problème posé. SYRCLAD-dénombrements résout 76 exercices de dénombrement. L'encodage linguistique de ces exercices est varié : il y a des mots, des nombres, des cartes, des jetons… Il y a 14 classes opérationnelles dont chacune débouche sur une méthode de résolution propre. Les plans de résolutions sont ainsi variés et adaptés aux classes et aux problèmes.

Le groupe Combien?[1] avait défini un ensemble de classes de problèmes avec des méthodes de résolution associées. Nous avons construit un graphe de classification en définissant les classes intermédiaires et nous l'avons soumis à l'approbation du groupe Combien?. La réalisation de SYRCLAD-dénombrements a permis de valider l'hypothèse du groupe Combien? de l'utilisation d'une classification pour modéliser et résoudre des problèmes de dénombrements. Il a permis de systématiser le processus de classement et d'expliciter les règles de reformulation. Le groupe Combien? peut maintenant se baser sur le graphe de classification établi pour SYRCLAD-dénombrements et sur les résultats de ce système.

La réalisation de SYRCLAD-dénombrements n'a pas modifié la classification des problèmes de dénombrement du groupe Combien?. Cependant, la nécessité d'appeler recursivement SYRCLAD dans la méthode de résolution de certaines classes a mis en évidence une distinction importante entre différentes méthodes : celles qu'on pourrait appeler "simples" (pas d'appel récursif) et celles qu'on pourrait appeler "composées". La formalisation d'une méthode de résolution "constructive" [Le Calvez et al 97] tient compte de cette distinction : une "construction" correspond aux méthodes simples et un "plan" peut être une construction ou une composition de constructions, et donc peut correspondre à une méthode composée. La progression pédagogique prévue commence par familiariser les élèves avec les constructions (grâce aux "machines à construire") avant d'introduire des manipulations de plans.

La réalisation de SYRCLAD-dénombrements nous à conduit à modifier un système à base de règles [Guin 94, Guin et al 95] pour distinguer les connaissances de classification, de reformulation et de résolution contenues dans ces règles. Ce travail demande de "déclarativiser" les connaissances

[1] H. Giroire et G. Tisseau de Paris6, F. Le Calvez et M. Urtasun de Paris 5 et J. Duma du Lycée Jacquard.

exprimées par les règles. Cela n'a pas posé de problèmes significatifs car ce système à base de règles avait été conçu avec une idée de classification des problèmes.

Chapitre 5

Problèmes additifs :

tester SYRCLAD en implémentant une classification donnée

Nous présentons dans ce chapitre l'application de SYRCLAD au domaine des problèmes additifs, c'est-à-dire SYRCLAD-additifs. Nous nous intéresserons tout d'abord aux spécificités de ce domaine ; nous indiquerons pourquoi il était intéressant d'appliquer SYRCLAD aux problèmes additifs. Nous détaillerons ensuite les connaissances données à SYRCLAD-additifs, et en particulier la classification. Enfin nous donnerons des exemples de résolutions d'exercices par SYRCLAD-additifs afin d'en évaluer les résultats.

Plan du chapitre

1 Pourquoi les problèmes additifs ?

1.1 Un domaine difficile

Le domaine des problèmes additifs est un des domaines les plus largement étudiés en didactique des mathématiques, psychologie cognitive, et linguistique.

Les problèmes additifs décrivent une situation concrète, souvent un jeu de billes. Par exemple : « Paul a joué deux parties de billes. A la première partie il a perdu 3 billes, à la seconde partie il en a gagné 5. Que s'est-il passé en tout ? » .

Certains sont bien résolus dès la maternelle, d'autres posent encore des difficultés en fin de troisième [Marthe 82]. Nous souhaitons modéliser le processus de compréhension et de résolution de ces exercices à un niveau permettant une explication détaillée pour les élèves. En modélisant le comportement d'un élève "idéal"[1], nous souhaitons pouvoir aider les élèves à acquérir une méthode générale de résolution de ces problèmes.

Le problème ci-dessus nous paraît simple à résoudre. Mais n'oublions pas que nous nous intéressons à la résolution de ces problèmes par des enfants. Nous ne devons donc pas utiliser plus de connaissances que celles que possèdent les élèves. Les adultes modélisent ces problèmes dans $\mathbb{Z}$ (l'ensemble des entiers relatifs) et la résolution consiste alors à résoudre une équation dans $\mathbf{Z}$ [Rebillard 95]. Mais les élèves ne connaissent que $\mathbf{N}$ (l'ensemble des entiers naturels), ils doivent donc rechercher d'autres stratégies de résolution de ces problèmes additifs.

Le domaine des problèmes additifs a fait l'objet de nombreuses recherches cognitives, linguistiques et didactiques. Le Chapitre 6 de L'enfant et le nombre [Fayol 90] fait une synthèse très complète de ces études.

M.S. Riley et al ont proposé dans [Riley et al 83] une classification des problèmes additifs qui distingue trois catégories de problème : combinaison, changement et comparaison.

La catégorie *combinaison* porte sur des situations statiques :
« Jean a 3 billes. Pierre a 4 billes. Jean et Pierre ont 7 billes. »
pour lesquels il faut trouver un total ou un état initial.

Le catégorie *changement* décrit une transformation "temporelle" appliquée à un état initial et aboutissant à un état final :
« Jean avait 3 billes. Il en a gagné 5. Jean a maintenant 8 billes. »

---

[1] au sens d'Anderson [Anderson et al 87] : expliciter comment l'élève idéal devrait se comporter

L'inconnue peut concerner l'état initial, la transformation ou l'état final. La transformation peut être additive ou soustractive.

La catégorie *comparaison* compare des quantités statiques à l'aide de formules "plus de / moins de".

«Jean a 3 billes. Pierre a 5 billes. Pierre a 2 billes de plus que Jean. »

La classification établie par G. Vergnaud [Vergnaud 82] subsume la classification de M.S. Riley et al. Il distingue le calcul numérique du *calcul relationnel* dans lequel réside la tâche principale de résolution des problèmes additifs. Ce calcul relationnel peut porter sur des *mesures* ou sur des *transformations.* Cette distinction permet une approche plus *opératoire* des modélisations cognitives. La catégorie "combinaison" est alors une composition de mesures, la catégorie "changement" est l'opération d'une transformation sur une mesure pour donner une mesure et la catégorie "comparaison" est une relation statique entre deux mesures. G. Vergnaud introduit également la *composition de deux transformations* :

« Jean a gagné 6 billes. Ensuite, il a gagné 3 billes. En tout, il a gagné 9 billes. »

Lorsque l'on évalue les taux de réussite à ces problèmes, on se rend compte que ce sont des problèmes difficiles. On remarque que la difficulté n'est pas liée à l'opération mathématique sous-jacente : une transformation négative ne s'avère pas nécessairement plus difficile qu'une transformation positive. En revanche les problèmes de la catégorie "changement" semblent plus faciles que ceux de la catégorie "combinaison", les problèmes de type "comparaison" s'avérant les plus difficiles. C'est la nature[1] de l'inconnue qui entraîne le plus de difficultés. De plus, G. Vergnaud montre dans [Vergnaud 82] que si les problèmes de changement ne posent plus de réelles difficultés à partir de 8-9 ans, ceux qui composent deux transformations entraînent encore de fréquents échecs à 10-11 ans. P. Marthe a observé les même difficultés pour ces problèmes de composition de transformations en fin de troisième, vers 15 ans [Marthe 82].

J.G. Greeno and M.S Riley montrent dans [Greeno et Riley 87] que les jeunes enfants possèdent des méthodes de résolution pour les problèmes additifs. Ils éprouvent des difficultés à les résoudre parce qu'ils n'arrivent pas à se représenter correctement la situation décrite dans l'énoncé. Ce qui permet à des enfants plus âgés de bien résoudre les problèmes additifs est leur capacité à modéliser les problèmes.

Hormis les connaissances mathématiques et les méthodes de modélisation des problèmes, la résolution de problèmes additifs nécessite des connaissances contextuelles implicites permettant la transformation du problème pour se ramener à un calcul sur les entiers naturels. Par exemple, il faut penser que si Pierre a gagné 5 billes, cela veut dire que Paul en a perdu autant. À un autre niveau, le fait que la composition des opérateurs correspondant à deux parties jouées successivement est commutative se révèle très utile pour transformer le problème, mais ce n'est pas enseigné aux élèves.

---

[1] état initial, transformation ou état final

Nous détaillerons au paragraphe 2.2 la classification que nous avons utilisée. Elle concerne les problèmes de combinaison, changement et composition de transformations. Cette classification est basée sur la notion d'opérateur. Ces opérateurs permettent de modéliser les problèmes dans $\mathbb{N}$.

1.2 Pourquoi appliquer SYRCLAD aux problèmes additifs ?

L'application aux problèmes additifs a permis de tester l'architecture de SYRCLAD et en particulier son indépendance par rapport au domaine. Nous disposions d'une classification de problèmes établie par des didacticiens, sur papier. A partir de cette classification et de l'architecture de SYRCLAD, nous avons construit un résolveur de problèmes additifs : SYRCLAD-additifs. Pour cela, nous avons exprimé cette classification dans le formalisme de SYRCLAD et nous avons donné à SYRCLAD-additifs les connaissances nécessaires pour l'exploiter, conformément aux bases de connaissances définies dans l'architecture de SYRCLAD.

2 Les bases de connaissances de SYRCLAD-additifs

2.1 Les modèles descriptifs

Les problèmes que nous considérons relatent des parties de billes. Il peut s'agir de problèmes de type combinaison, changement et composition de changements.

On manipule des ensembles de billes :

"5 billes"

```
est_un(b5,ensemble).
tout_élément(b5,y).
est_un(y,bille).
taille(b5,5).
```

Des personnes possèdent des ensembles de billes à un instant donné :

"pierre a 5 billes"

```
est_un(pierre,personne).
possède(pierre,t1,b5).
```

Une partie de billes est définie par un instant initial et un instant final. Il y a deux participants.

```
est_un(p,partie).
participants(p,pierre,jean).
partie(p,t1,t2).
```

Un participant perd ou gagne un ensemble de billes pendant une partie.

"Pierre a perdu 5 billes"

```
perdu(pierre,p,b5).
```

"Pierre a gagné 5 billes"

```
gagné(jean,p,b5).
```

"Pierre a joué une partie p, on ne sait pas s'il a gagné ou perdu des billes"

```
changement(pierre,p,bc).
```

On peut définir un ensemble de personnes :

```
est_un(p,ensemble_de_personnes).
éléments(p,[pierre,jean]).
```

Il peut y avoir une succession de parties :

```
est_un(p,succession_de_parties)).
succession(p,[[p1,t1,t2],[p2,t2,t3]]).
```

Dans chaque problème, il y a une action (une partie ou une succession de parties), et un entier à calculer :

```
problème(b332).
action(b322,p).
à_calculer(b332,n).
```

Les prédicats que nous venons de décrire définissent le langage de description des problèmes additifs qu'il faut utiliser pour proposer un problème à SYRCLAD-additifs.

2.2 La classification

Des stratégies de comptage ont été mises en évidence chez les élèves dans la résolution de problèmes additifs. Il s'agit de "séparer de", "compter vers l'arrière à partir de", "ajouter", "compter vers l'avant à partir de", "mise en correspondance"...T.P. Carpenter et J.M. Moser ont établi dans [Carpenter et Moser 83] des relations entre les catégories de problèmes définies en 1.1 et ces stratégies de comptage. Malheureusement, ces stratégies de comptage sont dépendantes du contexte.

N Bensimon-Darcel a étudié la modélisation d'énoncés de problèmes additifs en langage naturel [Bensimon-Darcel 93]. De même, M.D. LeBlanc et S. Weber-Russel présentent dans [Leblanc et Weber-Russel 96] un système sensible aux faibles variations dans l'énoncé du problème, pour tenir compte des facteurs linguistiques. La modélisation débouche sur le choix d'une stratégie de comptage parmi celles décrites ci-dessus.

La classification que nous avons utilisée est proposée par D. Guin [Guin 91]. Le but de cette classification est de permettre aux élèves de reconnaître la situation qui correspond à l'exercice et de la modéliser dans $\mathbb{N}$ grâce à des opérateurs. Elle explicite une méthode générale de modélisation et de résolution d'une large catégorie de problèmes additifs inspirée des travaux de G. Vergnaud [Vergnaud 85]. Les méthodes de résolution ne sont pas les stratégies de comptage décrites plus haut, mais des opérations à effectuer. Le résultat important est la modélisation en terme d'opérateurs obtenue par le classement de l'exercice, l'opération à effectuer s'en déduit immédiatement.

Les deux tableaux de la figure 1 décrivent la classification utilisée. La syntaxe f : x -> y signifie que l'opérateur f appliqué à l'opérande x donne le résultat y. Le point d'interrogation représente l'inconnue dont il faut trouver la valeur. Le premier tableau regroupe les problèmes simples, le deuxième les problèmes composés.

	Opérateurs	Types de problèmes	Solutions
1.1	ADD	ADD : (a, b) -> ?	? = a + b
1.2		ADD : (a, ?) -> b	? = b - a
2.1	AJOUT n	AJOUT b : a -> ?	? = a + b
2.2		AJOUT b : ? -> a	? = a - b
2.3		AJOUT ? : a -> b	? = b - a
3.1	RET n	RET b : a -> ?	? = a - b
3.2		RET b : ? -> a	? = a + b
3.3		RET ? : a -> b	? = a - b

	Types de problèmes	Solutions
4.1	AJOUT a o AJOUT b = ?	? = AJOUT (a + b)
4.2	? o AJOUT b = AJOUT a	
4.2.1	avec a > b	? = AJOUT (a - b)
4.2.2	avec a < b	? = RET (b - a)
5.1	RET a o RET b = ?	? = RET (a + b)
5.2	? o RET b = RET a	
5.2.1	avec a > b	? = RET (a - b)
5.2.2	avec a < b	? = AJOUT (b - a)
6.1	AJOUT a o RET b = ?	
6.1.1	avec a > b	? = AJOUT (a - b)
6.1.2	avec a < b	? = RET (b - a)
6.2.1	? o RET b = AJOUT a	? = AJOUT (a + b)
6.2.2	? o AJOUT b = RET a	? = RET (a + b)

Figure 1 : Classification des problèmes additifs de D. Guin

Les problèmes simples font intervenir un seul opérateur.

L'opérateur binaire ADDitionner agit sur un couple de valeurs dont il fait la somme. Ces problèmes appartiennent à la catégorie "combinaison" décrite au paragraphe 1.1. L'inconnue peut être soit le résultat de l'opération, soit un opérande.

Voici un exemple de problème de la classe 1.1 : « Jean a 4 billes, Amélie 3. Combien ont-ils de billes ensemble ? ». ADD : (4,3) -> ?

Les opérateurs unaires AJOUTer n et RETrancher n agissent sur une valeur à laquelle ils ajoutent ou retranchent la valeur n. Ces problème appartiennent à la catégorie "changement" décrite au

paragraphe 1.1. L'inconnue dont il faut déterminer la valeur peut être le résultat de l'opération, ou l'opérande, ou même l'opérateur.

Voici un exemple de problème de la classe 2.2 : « Pierre avait des billes. Il a gagné 5 billes en jouant avec Paul. Pierre a maintenant 8 billes. Combien en avait-il avant de jouer ? ».

AJOUT 5 : ? -> 8

Voici un exemple de problème de la classe 3.3 : « Paul a joué une partie de billes. Il avait 41 billes avant de jouer. Il en a maintenant 29. Que s'est-il passé pendant la partie ? ». RET ? : 41 -> 29

Les problèmes composés correspondent à des compositions de deux opérateurs. Ils appartiennent à la catégorie "composition de transformations" décrite au paragraphe 1.1. Il s'agit de deux parties de billes successives. L'inconnue peut être l'opérateur composé ou un des deux opérateurs que l'on compose.

Voici un exemple de la classe 4.2 : « Pierre a joué deux parties de billes. A la première partie il a gagné 16 billes. Il a joué une seconde partie. En faisant ses comptes, il s'aperçoit qu'il a gagné 20 billes en tout. Que s'est-il passé à la seconde partie ? ». AJOUT 16 o ? = AJOUT 20

Cette classification sur papier donne 18 classes opérationnelles. Certaines classes intermédiaires non-opérationnelles apparaissent, mais pas toutes. Nous avons construit un graphe de classification en nous basant sur cette classification papier. Ce graphe est présenté en entier mais succinctement en figure 2. La figure 3 représente une partie du graphe de classification des problèmes additifs de manière plus détaillée.

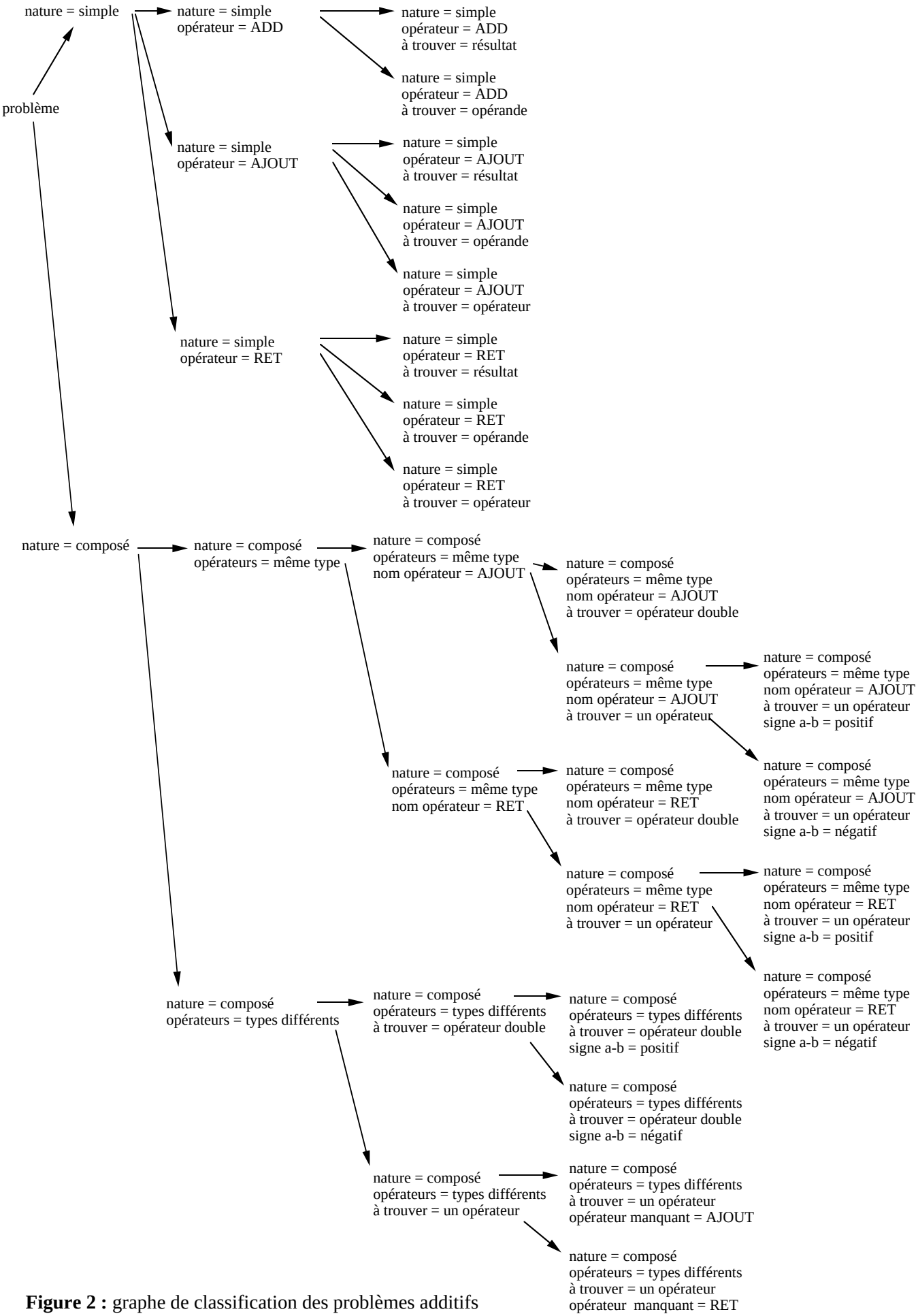


Figure 2 : graphe de classification des problèmes additifs

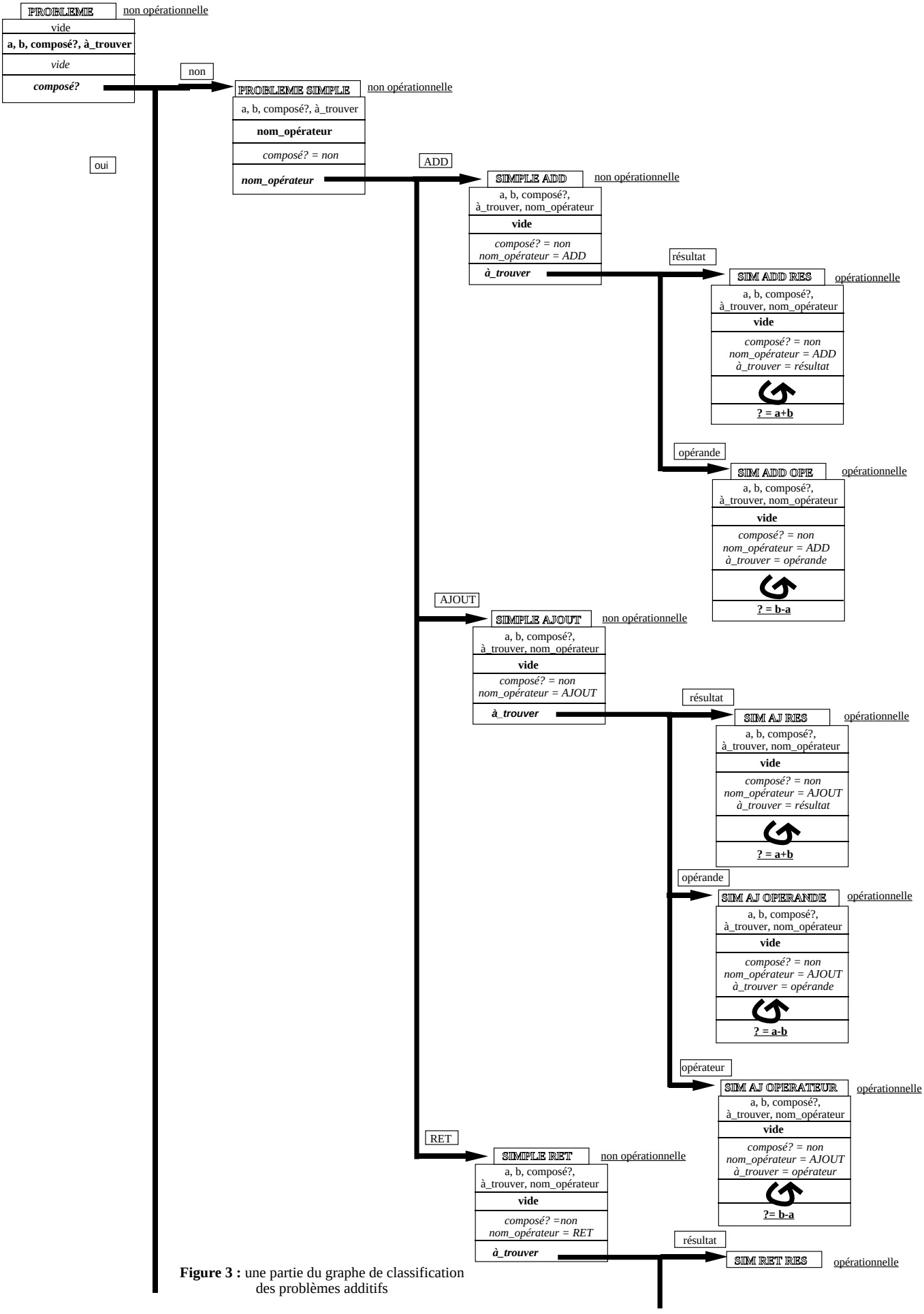


Figure 3 : une partie du graphe de classification des problèmes additifs

Le premier critère discriminant de la classification distingue les problèmes composés (deux parties successives) des problèmes simples. Pour les problèmes simples, on analyse la situation afin de déterminer l'opérateur modélisant cette situation : ADD, AJOUT ou RET. Pour les problèmes ADD, l'inconnue peut être soit le résultat, soit un opérande. Pour les problèmes AJOUT et RET, l'inconnue peut être soit le résultat, soit l'opérande, soit l'opérateur.

Pour les problèmes composés, les deux opérateurs sont soit de même type (deux AJOUT ou deux RET), soit de types différents (un AJOUT et un RET). Quand ils sont de même type, il peut s'agir d'AJOUT ou de RET. Dans chaque cas, l'inconnue peut être l'opérateur résultant ou un des deux opérateurs composés. Dans ce dernier cas, le signe de la différence entre les données intervient. Lorsque les deux opérateurs sont de type différent, l'inconnue peut être l'opérateur résultant ou l'un des deux opérateurs. Quand c'est l'opérateur résultant, le signe de la différence entre les données intervient. Quand c'est l'un des deux opérateurs, celui qu'il faut trouver est soit AJOUT, soit RET.

On voit ici que la résolution n'est pas aussi immédiate qu'il n'y paraît de prime abord, car elle nécessite en effet la prise en compte de plusieurs informations de types différents.

> Lorsqu'une partie fait intervenir deux partenaires, la question peut être « Que s'est-il passé en tout ? », sans préciser celui auquel on s'intéresse. C'est pourquoi on peut ajouter au graphe de classification un attribut discriminant intéressé, dont la valeur est "connu" ou "inconnu". S'il est connu, on descend à la racine du graphe de la figure 2. Si intéressé = inconnu, le problème est de classe "inconnu", classe opérationnelle dont la méthode de résolution indique de résoudre d'abord le problème en s'intéressant à l'un, puis de le résoudre en s'intéressant à l'autre. On peut considérer que c'est une résolution composée. Les deux solutions aux deux sous-problèmes seront équivalentes (Pierre a perdu 5 billes, Jacques a gagné 5 billes). Nous pouvons noter qu'il est aisé dans l'architecture de SYRCLAD d'ajouter une classe aux graphe de classification, ainsi que les connaissances de reformulation et de résolution associées. Par souci de clarté, nous n'utliserons dans la suite que le graphe présenté en figure 2.

Pour utiliser le graphe de classification que nous avons décrit, SYRCLAD-additifs a besoin de règles de reformulation.

2.3 Les connaissances de reformulation

Pour pouvoir utiliser le graphe de classification et déterminer la valeur des attributs discriminants, d'autres attributs sont définis :

- l'intéressé : la personne sur laquelle porte la question ; il y a en effet deux participants, il faut savoir si l'on s'intéresse aux billes possédées par le premier ou par le second.

- la liste des opérateurs intervenant dans une succession de deux parties ; elle est déterminée grâce à un paquet de règles.

Nous avons aussi besoin d'attributs non-discriminants permettant d'instancier le modèle opérationnel au problème et qui seront utilisés par la méthode de résolution. Il s'agit ici des deux entiers naturels a et b. Leur valeur est déterminée en même temps que celle de l'attribut à_trouver précisant la nature de l'inconnue.

Nous ne détaillerons pas ici les règles permettant de déterminer la valeur des attributs. Ces règles sont assez simples ; elles sont au nombre de 29.

Prenons l'exemple d'un problème simple. Pour trouver l'opérateur modélisant la situation, on utilise les connaissances suivantes :

Si deux personnes possèdent des billes et l'ensemble de ces deux personnes possède des billes, c'est ADD.

Si l'intéressé gagne un ensemble de billes pendant une partie, c'est AJOUT.

Si l'intéressé possède un ensemble de billes à l'état final dont la taille est supérieure à celle de l'ensemble de billes qu'il possédait à l'état initial, c'est AJOUT.

Si l'intéressé perd un ensemble de billes pendant une partie, c'est RET.

Si l'intéressé possède un ensemble de billes à l'état final dont la taille est inférieure à celle de l'ensemble de billes qu'il possédait à l'état initial, c'est RET.

2.4 Les modèles opérationnels

Les modèles opérationnels sont composés d'une part des attributs discriminants et de leurs valeurs déterminées pendant le classement et d'autre part de deux attributs non-discriminants, qui sont les deux données numériques a et b, pour instancier le problème par rapport à la classe et pour appliquer la méthode de résolution au problème.

Exemple : « Pierre a joué deux parties de billes. A la première partie il a gagné 16 billes. Il a joué une seconde partie. En faisant ses comptes, il s'aperçoit qu'il a gagné 20 billes en tout. Que s'est-il passé à la seconde partie ? »

Le modèle opérationnel construit par SYRCLAD-additifs est le suivant :

```
b4211
Exercice de classe c_composé_mtype_ajout_un_pos
nature du problème : composé
opérateurs de même type
type : AJOUT
à_trouver : un_opérateur
signe a-b : positif

a = 20
b = 16
```

On peut alors reformuler le problème ainsi :

AJOUT 16 o ? = AJOUT 20

3 Résultats

Nous donnons ici des exemples caractéristiques de résolution de problèmes additifs par SYRCLAD-additifs.

Exercice b12.1

« Pierre et Jean ont ensemble 8 billes. Pierre a 3 billes. Combien Jean a-t-il de billes ? ».

```
problème(b121).
est_un(pierre,personne).
est_un(jean,personne).
est_un(p,ensemble_de_personnes).
éléments(p,[pierre,jean]).
possède(p,i,e).
est_un(e,ensemble).
tout_élément(e,z).
est_un(z,bille).
taille(e,8).
possède(pierre,i,b3).
est_un(b3,ensemble).
tout_élément(b3,x).
est_un(x,bille).
taille(b3,3).
possède(jean,i,b5).
est_un(b5,ensemble).
tout_élément(b5,y).
est_un(y,bille).
taille(b5,t).
à_calculer(b121,t).
```

SYRCLAD classe le problème et construit le modèle opérationnel : ADD(3,?) -> 8

```
b121
nature_simple2 op_add at2
```

Il produit le résultat :

```
b121
Exercice de classe c_simple_add_opérande
Solution : 5
```

Exercice b32.2

« Bertrand joue aux billes. Il perd 7 billes. Après le jeu, il lui reste 3 billes. Combien de billes avait-il avant le jeu ? ».

```
problème(b322).
action(b322,p).
est_un(p,partie).
participants(p,bertrand,i).
est_un(bertrand,personne).
partie(p,t1,t2).
possède(bertrand,t1,e).
est_un(e,ensemble).
tout_élément(e,z).
est_un(z,bille).
taille(e,t).
perdu(bertrand,p,b7).
est_un(b7,ensemble).
tout_élément(b7,y).
est_un(y,bille).
taille(b7,7).
possède(bertrand,t2,b3).
est_un(b3,ensemble).
tout_élément(b3,x).
est_un(x,bille).
taille(b3,3).
à_calculer(b322,t).
```

SYRCLAD classe le problème et construit le modèle opérationnel : RET 7 : ? -> 3

```
b322
nature_simple1 i2 op_ret1 at4
```

Il produit le résultat :

```
b322
Exercice de classe c_simple_ret_opérande
Solution : 10
```

Exercice b41.2

« Hier, Paul et Pierre ont joué deux parties de billes. A la première partie, Paul a gagné 5 billes. A la seconde partie, Pierre a perdu 4 billes. Que s'est-il passé en tout pour Paul ? ».

```
problème(b412).
action(b412,p).
est_un(p,succession_de_parties).
nombre_de_parties(p,2).
succession(p,[[p1,t1,t2],[p2,t2,t3]]).
participants(p,pierre,paul).
est_un(pierre,personne).
est_un(paul,personne).
gagné(paul,p1,b5).
est_un(b5,ensemble).
tout_élément(b5,x).
est_un(x,bille).
taille(b5,5).
perdu(pierre,p2,b4).
est_un(b4,ensemble).
tout_élément(b4,y).
est_un(y,bille).
taille(b4,4).
changement(paul,p,bc).
est_un(bc,ensemble).
tout_élément(bc,z).
est_un(z,bille).
taille(bc,c).
à_calculer(b412,c).
```

SYRCLAD classe le problème et construit le modèle opérationnel :

```
b412
nature_composé op : init i3 op : ajout op : ajout top1 nop at6
```

Il produit le résultat suivant :

```
b412
Exercice de classe c_composé_mtype_ajout_double
Opération : Paul gagné 5 et Paul gagné 4 = ?
Solution : Paul gagné 9
```

On remarque ici l'utilisation par SYRCLAD-additifs du fait que "Pierre a perdu 4 billes" est équivalent à "Paul a gagné 4 billes".

Exercice b421.2

« Frédéric a joué deux parties de billes. A la seconde partie il a gagné 3 billes. Il ne se souvient plus de ce qui s'est passé à la première partie. Mais quand il compte ses billes à la fin, il s'aperçoit qu'il a gagné 8 billes en tout. Que s'est-il passé à la première partie ? ».

SYRCLAD déclenche 10 règles et produit le résultat suivant :

```
b4212
Exercice de classe c_composé_mtype_ajout_un_pos
Opération : ? et Frédéric gagné 3 = Frédéric gagné 8
Solution : Frédéric gagné 5
```

Exercice b621.1

« Pierre a joué deux parties de billes. Au cours de la première partie il en a perdu 3. Il a joué une seconde partie. En faisant ses comptes pour les deux parties il s'aperçoit qu'il a gagné 2 billes en tout. Que s'est-il passé à la seconde partie ? ».

SYRCLAD déclenche 9 règles et produit le résultat suivant :

```
b6211
Exercice de classe c_composé_typediff_un_ajout
Opération : Pierre perdu 3 et ? = Pierre gagné 2
Solution : Pierre gagné 5
```

4 Conclusion

Les modèles descriptifs des problèmes additifs sont variés. Le graphe de classification donné à SYRCLAD-additifs est relativement grand. Il y a 29 règles de reformulation et 18 classes opérationnelles possibles pour les problèmes additifs ; à chacune est associée une opération à effectuer pour obtenir la solution. Deux données permettent d'appliquer les méthodes de résolution aux problèmes. SYRCLAD-additifs résout grâce à cette classification 30 problèmes variés. Nous verrons au chapitre 9 qu'avec la méthode de décomposition de problèmes que nous avons donnée à SYRCLAD-additifs, une vingtaine d'exercices supplémentaires sont résolus.

Pour expliciter le graphe de classification que nous avons donné à SYRCLAD-additifs, nous avons dû construire des classes intermédiaires par rapport à la classification sur papier que nous avons utilisée, ce qui permet souvent de dégager des concepts intéressants. Nous avons également dû définir un langage de description des problèmes, et élaborer les règles de reformulation permettant de reformuler un problème concret en termes d'opérateurs. Les résultats de SYRCLAD-additifs permettent de valider la classification basée sur les opérateurs que nous avons implémentée.

Chapitre 6

Tarot : rendre explicite une classification cachée dans des règles

Nous présentons dans ce chapitre l'application de SYRCLAD au domaine du tarot, c'est-à-dire SYRCLAD-tarot. Nous nous intéresserons tout d'abord au jeu de tarot ; nous indiquerons pourquoi il était intéressant d'appliquer SYRCLAD à ce domaine. Nous détaillerons ensuite les connaissances données à SYRCLAD-tarot pour choisir un plan de jeu au tarot, et en particulier la classification. Enfin nous donnerons des exemples de plans choisis par SYRCLAD-tarot afin d'en évaluer les résultats.

Plan du chapitre

1 Pourquoi le tarot ?

1.1 Le jeu de tarot

Un jeu de tarot se compose de 78 cartes : 14 cartes pour les couleurs carreau, cœur, trèfle et pique (as, deux, ..., dix, valet, cavalier, dame, roi) et 22 *atouts* (du 1 (le petit) au 21, plus l'excuse). Nous nous intéressons au jeu à quatre joueurs. On distribue 18 cartes à chacun des quatre joueurs et il reste 6 cartes qui forment le *chien*. Le jeu commence par des *enchères* ; il y a quatre enchères possibles : la *prise*, la *garde*, la *garde-sans* et la *garde-contre*. Celui qui a annoncé l'enchère la plus haute devient le joueur attaquant. Si l'attaquant a annoncé comme enchère une prise ou une garde, il prend les 6 cartes du chien, les inclut dans son jeu puis choisit 6 cartes qu'il retire de son jeu afin de constituer *l'écart*. S'il a fait une garde-sans ou une garde-contre, il ne regarde pas les cartes du chien et ne constitue pas d'écart. Avant de commencer le jeu proprement dit, le joueur attaquant doit élaborer sa stratégie en choisissant un *plan de jeu*. Après avoir joué les cartes, on compte les points réalisés par l'attaquant à travers les cartes des plis qu'il a remportés.

1.2 Pourquoi appliquer SYRCLAD au tarot ?

J.-M. Nigro a réalisé Bateleur, un système qui joue au tarot [Nigro 95] ; ce système possède une expertise de choix du plan de jeu. L'application de SYRCLAD au tarot est basée sur l'étude de la base de règles de Bateleur conçue pour choisir un plan de jeu en début d'une partie de tarot. Il semblait que cette base de règles utilisait sans le dire une classification ; en effet, dans la partie condition des règles apparaissaient trois ou quatre attributs, que l'on peut considérer comme des critères de classification, et en partie conclusion apparaissait le plan de jeu choisi (cf. annexe).

L'architecture de SYRCLAD a permis d'expliciter une classification basée sur les attributs présents dans les règles. Le système SYRCLAD-tarot résout, en utilisant cette classification, les mêmes problèmes que la base de règles initiale.

2 Les bases de connaissances de SYRCLAD-tarot

2.1 Les modèles descriptifs

On a vu au paragraphe 1.1 le déroulement du début d'une partie de tarot. Au moment de choisir un plan de jeu, l'attaquant connaît les dix-huit cartes qu'il a en main et éventuellement les six cartes qu'il a écartées. En fonction de la qualité des cartes qu'il possède, il choisira des plans de jeu différents.

On pose le problème à SYRCLAD-tarot en lui donnant le contrat, les cartes de l'attaquant et les cartes de l'écart quand elles sont connues. Ce sont les seules données dont il a besoin.

Pour la partie p1 par exemple :

Contrat : prise
Écart : ♦ 1 ♥ 10 6 3 ♠ 10 7
Main de l'attaquant : ♣ R C 9 8 6 4 2
♦ R
♠ R
atout 21 20 19 17 10 5 4 1 excuse

Le système reçoit ces données en Prolog sous une forme équivalente :

```
problème(p1).
joueur(p1,j1).
contrat(p1,j1,prise).
écart(p1,j1,[[pique,dix],[pique,sept],[cœur,dix],[cœur,six],[cœur,trois],[c
arreau,as]]).
main(p1,j1,pique,[roi]).
main(p1,j1,cœur,[]).
main(p1,j1,carreau,[roi]).
main(p1,j1,trèfle,[roi,cavalier,neuf,huit,six,quatre,deux]).
main(p1,j1,atout,[21,20,19,17,10,5,4,1,exc]).
```

Prenons un autre exemple :

Contrat : garde-contre
Main de l'attaquant : ♦ R 6
♠ R D 8
♥ V
atout 21 20 19 18 17 16 8 7 6 5 1 exc

Comme le contrat est une garde-contre, l'attaquant n'a pas eu le chien et n'a pas fait d'écart, il ne connaît donc que les cartes qu'il a en main.

Pour tous les problèmes, la question est la même : choisir un plan de jeu.

2.2 La classification

Dans la base de règles de Bateleur, le choix du plan de jeu dépend de quatre critères : la hauteur d'atout, la longueur d'atout, le nombre de perdantes et la possession du petit.

La longueur d'atout

Il est important pour le joueur de posséder beaucoup d'atouts. On considère le nombre d'atouts moyen (22 divisé par 4 = 5). Le joueur possède beaucoup d'atouts s'il en a au moins cinq de plus que le nombre moyen, c'est-à-dire dix.

La hauteur d'atout

C'est bien de posséder beaucoup d'atouts, mais pour que ce soit efficace, il faut en avoir de très forts. Sur les sept atouts les plus forts (de 21 à 15), on compte ceux qui manquent à l'attaquant ; s'il y en plus de deux qui manquent, la hauteur d'atout est faible.

Le nombre de perdantes

Le but de l'attaquant est de perdre le moins de plis possibles, il doit donc évaluer le nombre de cartes perdantes qu'il possède dans chacune des quatre couleurs (trèfle, carreau, pique, cœur). On se place dans le cas le pire où un seul joueur possède toutes les cartes de la couleur considérée qui ne sont ni dans la main du joueur ni dans l'écart. Dans le cas d'une garde-sans ou d'une garde-contre où l'on ne connaît pas l'écart, on suppose que toutes les cartes sont à l'adversaire. On compare les deux mains de cartes, une carte gagnante de l'attaquant éliminant la plus petite carte du défenseur et une carte perdante de l'attaquant étant éliminée par la plus petite carte gagnante du défenseur (c'est-à-dire le cas le pire). On considère que l'attaquant a beaucoup de cartes perdantes si le nombre total de cartes perdantes pour les quatre couleurs est supérieur ou égal à six.

La possession du petit

Elle modifie évidemment le plan de jeu. Si l'on a le petit, on peut choisir de le sauver dès que possible ou de l'amener au bout ; si l'on ne l'a pas on peut éventuellement le chasser.

On détermine la classe d'une main de cartes en évaluant la valeur de ces quatre critères. A partir de ces quatre attributs, on peut construire plusieurs graphes de classification différents. Cela dépend de l'ordre dans lequel on examine les quatre attributs. Nous avons suivi l'ordre dans lequel les règles de Bateleur considèrent ces attributs. La possession du petit doit être considérée en dernier car elle permet uniquement de choisir une variante du même plan de jeu ; elle n'est pas mentionnée dans les stratégies usuelles. Le nombre de cartes perdantes n'est pas considéré dans toutes les règles, on doit donc s'en soucier après la hauteur et la longueur d'atout. Pour choisir l'ordre entre ces deux derniers attributs, nous nous sommes conformés à celui choisi par l'expert.

Le graphe de classification donné à SYRCLAD-tarot est présenté de manière succincte sur la figure 1. La figure 2 illustre de manière plus détaillée une partie de ce graphe. On observera sur la figure 2 que les quatre attributs discriminants sont tous définis à la racine du graphe.

2.3 Les connaissances de reformulation

On a décrit au paragraphe précédant la manière de calculer la valeur des quatre attributs discriminants. Treize règles permettent de calculer ces valeurs. Elles font intervenir quelques attributs intermédiaires : le nombre d'atouts manquants par rapport aux atouts les plus forts et le nombre de cartes perdantes dans chacune des quatre couleurs.

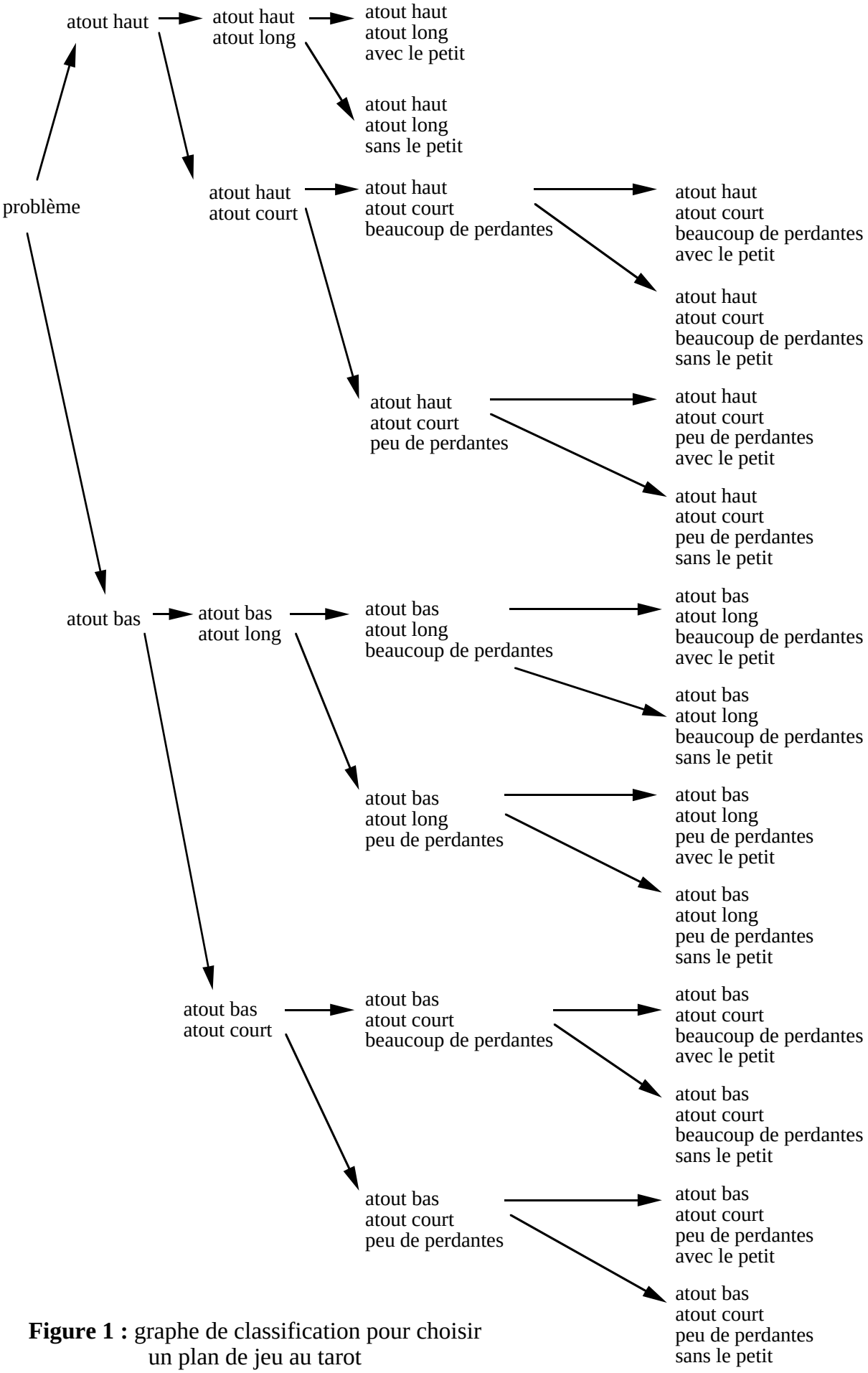


Figure 1 : graphe de classification pour choisir un plan de jeu au tarot

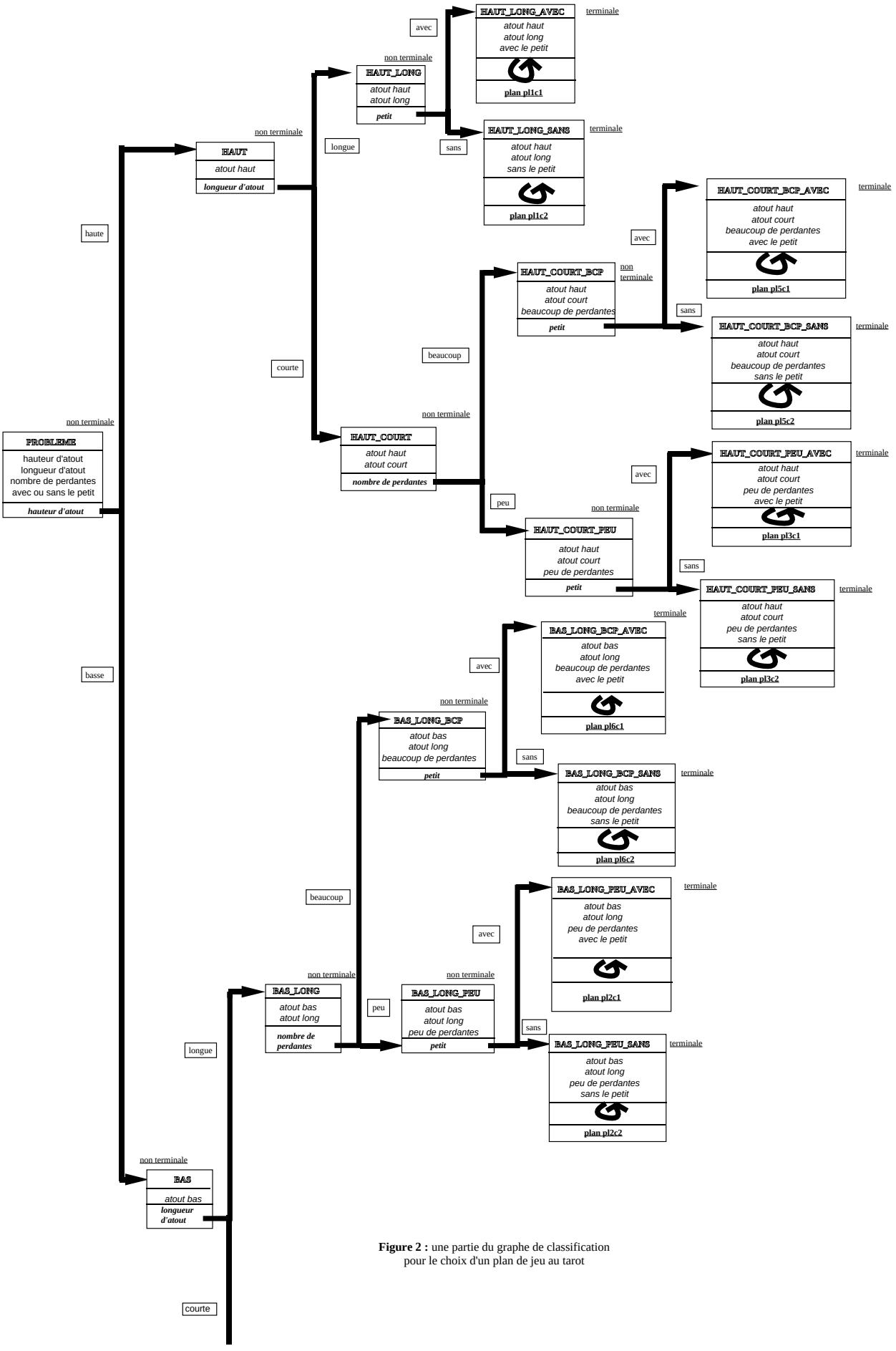


Figure 2 : une partie du graphe de classification
pour le choix d'un plan de jeu au tarot

2.4 Les modèles opérationnels

Le modèle opérationnel d'un problème de tarot est composé des attributs discriminants et de leurs valeurs qui ont été déterminées pendant le classement. Il n'y a pas d'attributs non discriminants car les méthodes de résolution ne s'adaptent pas aux problèmes. Il n'y a donc pas de différence entre la classe d'un problème et son modèle opérationnel.

Pour la partie p1 :

Contrat : prise
Écart : ♦ 1 ♥ 10 6 3 ♠ 10 7
Main de l'attaquant : ♣ R C 9 8 6 4 2
♦ R
♠ R
atout 21 20 19 17 10 5 4 1 excuse

Le modèle opérationnel est le suivant :

```
Hauteur d'atout : bas
Longueur d'atout : courte
Nombre de perdantes : beaucoup
Avec le petit
```

2.5 Les connaissances de résolution

L'expertise qui associe un plan de jeu à une classe est difficile. J.-M. Nigro a établi les règles de Bateleur en se basant sur les conseils des ouvrages sur le tarot, et en particulier [Jeannin-Naltet et Garrivet 83]. Cette expertise est décrite en annexe D de sa thèse, page 224. Nous reproduisons quelques-unes de ces règles en annexe.

Dans SYRCLAD-tarot, à chaque classe est associé un plan de jeu. Un plan est constitué d'une liste de buts donnés par ordre de priorité.

Par exemple :

Chasser le petit
Affranchir la longue
Jouer les perdantes
Épuiser les atouts adverses
Fermer le jeu

On a ainsi quatorze plans possibles qui sont associés à quatorze classes opérationnelles. Ces plans ne s'adaptent pas au problème posé. Le problème étant de choisir un plan de jeu pour une situation

donnée, ce sont des solutions plutôt que des méthodes de résolution qui sont associées aux classes opérationnelles.

Il y a dix buts pouvant apparaître dans les plans de jeu. Nous donnons ici la signification de certains d'entre eux:

Affranchir la longue : une longue est la couleur dans laquelle le joueur possède le plus de cartes. Affranchir la longue signifie jouer des cartes de cette couleur afin de ne garder que des cartes gagnantes dans cette couleur.

Fermer le jeu : lorsqu'en fin de jeu il ne reste au joueur que des cartes gagnantes, il peut aplatir toutes les cartes d'un coup pour montrer qu'il remporte tous les plis.

Amener le petit au bout : lorsque le petit est joué au dernier pli, le gagnant de ce pli bénéficie d'une prime de points.

Chasser le petit : essayer de prendre le 1 d'atout à l'adversaire.

Jouer les singletons et les doubletons : un joueur a un singleton (respectivement un doubleton) lorsqu'il ne possède qu'une carte (resp. deux) dans une couleur donnée.

3 Résultats

Partie p1 :

Contrat : prise
Écart : ♦ 1 ♥ 10 6 3 ♠ 10 7
Main de l'attaquant : ♣ R C 9 8 6 4 2
♦ R
♠ R
atout 21 20 19 17 10 5 4 1 exc

SYRCLAD-tarot classe le problème en le faisant descendre dans le graphe de classification, il applique des règles pour déterminer la valeur des critères de classification :

```
    p1
manquants hauteur2 longueur1 perdantes_carreau perdantes_cœur perdantes_trèfle
perdantes_pique perdantes1 petit1
```

L'attaquant n'a pas beaucoup d'atouts, ils ne sont pas très forts, il a beaucoup de cartes perdantes et il possède le petit.

Puis il affiche la classe qu'il a déterminée ainsi que le plan de jeu associé :

```
p1
Problème de classe c_bas_court_bcp_avec
Plan de jeu :
Sauver le petit dès que possible
Jouer les singletons et les doubletons
Affranchir la longue
Jouer la longue pour faire couper
Jouer les perdantes
Jouer atout
Fermer le jeu
```

Un exemple de main plus favorable pour l'attaquant :

Contrat : garde-contre
Main de l'attaquant : ♦ R 6
♠ R D 8
♥ V
atout 21 20 19 18 17 16 8 7 6 5 1 exc

Comme le contrat est une garde-contre, l'attaquant n'a pas eu le chien et n'a pas fait d'écart, il ne connaît donc pas les cartes du chien.

Le système déclenche des règles afin de classer le problème :

```
   p129_b
manquants hauteur1 longueur2 petit1
```

L'attaquant possède beaucoup d'atouts, dont beaucoup de très forts et il a le petit. Dans ce cas très favorable, il n'est même pas utile d'évaluer le nombre de perdantes. Le système conclut :

```
   p129
   Problème de classe c_haut_long_avec
   Plan de jeu :
   Affranchir la longue
   Jouer les perdantes
   Épuiser les atouts adverses
   Fermer le jeu
   Amener le petit au bout
```

Influence du contrat :

Le contrat joue sur la connaissance ou non des cartes de l'écart, il modifie donc l'évaluation du nombre de perdantes et par conséquent la classe du problème et le plan de jeu. Étudions le cas de l'exemple suivant :

Contrat : garde
Écart : ♦ D ♥ 7 4 ♠ 9 5 2
Main de l'attaquant : ♣ 9 8 5
♦ C V 7
♥ R D C
atout 20 19 18 16 15 10 7 6 1

Le fait que la dame de carreau soit dans l'écart implique que le valet de carreau n'est pas une carte perdante. Le système conclut donc qu'il y a peu de perdantes (5).

```
   pn11
manquants hauteur1 longueur1 perdantes_carreau perdantes_cœur perdantes_trèfle
perdantes_pique perdantes2 petit1
   pn11
   Problème de classe c_haut_court_peu_avec
   Plan de jeu :
   Sauver le petit dès que possible
   Affranchir la longue
```

```
Jouer la longue pour faire couper
Jouer les perdantes
Fermer le jeu
```

Considérons maintenant le cas d'un joueur qui possède la même main de cartes mais qui ne connaît pas les cartes du chien à cause du contrat :

Contrat : garde-sans
Main de l'attaquant : ♣ 9 8 5
♦ C V 7
♥ R D C
atout 20 19 18 16 15 10 7 6 1

Comme le système ne sait pas si la dame de carreau est dans le chien, il considère qu'elle est dans le jeu des adversaires, et que le valet de carreau est une carte perdante ; il y a donc beaucoup de cartes perdantes (6), et on conclut sur un plan de jeu différent :

```
     pn11_b
manquants hauteur1 longueur1 perdantes_carreau perdantes_cœur perdantes_trèfle
perdantes_pique perdantes1 petit1
     pn11_b
     Problème de classe c_haut_court_bcp_avec
     Plan de jeu :
     Sauver le petit dès que possible
     Affranchir la longue
     Jouer la longue pour faire couper
     Faire couper
     Jouer les perdantes
     Fermer le jeu
```

4 Conclusion

Bateleur [Nigro 95] possède une expertise de choix d'un plan de jeu en début d'une partie de tarot. L'application de SYRCLAD au tarot est basée sur l'étude de la base de règles de Bateleur. Cette base de règles utilise sans le dire une classification ; en effet, dans la partie condition des règles apparaissent trois ou quatre attributs, que l'on peut considérer comme des critères de classification, et en partie conclusion apparaît le plan de jeu choisi. L'architecture de SYRCLAD a permis d'expliciter une classification basée sur les attributs présents dans les règles. Le système SYRCLAD-tarot résout, en utilisant cette classification, les mêmes problèmes que Bateleur. L'avantage est que dans SYRCLAD-tarot, nous avons explicite une classification qui était cachée dans les règles de Bateleur. L'expertise utilisée est maintenant explicite et déclarative, et donc d'un accès plus facile.

SYRCLAD-tarot choisit un plan de jeu pour 29 situations différentes. Il y a 14 plans de jeu possibles. Ce domaine est moins riche que les dénombrements ou les problèmes additifs. En effet, il y a seulement deux types de modèles descriptifs possibles ; le graphe de classification est très symétrique, tous les attributs étant définis à la racine ; les règles de reformulation sont peu nombreuses et essentiellement calculatoires ; enfin, les plans ne s'adaptent pas au problème. La difficulté de ce domaine réside dans l'association d'un plan de jeu à une classe de problèmes. Cette expertise est donnée à SYRCLAD-tarot par l'expert.

Chapitre 7

Construire un graphe de classification : la thermodynamique

Nous présentons dans ce chapitre l'application de SYRCLAD au domaine de la thermodynamique, c'est-à-dire SYRCLAD-thermodynamique. Nous nous pencherons tout d'abord sur le domaine de la thermodynamique ; nous indiquerons pourquoi il était intéressant d'appliquer SYRCLAD à ce domaine. Nous décrirons ensuite Modélis [Tisseau 90], le système sur lequel nous nous sommes appuyés. Nous relaterons la manière dont nous avons construit un graphe de classification de problèmes de thermodynamique. Enfin nous récapitulerons les connaissances données à SYRCLAD-thermodynamique et nous donnerons des exemples de problèmes résolus par SYRCLAD-thermodynamique afin d'en évaluer les résultats.

Plan du chapitre

1 Pourquoi la thermodynamique ?

1.1 Le domaine de la thermodynamique

La théorie de la thermodynamique étudie des systèmes gazeux. Ces systèmes sont en équilibre à un instant donné ou en transformation entre deux instants. L'état d'un système à un instant est caractérisé par les valeurs de certains paramètres du système : la température, la pression, le volume, la masse, le nombre de moles... Une transformation peut être caractérisée par une loi sur l'évolution de ces paramètres. Par exemple, une transformation est isotherme si la température du système reste constante. Si le système est un gaz parfait, on sait que la relation PV = nRT reste vraie pendant toute la durée de la transformation. Certains paramètres ne dépendent pas du temps, comme la masse molaire ou la composition d'un gaz.

1.2 Pourquoi appliquer SYRCLAD à la thermodynamique ?

Comme en dénombrements, les étudiants éprouvent des difficultés à résoudre des exercices de thermodynamique après avoir suivi un cours théorique. En effet, les problèmes sont posés en termes concrets et on doit les modéliser avant de pouvoir appliquer les connaissances du cours. Cette phase de modélisation est difficile car elle nécessite des connaissances spécifiques, mais malheureusement elle n'est pas enseignée en tant que telle. C'est ce qui a conduit G. Tisseau à réaliser Modélis [Tisseau 90], un système qui modélise et résout des problèmes de thermodynamique en utilisant des connaissances explicites.

SYRCLAD est destiné à faciliter l'expression des connaissances dans les domaines où les problèmes sont concrets et doivent être modélisés pour être plus opérationnels ; nous avons donc pensé que la thermodynamique était un bon domaine d'application pour SYRCLAD. La question posée était : « peut-on construire une classification pour améliorer la modélisation et la résolution effectuées par Modélis ? ».

Il paraissait probable qu'une classification pouvait être dégagée car on pouvait remarquer des ressemblances dans les structures des résolutions de certains exercices. Dégager une classification des mains de cartes en début d'une partie de tarot à partir des règles de Bateleur n'a pas été difficile car d'une part les règles étaient peu nombreuses et d'autre part les prémisses de ces règles indiquaient les critères de classification et leurs conclusions donnaient les solutions. Par contre, les règles utilisées par Modélis pour résoudre des problèmes de thermodynamique sont nombreuses et compliquées. L'analyse de ces règles ne permet pas de dégager une classification immédiate.

Nous avons construit une classification en observant le fonctionnement dynamique d'un système expert utilisant les connaissances de résolution de Modélis. Cette classification permet de sélectionner parmi les connaissances de Modélis celles qui sont pertinentes pour un problème donné. De plus,

pour certaines classes de problèmes spécifiques, elle permet de produire un plan de résolution sans utiliser les connaissances de Modélis.

Comme nous avons construit une classification en observant les résolutions de Modélis, nous allons brièvement décrire son fonctionnement.

2 Modélis

Modélis traite une vingtaine d'exercices de thermodynamique rencontrés en classes préparatoires. Il se restreint essentiellement à l'étude des gaz parfaits. Il prend en entrée des exercices en langage naturel, qu'il analyse pour produire un modèle du problème puis il résout le problème. Nous présentons d'abord à travers des exemples la phase de modélisation, puis la phase de résolution.

2.1 Modéliser

Considérons l'exercice suivant : « Un réservoir fermé renferme de l'air à 35°C et sous une pression de 7 bars. Que devient la pression quand la température s'abaisse à 10°C ? »

Modélis utilise des connaissances d'analyse du langage naturel et des connaissances spécifiques au domaine de la thermodynamique pour produire un modèle du problème exprimé par un ensemble de propositions. Il s'agit dans cet exercice d'un système_C, gaz parfait dont on connaît la composition, qui subit une transformation isochore entre deux instants instant_F et instant_G :

```
gp(système_C). /* soit système_C un gaz parfait */
composition(système_C).    /* on connaît sa composition (air) */
trans_gp_fermé(système_C,instant_F,instant_G).    /*c'est un gaz parfait qui
subit une transformation fermée entre les instants instant_F et instant_G */
isochore(système_C,instant_F,instant_G).
   /* il est isochore entre ces deux instants */
température(système_C,instant_F).
   /* on connaît sa température à l'instant initial */
pression(système_C,instant_F). /* sa pression à l'instant initial */
température(système_C,instant_G). /* et sa température à l'instant final */
on_demande(pression(système_C,instant_G)).
   /* il faut trouver sa pression à l'instant final */
```

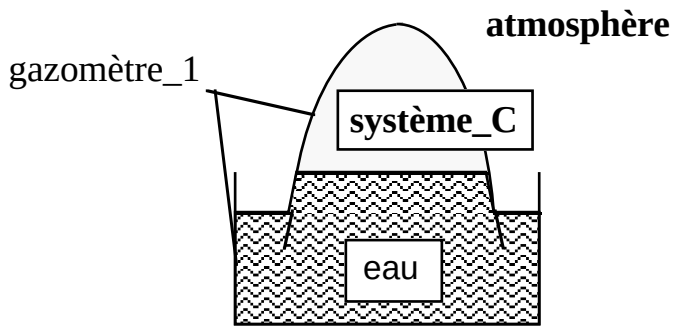
Voici un exercice simple de gazomètre : « Un gazomètre, dont la cloche, de 40 m de diamètre, a une masse de 315 t, renferme 35000 m^3 de gaz à 17 °C. La pression atmosphérique est de 740 mm Hg. On demande de déterminer :

1) la pression de distribution du gaz ;

2) la masse de gaz contenue dans le gazomètre.

La constante du gaz est r = 681 J/kg*deg. On néglige la poussée d'Archimède sur la partie immergée de la cloche. »

Le modèle produit par Modélis distingue deux systèmes gazeux : le gaz sous la cloche (système_C) et l'atmosphère. Le gazomètre gazomètre_1 les relie.




```
gp(système_C).    /* soit système_C un gaz parfait */
équi_gp(système_C,instant_D).
 /* il est en équillibre à l'instant instant_D */
température(système_C,instant_D). /* on connaît sa température */
volume(système_C,instant_D).  /* et son volume à cet instant */
r_molaire(système_C). /* on connaît la constante du gaz r */
pression(atmosphère,instant_D). /* la pression atmosphérique est connue */
gazomètre(gazomètre_1). /* soit gazomètre_1 un gazomètre */
masse(gazomètre_1). /* on connaît la masse de sa cloche */
diamètre(gazomètre_1). /* et son diamètre */
équi_gazomètre(gazomètre_1,système_C,atmosphère,instant_D). /* gazomètre_1
est en équilibre avec le gaz système_C et l'atmosphère à l'instant_D */
on_demande(pression(système_C,instant_D)).
on_demande(masse(système_C,instant_D)).
 /* il faut trouver la pression et la masse du gaz système_C */
```

Le lecteur aura remarqué que Modélis ne tient pas compte des valeurs numériques des grandeurs, mais se préoccupe uniquement de savoir si ces valeurs sont connues. Modélis produit en effet un plan de résolution qui indique ce qu'il faut calculer, dans quel ordre et avec quelles équations, mais n'effectue pas d'application numérique. C'est ce plan qui est demandé aux élèves avant l'application numérique.

Les modèles descriptifs que SYRCLAD-thermodynamique prendra en entrée sont les modèles produits Modélis, comme ceux que nous venons de présenter. Dans les autres domaines, les modèles descriptifs étaient très proches de l'énoncé en langue naturelle. Pour produire ces modèles descriptifs à partir de l'énoncé en langue naturelle, seules des connaissances d'analyse du langage naturel étaient nécessaires. En revanche, en thermodynamique, Modélis a également utilisé des connaissances spécifiques au domaine, et les modèles descriptifs ainsi produits sont plus éloignés de l'énoncé en langage naturel que dans les autres domaines. Modélis a effectué une première modélisation qui est très délicate dans le domaine de la thermodynamique. SYRCLAD-thermodynamique effectuera une deuxième modélisation, pour produire un modèle du problème plus opérationnel qui permettra une résolution plus efficace.

Voyons à présent comment Modélis résout un problème qu'il a modélisé.

2.2 Résoudre

Les lois physiques utilisées en thermodynamique se traduisent par des équations, par exemple PV = nRT. Dans Modélis, les équations sont représentées de manière qualitative, par des contraintes affirmant l'existence d'une relation entre des grandeurs.

Par exemple, si S est un gaz parfait en équilibre à l'instant I, la loi des gaz parfaits (PV = nRT) s'écrit :

```
rel([pression(S,I),volume(S,I),n(S,I),température(S,I)]).
```

A partir du modèle de l'exercice, Modélise utilise des connaissances de thermodynamique pour déterminer les lois qui s'appliquent. Ces connaissances sont exprimées par des règles "quand". Les règles "quand" regroupent les propriétés et les équations correspondant à une situation physique donnée. Par exemple dans le cas d'un gaz parfait (extrait de la règle) :

```
quand(gp(X)):- on_a([
fp(X), /* gaz parfait => fluide parfait */
rel([r_molaire(X),masse_molaire(X)],'r = R/M'),
rel([densité(X),r_molaire(X)],'densité = r(air)/r'),
si([diatomique(X)],
 alors([connu(cv(X),'diatomique'),connu(cp(X),'diatomique')])),
si([monoatomique(X)],
 alors([connu(cv(X),'monoatomique'),connu(cp(X),'monoatomique')]))]).
```

En effet, quand un système est un gaz parfait, on sait que c'est un fluide parfait, et certaines lois s'appliquent, comme celle reliant la masse molaire et la constante du gaz. On peut trouver dans les règles "quand" des règles si...alors qui représentent des connaissances applicables dans la situation décrite par la règle "quand". Toujours dans le cas d'un gaz parfait, si l'on sait de plus qu'il est diatomique, on connaîtra ses coefficients calorimétriques cp et cv. A chaque relation "rel" est associé un commentaire (par exemple 'r : R/M') qui permet d'expliciter dans le plan de résolution la loi utilisée.

Le moteur d'inférence de Modélis déclenche les règles "quand" déclenchables et obtient un ensemble de relations "rel" entre des grandeurs. Le modèle du problème donne un ensemble de grandeurs connues. La résolution consiste à propager les valeurs connues du modèle dans les équations représentées par les relations "rel" jusqu'à déduire la (ou les) valeur(s) demandée(s). Grâce à l'origine de chaque fait, on peut remonter de la conclusion jusqu'aux données pour produire le plan de résolution de l'exercice.

Prenons l'exercice suivant : « Une bouteille de gaz comprimé, de 14 l de capacité, renferme de l'oxygène sous une pression de 150 bars à une température de 15 °C. Quelle est la masse de gaz contenue dans la bouteille ?

On donne la constante du gaz r(O_2) = 260 J/kg*deg. »

Le modèle produit par Modélis est le suivant :

```
gp(système_B).
équi_gp(système_B,instant_C).
température(système_B,instant_C).
pression(système_B,instant_C).
volume(système_B,instant_C).
r(système_B).
on_demande(masse(système_B,instant_C)).
```

Deux règles "quand" (entre autres) se déclenchent :

- quand gp(système_B) asserte (entre autres) la relation :

```
rel([r_molaire(système_B),masse_molaire(système_B)],'r = R/M').
```

- quand équi_gp(système_B,instant_C) asserte (entre autres) la relation :

```
rel([pression(système_B,instant_C),volume(système_B,instant_C),n(système_B,
instant_C),température(système_B,instant_C)],'PV = nRT).
```

Or équi_gp(système_B,instant_C) implique équi_fp(système_B,instant_C) (fluide parfait en équilibre) qui implique équi_sq(système_B,instant_C) (système quelconque en équilibre).

La règle quand équi_sq(système_B,instant_C) se déclenche donc et asserte la relation :

```
rel([n(système_B,instant_C),masse_molaire(système_B),masse(système_B,
instant_C),],'n = m/M')
```

Comme on connaît r(système_B), la première relation nous donne masse_molaire(système_B).

Comme on connaît température(système_B, instant_C), pression(système_B, instant_C) et volume(système_B, instant_C), la deuxième relation nous donne n(système_B, instant_C).

Comme on connaît masse_molaire(système_B) et n(système_B, instant_C), la troisième relation nous donne masse(système_B,instant_C), qui était la grandeur demandée.

Nous avons reproduit la base de connaissances que Modélis utilise pour résoudre et nous avons réalisé un moteur équivalent à celui de Modélis pour utiliser les règles "quand" et résoudre les relations "rel". Appelons THERMOS le résolveur ainsi construit. THERMOS prend en entrée un modèle de problème produit par Modélis et utilise les règles quand pour résoudre le problème. C'est l'observation du fonctionnement de THERMOS qui va nous aider à construire une classification des problèmes de thermodynamique.

3 Construction d'un graphe de classification

Une classification des problèmes d'un domaine est constituée d'une part de critères de classification (attributs discriminants et leurs valeurs discriminantes) et d'autre part d'un ensemble de méthodes de résolutions liées à ces critères de classification. Les critères de classification permettent de construire une nouvelle représentation du problème tandis que les méthodes de résolution sont les connaissances procédurales attachées à ces nouveaux modèles de problèmes. M. Chi a observé dans

[Chi et al 81] que les classes de problèmes des experts correspondent aux solutions à utiliser, alors qu'une construction de classes par auto-observation amène à des classes basées sur les critères. Elle a aussi remarqué que les critères utilisés par les experts n'étaient pas ceux utilisés par les novices. Les novices reconnaissent certains critères importants mais ne savent pas y associer une méthode de résolution.

Pour construire la classification des problèmes d'un domaine, il faut trouver les différentes méthodes de résolution possibles, ainsi que les attributs des problèmes définissant les critères de classification, et il faut surtout trouver comment ils sont liés.

3.1 Une première classification

Pour le domaine de la thermodynamique, considérons le cas d'un apprenti-expert : "je". "Je" veut construire une classification des exercices de thermodynamique et espère que cela lui permettra de devenir un expert confirmé. Pour cela, "je" observe le fonctionnement de THERMOS sur le corpus de problèmes de Modélis. Dans un premier temps, il essaie de distinguer les méthodes de résolutions employées.

3.1.1 Méthodes de résolution

> Dans un premier temps, j'ai voulu utiliser UBL [Pintado 94] pour découvrir des méthodes de résolution. En effet, UBL possède des capacités d'apprentissage : il découvre des "chunk", qui sont des nouveaux théorèmes regroupant l'enchaînement de plusieurs autres et également des méthodes, qui proposent des sous-buts pertinents pour un but donné. M. Pintado a donc donné à UBL la base de règles "quand" de Modélis, mais les résultats n'ont pas été très intéressants pour la construction d'une classification. En effet, UBL reste au niveau des lois de la thermodynamique. Par exemple, il propose : « si l'on connaît P et V, et que l'on veut calculer T, un sous-but intéressant est n », qui est une (bonne) interprétation de la loi des gaz parfaits PV = nRT. En fait, il me semble que ces résultats sont décevants (pour l'utilisation que je souhaitais en faire) parce que nous n'avons pas donné à UBL de connaissances sémantiques sur la thermodynamique, UBL a donc du mal à passer au niveau méta.

En observant la résolution d'une vingtaine d'exercices de thermodynamique par THERMOS, j'ai remarqué que certaines règles étaient toujours utilisées ensemble, le déclenchement de l'une d'entre elles entraînant le déclenchement d'une autre qui à son tour provoque le déclenchement d'une troisième, etc. En utilisant des métaconnaissances sémantiques liées à la thermodynamique sur le contenu des règles (de quoi parlent-elles ?), j'ai pu distinguer cinq modules de connaissances.

Un premier ensemble de règles contient toutes les connaissances relatives à un gaz parfait en équilibre à un instant donné. Un deuxième module peut s'y associer, celui qui concerne la transformation d'un système entre deux instants. Toutes les lois ayant trait à la conservation d'une grandeur entre deux instants sont rassemblées dans le troisième module. Les deux derniers modules concernent les appareils reliant plusieurs systèmes gazeux ; l'un concerne les systèmes reliés entre eux par un système de surface, l'autre est spécialisé dans les gazomètres. J'ai aussi remarqué pour

des exercices très simples que l'on pouvait les résoudre en utilisant uniquement des connaissances sur des grandeurs intensives, ou uniquement extensives[1].

J'ai donc choisi d'étiqueter les règles "quand" ainsi que les règles si...alors qu'elles peuvent contenir par le module auquel elles appartiennent. Une règle si...alors peut avoir une étiquette différente de celle de la règle "quand" qui la contient.

Prenons par exemple un extrait de la règle "quand" qui traite de la transformation d'un système quelconque fermé S entre deux instants I et F :

```
quand(trans_sq_fermé(S,I,F)):- on_a([
trans_sq(S,I,F),
 /* S est un système quelconque un transformation entre I et F */
connu(variation(n,S,I,F),'Dn = 0'),
 /* la variation du nombre de moles de S entre I et F est nulle */
équi_sq(S,I), /* S est un système quelconque en équilibre à l'instant I */
équi_sq(S,F), /* S est un système quelconque en équilibre à l'instant F */
si([isochore(S,I,F)],alors([connu(inv1(volume,S,I,F),
 'isochore:V=constante')])),
 /* si de plus S est isochore entre I et F, son volume est constant */
si([adiabatique(S,I,F)],alors([connu(chaleur(S,I,F),
 'adiabatique:Q=0')]))]).
 /* si de plus S est adiabatique entre I et F, la chaleur est nulle */
```

Cette règle traite de la transformation d'un système entre deux instants, elle appartient donc au deuxième module de connaissances. Elle contient deux règles si...alors qui s'appliquent dans le cadre de la règle "quand" mais qui traitent de la conservation d'une grandeur. Les deux règles si...alors appartiennent donc au troisième module de connaissances.

Les règles "quand" utilisées par THERMOS sont les règles "quand" de Modélis, les règles si..alors restent incluses dans les règles "quand". Chacune de ces règles est étiquetée par le module de connaissances auquel elle appartient. Sans avoir modifié les règles de Modélis, on peut désormais parler pour THERMOS de *l'ensemble des règles relatives à* la conservation d'une grandeur par exemple.

Notre recherche des différentes méthodes de résolution possibles nous a amené à distinguer différents modules de connaissances. Pour cela, nous avons utilisé d'une part l'observation de résolutions d'exercices et d'autre part des métaconnaissances sémantiques sur les règles utilisées.

On connaît maintenant les modules de connaissances que l'on peut utiliser, il faut déterminer les critères importants pour la classification.

---

[1] Une grandeur g est extensive si étant donné deux systèmes S1 et S2, g(S1 U S2) = g(S1) + g(S2), c'est le cas du volume, du nombre de moles ; une grandeur g est intensive si elle n'est pas extensive, c'est le cas de la pression, de la température ou de la masse volumique.

3.1.2 Critères de la classification

Un expert de la thermodynamique (G. Tisseau) nous a indiqué que l'on évalue la difficulté d'un problème de thermodynamique en fonction du nombre de systèmes et du nombre d'instants présents dans l'exercice ; en effet, il est plus facile de raisonner sur un système à un instant donné que sur plusieurs systèmes liés qui évoluent dans le temps. Nous en avons conclu que le nombre de systèmes et le nombre d'instants sont deux critères du problème qui peuvent être intéressants. Étant donné les modules de connaissances que nous avons distingués, certains autres critères peuvent se révéler intéressants : est-il fait mention dans l'énoncé d'une transformation qui conserve une grandeur (isochore, isotherme, isobare, adiabatique) ; est-il fait mention d'un gazomètre, ou de systèmes reliés par un système de surface ? On peut aussi repérer les exercices où il n'est question dans l'énoncé que de grandeurs intensives (respectivement extensives).

A présent que nous avons distingué d'une part des méthodes de résolutions et d'autre part des critères de classification qui semblent adaptés à ces méthodes de résolution, il faut relier les critères aux méthodes de résolutions de manière à construire un graphe de classification.

3.1.3 Relier les critères et les méthodes de résolution

Pour relier les critères de classification aux méthodes de résolution, j'ai à nouveau utilisé l'observation des exercices résolus par THERMOS, mais aussi mes connaissances sur le domaine, le tout me permettant de formuler des règles dont voici quelques exemples :

- on ne peut pas parler de transformation d'un système s'il n'y a qu'un instant,

- on peut parler de conservation d'une grandeur uniquement s'il y a deux instants,

- on ne peut pas parler de gazomètre ou de système de surface s'il n'y a qu'un système,

- quand il y a plusieurs systèmes, ils peuvent être soit l'un contenu dans l'autre (ambiance), soit reliés par un système de surface, soit composés pour former un gazomètre.

Ces règles (de bon sens thermodynamique) m'ont aidé à relier les critères de classification aux méthodes de résolutions pour construire le graphe de classification présenté sur la figure 1. Le premier attribut discriminant est le nombre de systèmes, le deuxième le nombre d'instants. Pour les exercices les plus simples (un système, un instant), on distingue ceux qui ne font intervenir que des grandeurs intensives ou que des grandeurs extensives. Quand il y a deux systèmes et un instant, il peut s'agir d'un gazomètre en équilibre. Lorsqu'il y a deux systèmes et deux instants, il y a plusieurs manières de lier les deux systèmes. Lorsqu'il y a deux instants, on recherche systématiquement (qu'il y ait un ou deux systèmes) la conservation d'une grandeur.

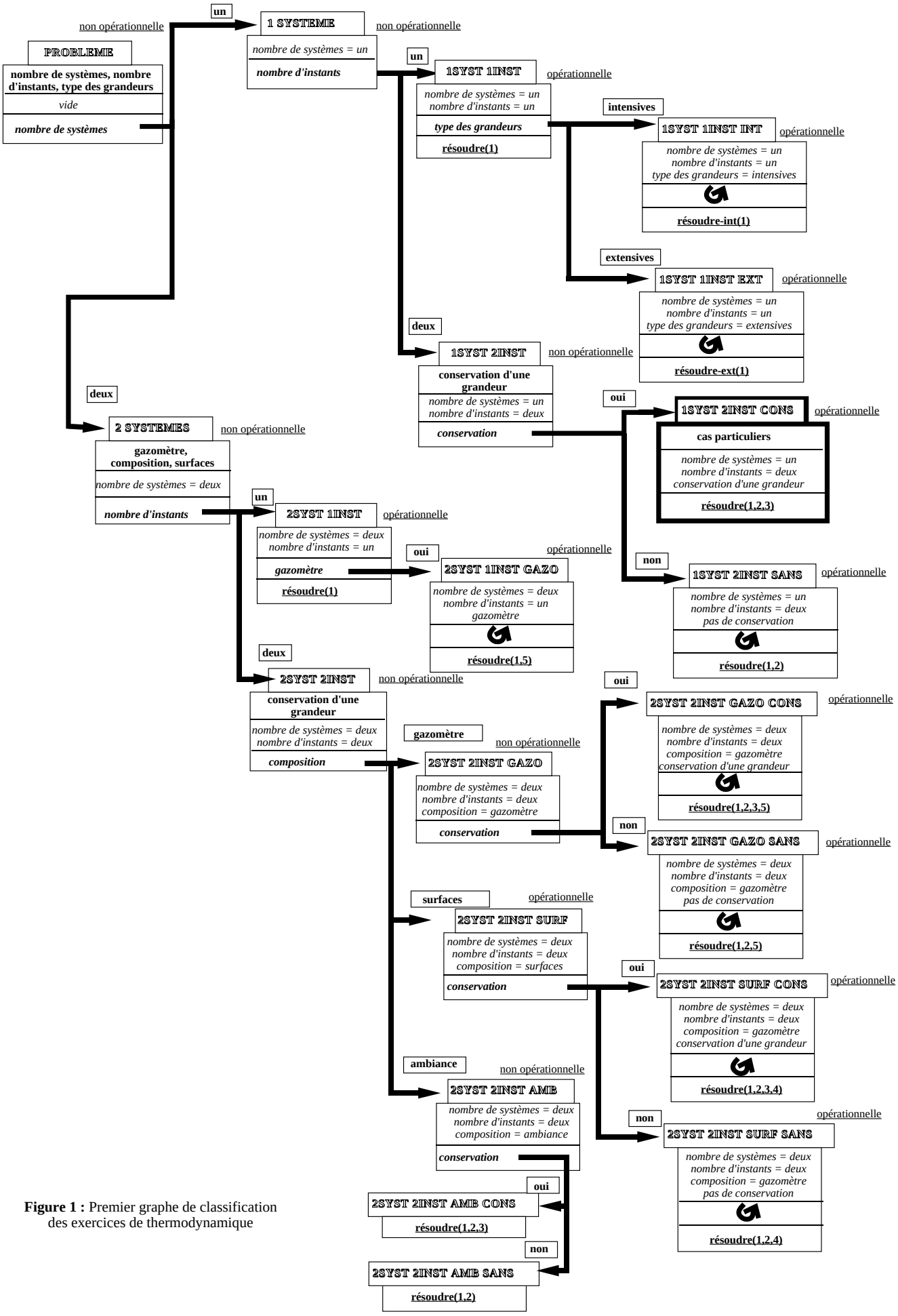


Figure 1 : Premier graphe de classification des exercices de thermodynamique

Les méthodes de résolutions associées aux classes opérationnelles appellent THERMOS avec les modules de connaissances pertinents. Par exemple résoudre(1,2,3) signifie : utiliser THERMOS avec les trois premiers modules de connaissances.

Module 1 : connaissances relatives à un gaz parfait en équilibre à un instant donné
Module 2 : connaissances relatives à la transformation d'un système entre deux instants
Module 3 : lois ayant trait à la conservation d'une grandeur entre deux instants
Module 4 : lois ayant trait aux systèmes de surface
Module 5 : lois ayant trait aux gazomètres
Module 6 : lois ayant trait aux mélanges de gaz

La méthode de résolution peut de plus préciser à THERMOS de résoudre des relations entre grandeurs correspondant aux équations physiques (les "rel") en essayant de conclure uniquement sur des grandeurs intensives : résoudre_int(1) (respectivement extensives : résoudre_ext(1)).

Nous reviendrons au paragraphe 4 sur le graphe de classification complet que nous avons fourni à SYRCLAD-thermodynamique.

3.2 Amélioration de la première classification

En utilisant le graphe de classification présenté en figure 1 sur les exercices du corpus de Modélis, j'ai remarqué que les exercices étaient bien répartis entre les diverses classes opérationnelles, sauf pour la classe un_système-deux_instants-conservation qui regroupait un grand nombre d'exercices. Ce déséquilibre m'a amené à penser qu'il fallait peut-être ajouter des sous-classes à cette classe, de manière à spécialiser les classes de problèmes et les méthodes de résolutions.

3.2.1 Méthodes de résolution

En observant comment THERMOS résout les exercices appartenant à la classe un_système-deux_instants-conservation, j'ai remarqué dans les plans produits des séquences de raisonnement qui revenaient dans plusieurs problèmes.

Prenons par exemple l'exercice suivant : « Un réservoir fermé renferme de l'air à 35°C et sous une pression de 7 bars. Que devient la pression quand la température s'abaisse à 10°C ? »

Le plan de résolution de THERMOS (traduit en français) est le suivant :
On a un système fermé (donc n est constant), et le volume est constant pendant la transformation. Comme PV = nRT, il existe une fonction de la pression et de la température constante pendant la transformation. Comme on connaît la pression et la température à l'instant initial, on connaît la valeur de cette fonction. Comme d'autre part on connaît la température à l'instant final, on en déduit la pression à l'instant final.

Ce raisonnement est souvent utilisé par THERMOS pour les problèmes de transformation d'un système avec une variable invariante. J'ai essayé de distinguer plusieurs variantes de ce raisonnement, et de les généraliser pour obtenir plusieurs plans de résolution type appliqués chacun à

plusieurs exercices. Ces plans-type seront présentés au paragraphe 4 quand nous reprendrons le graphe de classification complet.

3.2.2 Critères de classification

Après avoir distingué plusieurs plans-type, j'ai cherché les critères correspondant à chacun de ces plans. J'ai donc cherché les données communes aux modèles des exercices résolus avec un plan, en particulier les hypothèses qui permettaient de faire le raisonnement présenté dans le plan.

J'ai pu ainsi définir des cas particuliers de problèmes de la classe un_système-deux_instants-conservation. Pour l'exercice ci-dessus, nous sommes dans le cas particulier où il y a parmi les variables P,V,T une variable invariante (V). Les deux autres variables (P et T) sont connues à l'instant initial, l'une d'entre elles (T) est connue à l'instant final, et on demande la valeur de l'autre (P). Dans ce cas, un raisonnement comme celui que nous avons tenu en 3.2.1 est efficace.

3.2.3 Enrichir le graphe de classification

En utilisant les plans-type et les critères définissant des cas particuliers, nous avons enrichi le graphe de classification en ajoutant des sous-classes à la classe un_système-deux_instants-conservation (cf. figure 2).

Les graphes de classification sont déclaratifs, et on peut ajouter des classes facilement. Les problèmes où on mélange deux gaz ne sont pas traités par Modélis. Pour que SYRCLAD-thermodynamique puisse les traiter, nous avons ajouté un fils à la classe 2systèmes-2instants (cf. figure 1). En effet, un mélange de deux gaz est un nouveau type de composition de systèmes. Nous avons également ajouté les connaissances liées aux mélanges de gaz à la base de connaissances de THERMOS. Ces connaissances spécifiques forment un nouveau module de règles "quand". Il sera utilisé pour résoudre les problèmes de classe 2systèmes-2instants-mélange

3.3 Conclusion

On a vu que pour construire un graphe de classification, on utilise beaucoup l'observation des résolutions de problèmes, ainsi que des connaissances sur le domaine pour tirer un enseignement de cette observation. Un système qui construit des classifications de problèmes devra donc posséder de grandes capacités d'observation de traces de résolution, ainsi que des métaconnaissances sémantiques sur les objets du domaine et sur les lois utilisées pour la résolution.

Nous allons à présent revenir sur les connaissances données à SYRCLAD-thermodynamique, et en particulier sur le graphe de classification construit.

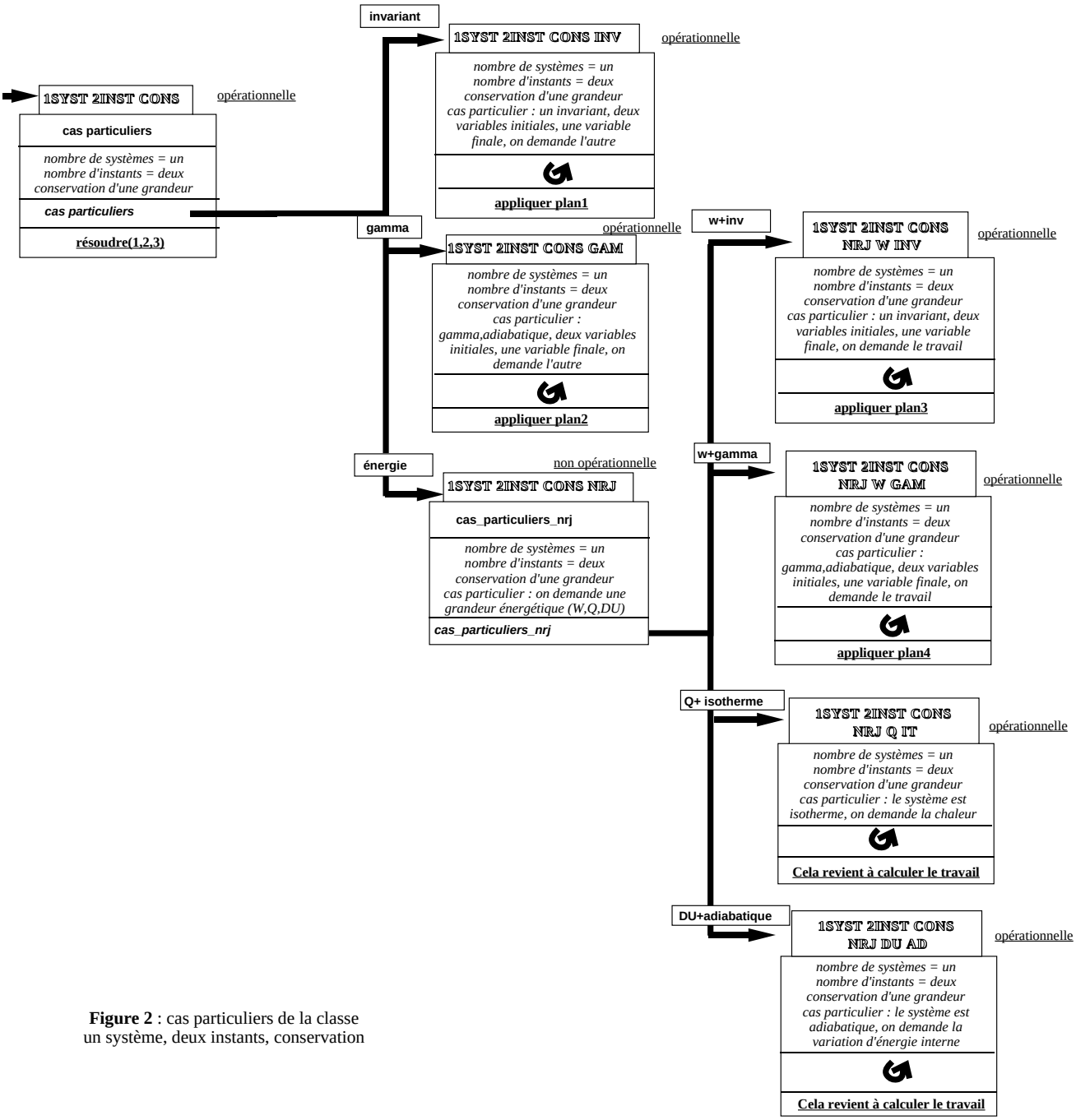


Figure 2 : cas particuliers de la classe
un système, deux instants, conservation

4 Les bases de connaissances de SYRCLAD-thermodynamique

4.1 Le langage pour exprimer les modèles descriptifs

On s'intéresse aux exercices de thermodynamique rencontrés en classes préparatoires en se restreignant essentiellement à l'étude des gaz parfaits.

SYRCLAD-thermodynamique prend en entrée les exercices tels que Modélis les a produits après l'analyse de l'énoncé en langage naturel et la modélisation. Une première étape de modélisation a donc déjà eu lieu. SYRCLAD-thermodynamique effectuera une deuxième modélisation, pour produire un modèle du problème plus opérationnel qui permettra une résolution plus efficace.

Le langage de la logique du premier ordre dans lequel sont exprimés les modèles descriptifs est donc défini par Modélis. Quatre-vingt-cinq prédicats peuvent être présents dans les modèles descriptifs. Pour chaque problème, il y un ou des systèmes et un ou des instants. Beaucoup de prédicats définissent les grandeurs connues :

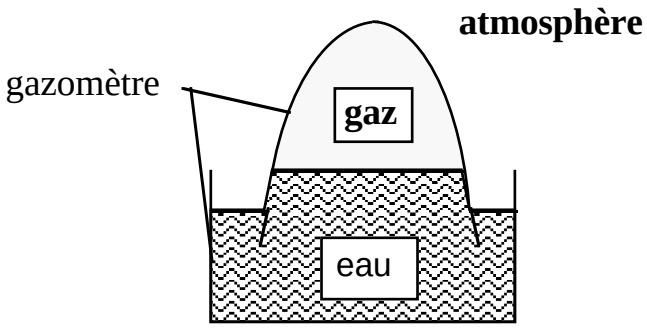
```
masse(S,I), pression(S,I), travail(S,I,F), gamma(S), composition(S),
hauteur_sous_cloche(Gazomètre,I)
```

Certains prédicats définissent des propriétés :

```
adiabatique(Sys,I,F), diatomique(Sys), équi_fp(Sys,I), gazomètre(S)
```

D'autres expriment des relations entre systèmes :

```
équi_gazomètre(gazomètre,gaz,atmosphère,instant)
```



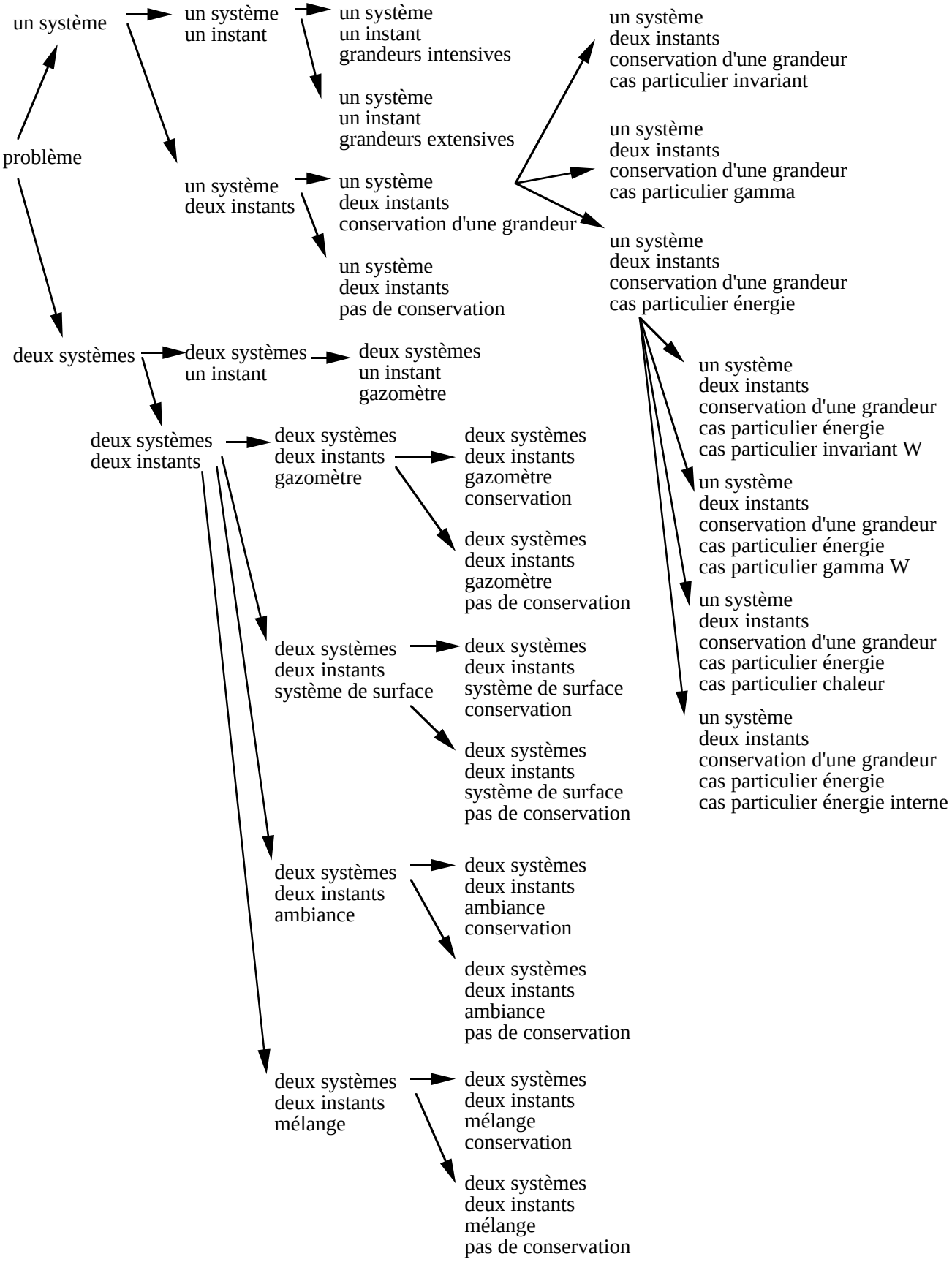


Figure 3 : graphe complet des problèmes de thermodynamique

4.2 Le graphe de classification

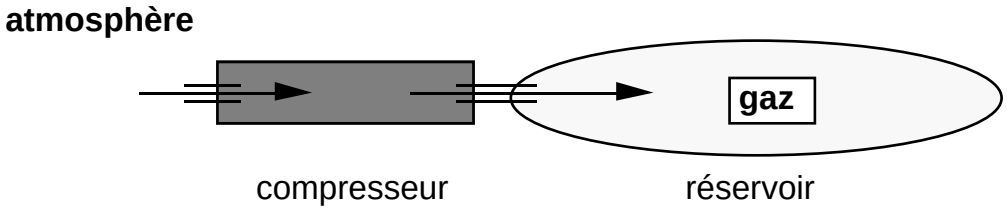
Nous avons décrit dans le paragraphe 3 la manière dont nous avons construit un graphe de classification de certains exercices de thermodynamique. Ce graphe est présenté en entier mais de manière succincte sur la figure 3 ; il est donné en entier et en Prolog dans les annexes.

Le premier attribut discriminant est le nombre de systèmes, le deuxième le nombre d'instants. Pour les exercices les plus simples (un système, un instant), on distingue ceux qui ne font intervenir que des grandeurs intensives ou que des grandeurs extensives. Quand il y a deux systèmes et un instant, il peut s'agir d'un gazomètre en équilibre. Lorsqu'il y a deux systèmes et deux instants, il y a plusieurs manières de lier les deux systèmes. Lorsqu'il y a deux instants, on recherche systématiquement (qu'il y ait un ou deux systèmes) la conservation d'une grandeur.

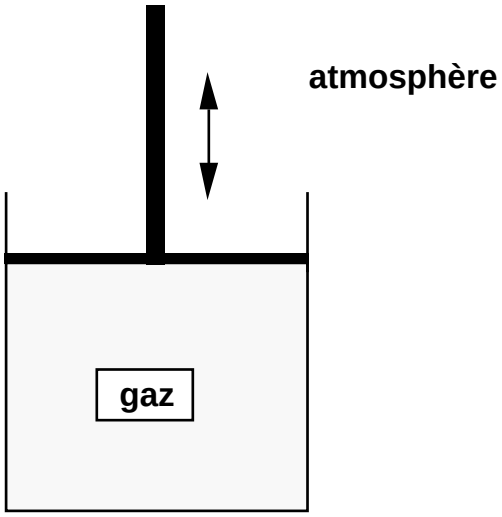
Détaillons maintenant les différents types de liens entre deux systèmes.

Un système gazeux est lié à une atmosphère par l'intermédiaire d'un gazomètre.

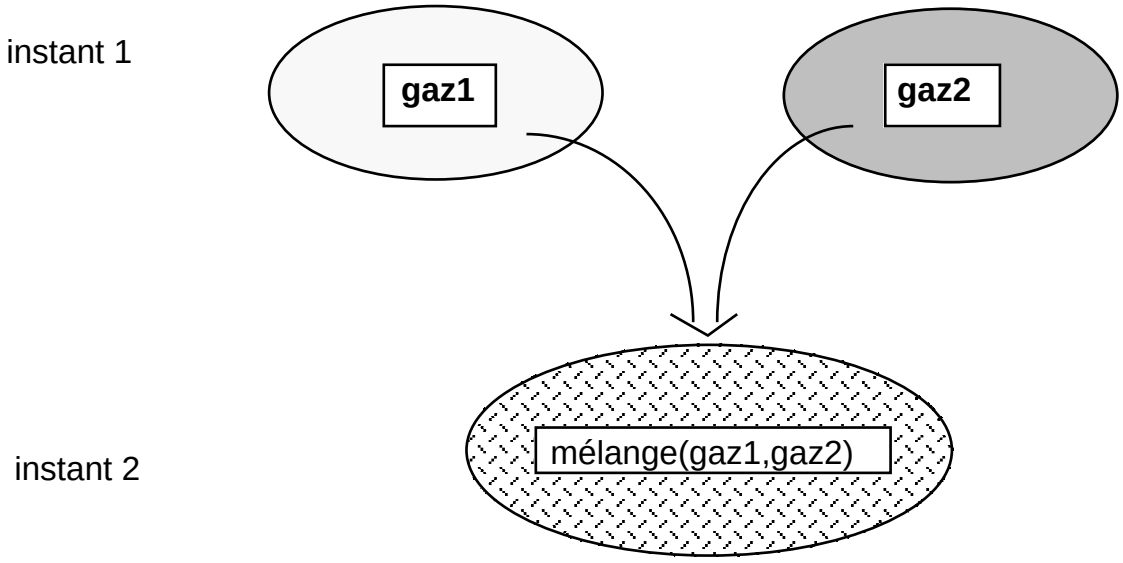
Deux systèmes peuvent être reliés par des systèmes de surface :



Un système peut évoluer dans l'ambiance de l'autre :



On peut aussi mélanger deux gaz :



Détaillons à présent les cas particuliers d'un système en transformation avec conservation d'une grandeur :

Cas particulier invariant : parmi les grandeurs P,T,V, l'une (par exemple T) est invariante, les deux autres (V et P) sont connues à l'instant initial. Parmi ces deux, l'une (par exemple P) est de plus connue à l'instant final, et on demande de calculer l'autre (V).

Cas particulier gamma : la transformation est adiabatique, et on connaît γ[1]. Parmi les grandeurs P,T,V, deux (p. ex. V et P) sont connues à l'instant initial. Parmi ces deux, l'une (p. ex. P) est de plus connue à l'instant final, et on demande de calculer l'autre (V).

Les autres cas particuliers sont des problèmes où l'on demande la valeur d'une grandeur énergétique : le travail (W), la chaleur (Q) ou la variation d'énergie interne (DU).

Cas particulier invariant W : parmi les grandeurs P,T,V, l'une (par exemple T) est invariante, les deux autres (V et P) sont connues à l'instant initial. Parmi ces deux, l'une (p. ex. P) est connue à l'instant final, et on demande de calculer le travail reçu par le système.

Cas particulier gamma W : la transformation est adiabatique, et on connaît γ. Parmi les grandeurs P,T,V, deux (p. ex. V et P) sont connues à l'instant initial. Parmi ces deux, l'une (p. ex. P) est connue à l'instant final, et on demande de calculer le travail reçu par le système.

Cas particulier chaleur : la transformation est isotherme et on demande de calculer la chaleur reçue par le système.

---

[1] gamma = c_p/c_v

Cas particulier énergie interne : la transformation est adiabatique et on demande de calculer la variation de l'énergie interne du système.

4.3 Les connaissances de reformulation

Vingt-sept règles permettent de déterminer la valeur des attributs discriminants à partir des faits contenus dans le modèle descriptif.

Par exemple, pour déterminer s'il y ou non conservation d'une grandeur, un ensemble de règles cherche dans le modèle descriptif un fait sur la conservation d'une grandeur : la transformation est-elle isochore, isotherme, isobare ou adiabatique ? Est-il précisé qu'une grandeur est conservée, que sa valeur est constante ?

Pour déterminer comment deux systèmes sont liés, un ensemble de règles cherche dans le modèle descriptif une construction représentant un système de surface, la présence d'un gazomètre, ou un mélange de gaz.

Un ensemble de règles permet de reconnaître les cas particuliers définis au paragraphe précédent.

On trouvera dans les annexes quelques-unes de ces règles en Prolog.

4.4 Les modèles opérationnels

Il y a pour le domaine de la thermodynamique deux types très différents de modèles opérationnels : ceux construits dans les classes particulières qui débouchent sur un plan produit par SYRCLAD-thermodynamique (sans appel à THERMOS), et ceux correspondant aux classes qui appellent THERMOS avec certains modules de connaissances.

4.4.1 Les cas particuliers

Pour les classes spécialisant la classe un_système-deux_instants-conservation, le classement est assez précis pour induire la construction d'un modèle opérationnel qui se suffit à lui-même. Ce modèle opérationnel contient les valeurs des attributs discriminants calculées pendant le classement.

Par exemple pour la classe cas_particulier_invariant :

```
Un système
Deux instants
Conservation d'une grandeur
Une variable invariante, deux variables initiales, une variable finale, on
demande l'autre[1].
```

[1] Plus précisément : parmi les grandeurs P,T,V, l'une (par exemple T) est invariante, les deux autres (V et P) sont connues à l'instant initial. Parmi ces deux, l'une (par exemple P) est de plus connue à l'instant final, et on demande de calculer l'autre (V).

Le modèle opérationnel est complété par un attribut non discriminant qui permet d'instancier le problème traité par rapport à la classe.

Par exemple pour le problème suivant : « Un réservoir fermé renferme de l'air à 35°C et sous une pression de 7 bars. Que devient la pression quand la température s'abaisse à 10°C ? », cet attribut précise :

```
Une variable invariante : V, deux variables initiales : T et P, une
variable finale : T, on demande l'autre : P.
```

Comme dans les autres domaines, ces modèles opérationnels se suffisent à eux-mêmes, il s'agit d'une autre représentation du problème initial.

4.4.2 Des problèmes enrichis

Les autres classes ne sont pas aussi spécifiques que les cas particuliers, elles déboucheront sur un appel à THERMOS. Les modèles opérationnels correspondant à ces classes sont composés uniquement des valeurs des attributs discriminants calculées pendant le classement. Ils ne contiennent pas d'attributs non-discriminants habituellement utilisés pour instancier le problème par rapport à la classe.

Par exemple :

```
Deux systèmes
Deux instants
Composition = mélange
Pas de conservation
```

Ces valeurs d'attributs sont utiles, nous avons vu qu'elles nous permettront de sélectionner les connaissances à utiliser. Mais contrairement aux autres domaines d'application et aux modèles du paragraphe précédent, ce modèle est insuffisant, il ne constitue pas à lui seul un modèle de problème. Il doit être complété pour la résolution par les faits du modèle descriptif. Pour ces classes, le modèle opérationnel est en fait composé d'une part des valeurs des attributs discriminants et d'autre part des faits du modèle descriptif. Le modèle opérationnel ne remplace pas le modèle descriptif, il s'agit d'un modèle descriptif enrichi. Nous retrouvons ici l'idée utilisée par J.-M. Bazin dans GEOMUS [Bazin 93].

4.5 Les méthodes de résolution

Nous avons vu qu'il y a deux types de modèles opérationnels correspondant à des classes qui débouchent sur deux types de méthodes de résolution. Nous présentons d'abord les méthodes de résolutions associées aux cas particuliers, puis les méthodes de résolutions associées aux autres classes moins spécifiques.

4.5.1 Produire un plan de résolution

Nous avons vu que les classes de la branche un_système-deux_instants-conservation sont très spécifiques et que l'on a associé à chacune d'entre elles un plan de résolution type.

Par exemple, pour la classe cas_particulier_invariant : « il y a parmi P,T et V une variable invariante, deux variables initiales, une variable finale et on demande l'autre » , le plan type est le suivant :

```
S est un système fermé (n=cte), et la variable Var_inv est constante entre
I et F, donc il existe une fonction f(Var1,Var2) constante entre I et F.
Comme on connaît Var1(S,I) et Var2(S,I), on connaît la valeur de cette
fonction. Comme d'autre part on connaît Var(S,F), on en déduit Vari(S,F).
```

où S est le système en transformation entre l'instant I et l'instant F, Var_inv est la variable invariante pendant la transformation parmi P,T et V. Var1 et Var2 sont les deux autres grandeurs. Vari est la grandeur qu'il faut calculer (soit Var1, soit Var2), Var est l'autre grandeur (soit Var2, soit Var1).

On a vu en 4.4.1 que le problème suivant : « Un réservoir fermé renferme de l'air à 35°C et sous une pression de 7 bars. Que devient la pression quand la température s'abaisse à 10°C ? » est de cette classe. Modélis considère qu'il s'agit d'un système système_C (l'air) en transformation entre deux instants : instant_F et instant_G.

Le modèle opérationnel est instancié par rapport à la classe grâce à l'attribut non-dsicriminant qui précise :

```
Une variable invariante : V, deux variables initiales : T et P, une
variable finale : T, on demande l'autre : P.
```

SYRCLAD-thermodynamique adapte le plan-type ci-dessus au problème :

```
système_C est un système fermé (n=cte), et la variable volume est constante
entre instant_F et instant_G, donc il existe une fonction f(pression,
température) constante entre instant_F et instant_G. Comme on connaît
pression(système_C, instant_F) et température(système_C, instant_F), on
connaît la valeur de cette fonction. Comme d'autre part on connaît
température(système_C, instant_G), on en déduit pression(système_C,
instant_G).
```

Parmi les six classes opérationnelles de la branche un_système-deux_instants-conservation (voir figure 2 et fin du paragraphe 4.2), quatre classes ont des méthodes de résolution qui consistent à appliquer un plan de résolution type. Nous avons vu celui qui est associé à la classe cas_particulier_invariant. Pour chacune des trois autres classes, il existe deux versions du plan-type, suivant les grandeurs (P,V ou T) connues.

Les deux autres classes (cas_particulier_chaleur et cas_particulier_énergie_interne) ont des méthodes de résolution composées. C'est le cas pour une transformation isotherme d'un système S

entre deux instant I et F où l'on demande de calculer la chaleur reçue par le système. Le plan produit par SYRCLAD-thermodynamique est le suivant :

```
Le système S est isotherme entre les instants I et F, donc l'énergie
interne est constante, et on a Q = -W. Calculons donc le travail du système
S entre les instants I et F.
```

Il faut alors résoudre un sous-problème : calculer le travail. Ce problème sera classé puis résolu par SYRCLAD-thermodynamique.

4.5.2 Sélectionner des connaissances

Les classes qui n'appartiennent pas à la branche un_système-deux_instants-conservation permettent de sélectionner les connaissances pertinentes pour le problème. Les méthodes de résolution consistent à utiliser THERMOS avec les modules de connaissances choisis.

Ces modules sont au nombre de six :

1 : connaissances relatives à un gaz parfait en équilibre à un instant donné
2 : connaissances relatives à la transformation d'un système entre deux instants
3 : lois ayant trait à la conservation d'une grandeur entre deux instants
4 : lois ayant trait aux systèmes de surface
5 : lois ayant trait aux gazomètres
6 : lois ayant trait aux mélanges de gaz

Le paragraphe 5 présente de nombreux exemples de problèmes résolus par SYRCLAD-thermodynamique. Certaines solutions sont des plans produits par SYRCLAD-thermodynamique, et d'autres sont produites par THERMOS.

5 Résultats

Nous donnons ici des exemples d'exercices de thermodynamique résolus par SYRCLAD-thermodynamique ; ils sont présentés par classe. Les noms des problèmes (bailly_07_01, bailly_10_02, suardet_17) correspondent à des numéros d'exercices de manuels de thermodynamique [Bailly 71, Bertin et Renault 84, Suardet 85, Boutigny 85, Maury et Hulin 86].

Un système, un instant, grandeurs intensives

« Quel est le volume massique du propane dans les conditions normales ? »

Modèle descriptif

Les conditions normales signifient une pression de une atmosphère et une température de 25°C. Modélis donne :

```
problème(bailly_07_01).
systèmes(bailly_07_01,[système_D]).
instants(bailly_07_01,[instant_E]).
température(système_D,instant_E).          /* on connnait la température */
```

```
pression(système_D,instant_E).                          /* et la pression */
gp(système_D).                        /* le propane est un gaz parfait, */
équi_gp(système_D,instant_E).                    /* il est en équilibre */
composition(système_D).                 /* et on connaît sa composition.*/
on_demande(bailly_07_01,[vol_massique(système_D,instant_E)]).
```

Classement

SYRCLAD-thermodynamique déclenche des règles pour classer le problème :

```
bailly_07_01
nb_sys nb_ins int
c_1s_1i_int
```

Il y a un système en équilibre à un instant. Toutes les grandeurs sont intensives.

Résolution

La méthode de résolution indique qu'il faut utiliser THERMOS avec le module de connaissances numéro 1 : les règles relatives à un système en équilibre. De plus, THERMOS n'utilisera que les relations faisant intervenir des grandeurs toutes intensives.

Des règles "quand" se déclenchent :

```
gp(système_D) équi_gp(système_D, instant_E) fp(système_D) équi_fp(système_D,
instant_E) équi_gp_int(système_D, instant_E) sq(système_D) équi_sq(système_D,
instant_E) équi_fp_int(système_D, instant_E) équi_sq_int(système_D, instant_E)
```

gp(système_D) implique fp(système_D) (fluide parfait) qui implique sq(système_D) (système quelconque). équi_gp(système_D, instant_E) implique équi_fp(système_D, instant_E) et équi_gp_int(système_D, instant_E) qui impliquent équi_sq(système_D, instant_E) et équi_fp_int(système_D, instant_E) et enfin équi_sq_int(système_D, instant_E)

Et une règle "si" amenée par sq(système_D) :
composition(système_D) implique masse_molaire(système_D).

Toutes les règles "quand" ont asserté un ensemble de relations, en particulier :

```
rel(pression(système_D,instant_E),vol_molaire(système_D,instant_E),
température(système_D, instant_E))              /* p*volMolaire=R*T */     1
rel(vol_molaire(système_D,instant_E),masse_molaire(système_D),masse_volumiq
ue(système_D, instant_E))              /* M=masse_volic*vol_molaire */     2
rel(vol_massique(système_D, instant_E), masse_volumique(système_D,
instant_E))                       /* vol_massique=1/masse_volumique */     3
```

Le plan de résolution produit par THERMOS est alors le suivant :

```
pression(système_D,instant_E) + température(système_D,instant_E) + 1
=> vol_molaire(système_D, instant_E)

masse_molaire(système_D) + vol_molaire(système_D) + 2
=> masse_volumique(système_D, instant_E)

masse_volumique(système_D, instant_E) + 3
=> vol_massique(système_D, instant_E)
```

Un système, deux instants, pas de conservation

« Quel volume occupe 1 kg d'air à 15°C, sous la pression de 100 bars absolus ? À quelle température faut-il porter cet air pour que, sous la même pression, son volume double ? On donne r de l'air = 287 J/kg*deg. »

Modèle descriptif

```
problème(bailly_07_15).
systèmes(bailly_07_15,[système_B]).
instants(bailly_07_15,[instant_D,instant_E]).
pression(système_B,instant_E).
température(système_B,instant_D).
pression(système_B,instant_D).
masse(système_B,instant_D).
rapport(volume,système_B,instant_D,instant_E).
r_molaire(système_B).
composition(système_B).
gp(système_B).
trans_gp_fermé(système_B,instant_D,instant_E).
on_demande(bailly_07_15,[température(système_B,instant_E),
 volume(système_B,instant_D)]).
```

Classement

```
bailly_07_15
nb_sys nb_ins sans
c_1s_2i_sans
```

Il y a un système, deux instants, et pas de conservation de grandeur[1].

Résolution

Les modules de connaissances utilisés par THERMOS pour les problèmes de cette classe sont les modules 1 (un système en équilibre) et 2 (un système en transformation).

Il y aura donc plus de règles "quand" déclenchées que pour l'exercice précédent. Par exemple la règle quand(trans_gp_fermé(système_B,instant_D,instant_E)), qui asserte la loi : V1/V2 = P1T1/P2T2.

Le plan de résolution est le suivant :

```
rel(rapport(volume, système_B, instant_D, instant_E), pression(système_B,
instant_D), pression(système_B, instant_E), température(système_B,
instant_D), température(système_B, instant_E)) /* V1/V2 = (p1 T1)/(p2 T2)
*/ +
```

[1] Quand le modèle descriptif ne contient aucun fait traduisant une conservation de grandeur, une règle par défaut déduit qu'il n'y a pas de conservation.

```
rapport(volume, système_B, instant_D, instant_E) + pression(système_B,
instant_D) + pression(système_B, instant_E) + température(système_B,
instant_D)
=> température(système_B,  instant_E)
```

Comme on connaît la composition du gaz, on connaît sa masse molaire.

```
rel(n(système_B, instant_D), masse(système_B, instant_D),
masse_molaire(système_B)) /* n = m/M */ + masse(système_B, instant_D) +
masse_molaire(système_B))
=> n(système_B, instant_D)

rel(pression(système_B, instant_D), volume(système_B, instant_D),
n(système_B, instant_D), température(système_B, instant_D)) /* pV=nRT */ +
pression(système_B, instant_D) + n(système_B, instant_D) +
température(système_B, instant_D)
=> volume(système_B,  instant_D)
```

Deux systèmes, un instant, gazomètre

« Un gazomètre, dont la cloche, de 40 m de diamètre, a une masse de 315 t, renferme 35000 m^3 de gaz à 17°C. La pression atmosphérique est de 740 mm Hg. On demande de déterminer :

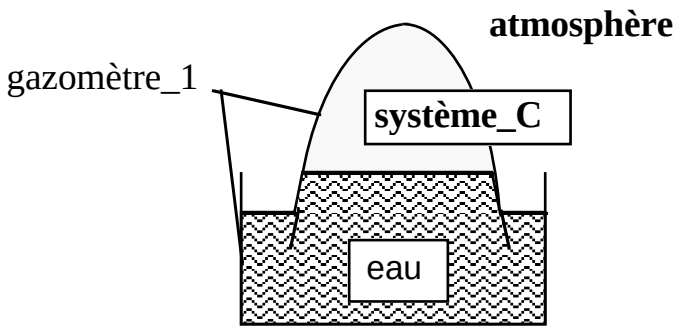
- la pression de distribution du gaz ;

- la masse de gaz contenue dans le gazomètre.

La constante du gaz est r = 680 J/kg*deg. On néglige la poussée d'Archimède sur la partie immergée de la cloche. »

Modèle descriptif

```
problème(bailly_07_04).
systèmes(bailly_07_04,[système_C,atmosphère]).
instants(bailly_07_04,[instant_D]).
température(système_C,instant_D).
pression(atmosphère,instant_D).
volume(système_C,instant_D).
r_molaire(système_C).
gazomètre(gazomètre_1).
masse(gazomètre_1).
diamètre(gazomètre_1).
gp(système_C).
équi_gp(système_C,instant_D).
équi_gazomètre(gazomètre_1,système_C,atmosphère,instant_D).
on_demande(bailly_07_04,[pression(système_C,instant_D),
 masse(système_C,instant_D)]).
```



Bailly_07_04

Classement

```
bailly_07_04
nb_sys nb_ins gazo
c_2s_1i_gazo
```

Il y a deux systèmes gazeux : le gaz sous la cloche et l'atmosphère. Ils sont reliés par le gazomètre.

Résolution

THERMOS utilise deux modules de connaissances : le 1 (système en équilibre) et le 5 (gazomètres). Dans le module lié aux gazomètres, la règle quand(gazomètre(gazpmètre_1)) est déclenchée, elle asserte entre autres : pression de la cloche = poids/section. La règle quand(équi_gazomètre(gazomètre_1,système_C,atmosphère,instant_D)) est aussi déclenchée, elle asserte entre autres la relation : pression du gaz = pression de la cloche + pression atmosphérique.

Le plan de résolution produit par THERMOS est le suivant :

```
rel(r_molaire(système_C), masse_molaire(système_C)) /* r = R/M*/ +
r_molaire(système_C)
=> masse_molaire(système_C)

rel(diamètre(gazomètre_1), section(gazomètre_1)) /* section = pi diamètre
*/ + diamètre(gazomètre_1)
=> section(gazomètre_1)

rel(masse(gazomètre_1), poids(gazomètre_1)) /* poids = g masse */ +
masse(gazomètre_1)
=> poids(gazomètre_1)

rel(poids(gazomètre_1), section(gazomètre_1), p_cloche(gazomètre_1)) /* p =
poids/section */ + poids(gazomètre_1) + section(gazomètre_1)
=> p_cloche(gazomètre_1)

rel(pression(atmosphère, instant_D), pression(système_C, instant_D),
p_cloche(gazomètre_1)) /* équilibre mécanique: p(gaz) = p(Cloche) +
p(atmosphere) */ + pression(atmosphère,instant_D) + p_cloche(gazomètre_1)
=> pression(système_C,  instant_D)

rel([pression(système_C, instant_D), volume(système_C, instant_D),
n(système_C, instant_D), température(système_C, instant_D)) /* pV=nRT */ +
```

```
pression(système_C, instant_D) + volume(système_C, instant_D) +
température(système_C, instant_D)
=> n(système_C, instant_D)

rel(n(système_C, instant_D), masse(système_C, instant_D),
masse_molaire(système_C)) /* n = m/M */ + n(système_C, instant_D) +
masse_molaire(système_C)
=> masse(système_C,  instant_D)
```

Deux systèmes, deux instants, ambiance, conservation d'une grandeur

« Un cylindre indilatable et indéformable, fermé par un piston de masse négligeable et mobile sans frottement, renferme 0,1 m^3 d'un gaz parfait à la température de 27 degrés C et à la pression de 1,0E5 pascals.

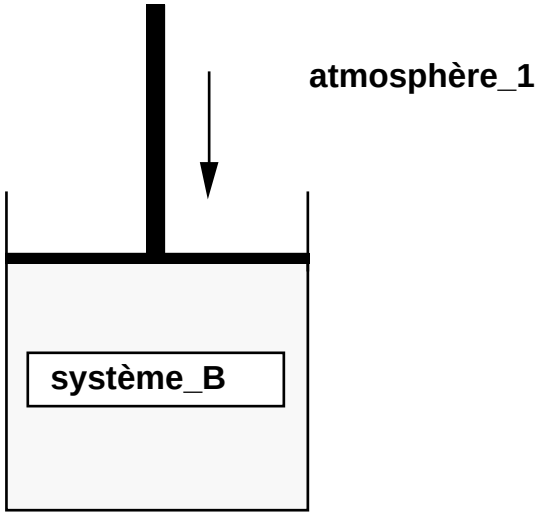
Il est placé dans une atmosphère où la pression est constante et égale à 1,0E5 pascals. Calculer le travail effectif qu'il faut fournir au piston pour exécuter une compression isotherme réduisant le volume à 0,01 m^3.

Quelle est la quantité de chaleur échangée par le gaz avec le milieu extérieur pendant cette compression ?

La masse molaire du gaz est 44 g. cv = 0,71 J/g*K. »

Modèle descriptif

```
problème(suardet17).
systèmes(suardet17,[système_B,atmosphère_1]).
instants(suardet17,[instant_D,instant_E]).
température(système_B,instant_D).
pression(système_B,instant_D).
masse(système_B,instant_E).
pression(atmosphère_1,instant_D).
volume(système_B,instant_E).
volume(système_B,instant_D).
masse(système_B).
compression(système_B,instant_D,instant_E).
trans_gp_fermé(système_B,instant_D,instant_E).
trans_gp_int(atmosphère_1,instant_D,instant_E).
isotherme(système_B,instant_D,instant_E).
isobare(atmosphère_1,instant_D,instant_E).
cv(système_B,instant_D).
ambiance(système_B,atmosphère_1,instant_D,instant_E).
gp(système_B).
on_demande(suardet17,[travail_effectif(système_B,atmosphère_1,instant_D,ins
tant_E),chaleur(système_B,instant_D,instant_E)]).
```



Suardet17

Classement

```
suardet17
nb_sys nb_ins comp4 isotherme
c_2s_2i_ambiance_cons
```

Il y a deux systèmes, deux instants. Un système (système_B) est plongé dans l'ambiance de l'autre (atmosphère_1). Comme la transformation est isotherme, il y a conservation d'une grandeur.

Résolution

THERMOS utilise trois modules de connaissances : le 1 (système en équilibre), le 2 (système en transformation) et le 3 (conservation d'une grandeur). Il n'y a pas de connaissances particulières pour le lien entre deux systèmes : ambiance. En effet, ici le piston n'a aucune importance.

Dans le module 3, des règles se déclenchent, par exemple : si une transformation est isotherme, alors PV = constante : on connaît une fonction de P et V constante pendant la transformation .

Le plan de résolution produit par THERMOS et rédigé en français est le suivant :

```
On connaît une fonction de P et V constante pendant la transformation. Or on
connaît P et V à l'instant initial, donc on connaît la valeur de cette
fonction. Comme d'autre part on connaît aussi le volume à l'instant final, on
peut intégrer dW en V, et on obtient travail(système_B,instant_D,instant_E).
On connaît la variation de volume du système_B entre les deux instants, or W =
-pDV, on obtient donc travail_fourni(atmosphère_1, système_B, instant_D,
instant_E).
travail_effectif(système_B,instant_D,instant_E) = travail(système_B, instant_D,
instant_E) - travail_fourni(atmosphère_1, système_B, instant_D, instant_E).
La transformation est isotherme , donc l'énergie interne est constante, et on a
Q = -W.
```

Deux systèmes, deux instants, mélange, pas de conservation

« Un litre d'oxygène (O_2 = 32 g) à 20 °C sous 3 atmosphères et 3 litres de dioxyde de carbone (CO_2 = 44 g) à 50 °C sous 2 atmosphères sont mélangés dans un récipient de volume 5 litres à 40 °C. Calculer la pression et la masse molaire du mélange. »

Modèle descriptif

```
problème(suardet_06_02).
systèmes(suardet_06_02,[s1,s2]).
instants(suardet_06_02,[i,f]).
équi_gp(s1,i).
volume(s1,i).
masse_molaire(s1).
température(s1,i).
pression(s1,i).
équi_gp(s2,i).
volume(s2,i).
masse_molaire(s2).
température(s2,i).
pression(s2,i).
mélange([s1,s2],i,s3,f).
équi_gp(s3,f).
volume(s3,f).
température(s3,f).
on_demande(suardet_06_02,[pression(s3,f),masse_molaire(s3)]).
```

Classement

```
suardet_06_02
nb_sys nb_ins comp3 sans
c_2s_2i_mélange_sans
```

Il y a deux systèmes, deux instants. Les deux systèmes sont mélangés. Il n'y a pas de conservation d'une grandeur.

Résolution

THERMOS utilise les modules 1 (système en équilibre), 2 (système en transformation) et 6 (mélange de gaz). Dans ce module 6, la règle quand(mélange([s1,s2],i,s3,f)) se déclenche et asserte les lois : m1 + m2 = m3 et n1 + n2 = n3.

Le plan de résolution est le suivant :

rel(pression(s1, i), volume(s1, i), n(s1, i), température(s1, i)) /* p V = n R T */ + pression(s1, i) + volume(s1, i) + température(s1, i)
=> n(s1, i)

rel(pression(s2, i), volume(s2, i), n(s2, i), température(s2, i)) /* p V = n R T */ + pression(s2, i) + volume(s2, i) + température(s2, i)
=> n(s2, i)

rel(n(s1, i), n(s2, i), n(s3, f)) /* n1+n2=n3 */ + n(s1, i) + n(s2, i)
=> n(s3, f)

rel(n(s1, i), masse(s1, i), masse_molaire(s1)) /* n = m/M */ + n(s1, i) + masse_molaire(s1)
=> masse(s1, i)

rel(n(s2, i), masse(s2, i), masse_molaire(s2)) /* n = m/M */ + n(s2, i) + masse_molaire(s2)
=> masse(s2, i)

rel(pression(s3, f), volume(s3, f), n(s3, f), température(s3, f)) /* p V = n R T */ + volume(s3, f) + n(s3, f) + température(s3, f)
=> **pression(s3, f)**

rel(masse(s1, i), masse(s2, i), masse(s3, f)) /* m1+m2=m3 */ + masse(s1, i) + masse(s2, i)
=> masse(s3, f)

rel(n(s3, f), masse(s3, f), masse_molaire(s3)) /* n = m/M */ + n(s3, f) + masse(s3, f)
=> **masse_molaire(s3)**

Un système, deux instants, conservation, gamma

« Un gaz parfait est enfermé dans un cylindre dont le volume est limité par un piston. L'état initial de ce gaz est caractérisé par les paramètres P_0,T_0.

En appuyant sur le piston, on élève lentement la pression jusqu'à la valeur P_1. On suppose que cette transformation quasi statique est en outre adiabatique. Calculer la température finale T_1, en fonction de T_0, P_0, P_1 et du rapport γ, supposé indépendant de T, des capacités thermiques du gaz. »

Classement

```
boutigny21bis
nb_sys nb_ins adiabatique cas_part_gam
c_1s_2i_cons_gam
```

Il y a un système, deux instants. Comme la transformation est adiabatique, il y a conservation d'une grandeur. Nous somme dans le cas particulier - gamma.

Modèle opérationnel

Le modèle opérationnel du problème est composé par les valeurs des attributs disciminants :

```
Un système
Deux instants
Conservation d'une grandeur
Cas particulier gamma
```

Il est complété afin d'instancier le problème par rapport à la classe :

```
On connaît gamma, la transformation est adiabatique, deux variables
initiales : P et T, une variable finale : P, on demande l'autre : T.
```

Plan

SYRCLAD-thermodynamique adapte au problème le plan-type pour le cas particulier gamma grâce à l'instanciation ci-dessus :

```
Le système s est adiabatique entre les instants i et f. Comme on connaît
gamma, on utilise la loi de Laplace et le fait que PV/T=cte (système
fermé), on en déduit qu'il existe une fonction f(pression,température)
constante entre i et f. Comme on connaît pression(s,i) et température(s,i),
on connaît la valeur de cette fonction. Comme d'autre part on connaît
pression(s,f), on en déduit température(s,f).
```

Un système, deux instants, conservation, énergie, invariant W

« Un récipient de volume 10 litres contient de l'air sous la pression de 80 cm de mercure. On fait subir à ce gaz une compression isotherme jusqu'à la pression de 800 cm de mercure. Quels sont les échanges de travail du gaz avec le mileu extérieur ? On assimilera l'air à un gaz parfait. On donne la masse du litre d'air normal 1,293 gr et gamma = 1,4. »

Modèle opérationnel

Le modèle opérationnel du problème est composé par les valeurs des attributs disciminants :

```
Un système
Deux instants
Conservation d'une grandeur
Cas particulier énergie
Cas particulier travail-invariant
```

Il est complété afin d'instancier le problème par rapport à la classe :

```
Un invariant : T, deux variables initiales : P et V, une variable finale :
P, on demande le travail.
```

Plan

Le plan-type pour le cas particulier travail-invariant est adapté par SYRCLAD-thermodynamique au problème grâce à l'instanciation ci-dessus :

```
système_D est un système fermé (n=cte), et la variable température est
constante entre instant_E et instant_F, donc il existe une fonction
f(pression,volume) constante entre instant_E et instant_F. Comme on connaît
pression(système_D,instant_E) et volume(système_D, instant_E ), on connaît
la valeur de cette fonction. Comme d'autre part on connaît
pression(système_D,instant_F), on peut intégrer dw en pression pour obtenir
le travail du système système_D entre les instants instant_E et instant_F.
```

Un système, deux instants, conservation, énergie, gamma W

« On fait subir une compression adiabatique quasi-statique à 1l d'azote de gamma=1,4 à 20°C, jusqu'à le réduire à 2/3l. Calculer le travail reçu par le gaz. »

Modèle opérationnel

Le modèle opérationnel du problème est composé par les valeurs des attributs disciminants :

```
Un système
Deux instants
Conservation d'une grandeur
Cas particulier énergie
Cas particulier travail-gamma
```

Il est complété afin d'instancier le problème par rapport à la classe :

```
On connaît gamma, la transformation est adiabatique, deux variables
initiales : T et V, une variable finale : V, on demande le travail.
```

Plan

Le plan-type pour le cas particulier travail-gamma est adapté par SYRCLAD-thermodynamique au problème grâce à l'instanciation ci-dessus :

```
Le système système_C est adiabatique entre les instants instant_G et
instant_H. Comme on connaît gamma, on utilise la loi de Laplace et le fait
que PV/T=cte (système fermé), on en déduit qu'il existe une fonction
f(volume,température) constante entre instant_G et instant_H. Comme on
connaît volume(système_C,instant_G) et température(système_C,instant_G), on
connaît la valeur de cette fonction. Comme d'autre part PV/T=cte et que
l'on connaît volume(système_C,instant_H), on peut intégrer dw en  volume
pour obtenir le travail du système système_C entre les instants instant_G
et instant_H.
```

Un système, deux instants, conservation, énergie, isotherme Q

« Un récipient de volume 10 litres contient de l'air sous la pression de 80 cm de mercure. On fait subir à ce gaz une compression isotherme jusqu'à la pression de 800 cm de mercure. Quels sont les échanges de chaleur du gaz avec le milieu extérieur ? On assimilera l'air à un gaz parfait. On donne la masse du litre d'air normal 1,293 gr et gamma = 1,4. ».

Modèle opérationnel

Le modèle opérationnel du problème est composé par les valeurs des attributs disciminants :

```
Un système
Deux instants
Conservation d'une grandeur
Cas particulier énergie
Cas particulier chaleur isotherme
```

Plan

La méthode de résolution pour cette classe consiste à résoudre un sous-problème : calculer le travail :

```
Le système S est isotherme entre les instants I et F, donc l'énergie
interne est constante, et on a Q = -W. Calculons donc le travail du système
S entre les instants I et F.
```

SYRCLAD-thermodynamique classe et résout le sous-problème :

```
S est un système fermé (n=cte), et la variable température est constante
entre I et F, donc il existe une fonction f(pression, volume) constante
entre I et F. Comme on connaît pression(S, I) et volume(S, I), on connaît
la valeur de cette fonction. Comme d'autre part on connaît pression(S, F),
on peut intégrer dw en pression pour obtenir le travail du système S entre
les instants I et F.
```

6 Conclusion

SYRCLAD-thermodynamique résout une quarantaine d'exercices de thermodynamique variés. Nous verrons au chapitre 9 que la méthode de décomposition de problèmes que nous avons donnée à SYRCLAD-thermodynamique lui permet de résoudre une vingtaine d'exercices supplémentaires. Les modèles descriptifs des problèmes de thermodynamique font intervenir un grand nombre de prédicats. Le graphe de classification construit est riche. Les exercices du corpus sont répartis de manière régulière dans les classes de ce graphe. Sa taille est de 31 classes dont 21 opérationnelles. Les méthodes de résolutions sont variées, certaines utilisant un moteur d'inférence (THERMOS) sur des modules de connaissances pertinentes pour le problème. D'autres produisent un plan de résolution adapté au problème.

6.1 Construction du graphe de classification

L'application de SYRCLAD à la thermodynamique a été basée sur l'étude de Modélis [Tisseau 90]. Nous avons tenté de construire une classification de problèmes de thermodynamique en suivant l'avis de l'expert (G. Tisseau) selon lequel "on devrait pouvoir dégager une classification". Ce travail de construction d'un graphe de classification a été long et vraiment difficile, d'une part parce que c'est un travail difficile quel que soit le domaine et d'autre part parce que le domaine de la thermodynamique est complexe. Nous avons décrit dans ce chapitre la démarche que nous avons utilisée pour construire cette classification. La méthode que nous avons employée peut s'avérer efficace pour construire un graphe de classification dans un autre domaine. L'idée générale est d'étudier de nombreuses résolutions d'exercices. C'est cette idée qui est utilisée par ALEX [Séroussi et Morice 93] et d'EUREKA [Elio et Scharf 90] (cf. chapitre 2). Un conseil possible pourrait être : pour classer des problèmes, commencer par classer des résolutions. Dans ces résolutions, repérer les connaissances qui sont utilisées ensemble, et si possible plus précisément les séquences d'applications de connaissances. Dans le premier cas, on obtiendra des "modules de connaissances" et dans l'autre des "plans" (macros, chunks). Classer alors les problèmes en fonction des modules et

des plans qu'on utilise pour les résoudre. Cela devrait permettre de repérer les classes opérationnelles. La difficulté consiste ensuite à décrire ces classes non plus en fonction des résolutions, mais en fonction des énoncés. On peut au moins essayer de repérer si certains traits de surface (présents dans l'énoncé ou calculables facilement) sont caractéristiques de certaines classes. En général, cela ne suffit pas. Il faut alors introduire des concepts plus abstraits liés aux conditions d'application des modules ou des plans (exemple : notion de transformation, d'où l'introduction du nombre d'instants). La difficulté est surtout là : quels concepts abstraits introduire (parfois ils existent déjà dans la théorie, parfois il faut les inventer), et comment analyser un énoncé (pour le reformuler) en fonction de ces concepts ? Des exemples pratiques sont donnes dans ce chapitre : "on ne peut parler de ... que s'il y a ...".

Cette démarche de construction d'une classification est peut-être adaptée à ce type de domaines où les problèmes consistent à propager des valeurs dans des formules, comme ceux abordés par G. Pecego dans le cadre de la génération de problèmes : chimie, électricité, mécanique [Pecego 97].

Le graphe de classification que nous avons construit permet de résoudre efficacement des problèmes de thermodynamique. Nous l'avons soumis à un expert du domaine (G. Tisseau) qui l'a approuvé. Si l'on veut utiliser ce graphe dans le cadre d'un EIAO, il faudra le faire valider par un didacticien de la discipline.

6.2 Apport par rapport à Modélis

Une des taches importantes de Modélis est de construire un modèle du problème à partir d'un énoncé en langage naturel. Cette tache n'a pas été abordée par SYRCLAD-thermodynamique et on ne peut donc pas les comparer sur ce point. SYRCLAD-thermodynamique prend en entrée comme modèle descriptif le modèle engendré par le modéliseur de Modélis. Pour obtenir ce modèle, Modélis doit identifier les systèmes, les instants, les lieux, les mesures (quel système, quel instant, quelle quantité, dans quelles conditions), les quantités demandées, les transformations, les propriétés observées ou supposées (être un gaz parfait), les dispositifs connus (gazomètre).

SYRCLAD-thermodynamique doit être comparé avec le résolveur de Modélis. Celui-ci fonctionne essentiellement en construisant d'abord un réseau de contraintes traduisant les hypothèses puis en propageant les informations connues dans ce réseau pour obtenir des informations nouvelles. Le moteur THERMOS utilisé par SYRCLAD-thermodynamique est fonctionnellement équivalent au résolveur de Modélis mais n'est pas implémenté de la même façon. Dans Modélis, le réseau construit ne contient que les connaissances applicables dans la situation donnée, et ensuite chaque information nouvelle n'active que les contraintes concernées (on suit les liens du réseau). Dans THERMOS, les contraintes potentielles sont représentées par des règles qui sont toutes parcourues systématiquement à chaque cycle. Cette différence n'est pas importante dans le contexte de cette thèse : l'essentiel n'était pas la réalisation d'un moteur THERMOS optimisé, mais l'insertion d'un moteur de ce type dans le cadre conceptuel de SYRCLAD.

Dans cette façon de résoudre, Modélis (et donc THERMOS) n'a aucune vision globale du problème. La solution émerge simplement de la propagation dans le réseau, sans qu'il y ait de plan préalable. Le résolveur de Modélis n'a pas de connaissances cachées concernant une classification des problèmes :

il n’utilise que les propriétés de base de la thermodynamique. Mais SYRCLAD-thermodynamique ne se réduit pas à THERMOS. SYRCLAD-thermodynamique, lui, a des connaissances sur les classes de problèmes possibles, et il cherche à identifier la classe du problème avant de lancer la résolution proprement dite. Cela lui permet dans certains cas de ne pas utiliser du tout THERMOS car la solution résulte alors d’un plan prédéfini (que Modélis ne connaît pas).

SYRCLAD-thermodynamique produit une reformulation du problème plus intelligible que Modélis, même si elles sont mathématiquement équivalentes. Il peut par exemple expliciter le rôle des données en les situant dans un contexte : "on connaît les valeurs *initiales* de *deux* quantités et la valeur *finale* de l’*une* d’entre elles, on demande la valeur *finale* de l’*autre*". Modélis se contente d'énumérer les faits connus, alors que SYRCLAD-thermodynamique essaye de les interpréter et de les présenter sous une forme structurée, en faisant ressortir de la modélisation de l'énoncé les points qui seront importants pour la résolution. Ses solutions peuvent être également exemplaires : il montre bien qu’à certaines classes de problèmes sont attachés des plans-types et qu’il est avantageux de chercher d’abord à reconnaître la classe du problème.

Partie III : transversalement aux domaines

La troisième partie du mémoire présente des travaux effectués transversalement aux quatre domaines et après implémentation des systèmes SYRCLAD-X correspondant.

Le chapitre 8 présente une étude de la complexité de ces quatre domaines, la complexité d'un domaine étant définie par rapport à l'architecture de SYRCLAD.

Le chapitre 9 présente comment SYRCLAD décompose un problème en sous-problèmes afin de le résoudre. Il détaille comment le système compare un problème qu'il ne sait pas résoudre au graphe de classification, pour déterminer en quoi il est différent des problèmes que le système sait directement résoudre. SYRCLAD construit ainsi des sous-problèmes à résoudre pour que le problème donné entre dans la classification. Cette méthode a été appliquée pour les problèmes additifs et la thermodynamique.

Chapitre 8

Des critères pour évaluer la complexité des connaissances d'un domaine X données à SYRCLAD-X

Nous souhaitons faire dans ce chapitre un bilan des quatre applications de SYRCLAD, en permettant au lecteur de se faire une idée de la taille des quatre systèmes SYRCLAD-dénombrements, SYRCLAD-additifs, SYRCLAD-tarot et SYRCLAD-thermodynamique. Nous définissons des critères d'évaluation de la complexité de chacun de ces systèmes, c'est-à-dire de la complexité des connaissances fournies par l'expert du domaine X à SYRCLAD pour réaliser SYRCLAD-X. Nous présenterons les critères que nous avons retenus pour évaluer dans chaque domaine la complexité du langage de description des problèmes et des bases de connaissances de classification, de reformulation et de résolution. Nous préciserons les résultats obtenus pour chacun de ces critères, puis nous récapitulerons domaine par domaine la complexité obtenue et la nature des difficultés rencontrées pour résoudre un problème dans ce domaine.

Plan du chapitre

L'architecture de SYRCLAD et le processus de résolution effectué par son noyau sont indépendants du domaine. Le langage de description des problèmes et les bases de connaissances de classification, de reformulation et de résolution sont fournies par l'expert du domaine. Nous définissons ici la complexité d'un SYRCLAD-X comme la quantité globale de connaissances nécessaires à ce système pour résoudre des problèmes du domaine, c'est-à-dire la quantité de connaissances décrites par le langage de description et les trois bases de connaissances.

Intuitivement, on pourrait dire que la thermodynamique est un domaine plus complexe que les problèmes additifs, et que le tarot semble plus calculatoire. Mais comment comparer la complexité des dénombrements et celle de la thermodynamique, sont-elles de même nature ? Nous définissons la complexité d'un domaine par la complexité des bases de connaissances fournies à SYRCLAD dans ce domaine et celle du langage de description. Si un tuteur intelligent dans un domaine X utilise SYRCLAD-X comme résolveur de référence, cette mesure de la nature de la complexité d'un domaine permet de savoir quelles sont les étapes de résolution nécessitant le plus de connaissances pour le système dans ce domaine. Nous faisons l'hypothèse que la quantité globale de connaissances qu'il faut donner à un système pour qu'il puisse résoudre des problèmes du domaine est un premier indice pour évaluer la difficulté qu'éprouve un humain à résoudre ces problèmes. En effet, nous supposons qu'une grande quantité de connaissances est plus difficile à mémoriser et à utiliser pour un être humain. Comme la démarche de résolution de SYRCLAD-X doit être celle qu'un tuteur veut enseigner, celui-ci pourra alors tenir compte des étapes difficiles pour SYRCLAD-X afin d'élaborer ses stratégies d'enseignement. La complexité que nous définissons est celle des connaissances utilisées par le résolveur, il y a donc lieu de la confronter à une complexité "pour les élèves".

Cette étude sur la complexité des domaines n'est pas achevée puisque nous n'avons défini que des critères sur la quantité globale de connaissances données à un système ; il faudra par la suite adapter ces critères à un niveau individuel : à une classe de problème, et à un problème résolu par le système. Bien que nous ayons défini des critères numériques, les conclusions que nous en tirons sont encore approximatives et un peu subjectives puisque nous évaluons nous-mêmes les conlusions à tirer des critères numériques et nous nous en servons pour corroborer notre appréciation de la difficulté à résoudre des problèmes d'un domaine. Nous préciserons en fin de chapitre les perspectives que nous avons quant à la poursuite de cette étude.

1 Complexité du langage de description

1.1 Définition des critères

Dans chaque domaine X, l'expert du domaine doit définir un langage de la logique du premier ordre qu'on peut utiliser pour exprimer les modèles descriptifs des problèmes. Ce langage est défini par un ensemble de prédicats utilisables. Nous définissons la complexité du langage de description par le nombre de prédicats qui le composent. Cette complexité ainsi définie est globale, il faudra également tenir compte du nombre effectif de prédicats utilisés dans *un* modèle descriptif.

Pour décrire la complexité du langage de description, nous avons choisi de répartir les prédicats en trois catégories : ceux introduisant un objet du domaine (par exemple : est_un(s,sac)), ceux décrivant

une relation entre objets (par exemple : contenu(s,e)), et ceux qui précisent une propriété d'un objet (par exemple : taille(e,25)). Les objets et la façon dont ils sont reliés définissent des situations concrètes qui sont celles décrites par les énoncés en langage naturel. Les énoncés contiennent également une question à laquelle il faut répondre pour résoudre le problème.

Pour évaluer la complexité du langage de description, nous avons choisi de compter le nombre de prédicats de chacune des trois catégories, nous nous intéressons au(x) prédicat(s) utilisé(s) pour poser la question. Les ensembles de prédicats définissant pour chaque domaine le langage de description sont donnés en annexe.

1.2 Dénombrements

Le langage de description des problèmes de dénombrement est défini par 69 prédicats dont :

26 prédicats de type objet,
22 prédicats de type relation,
21 prédicats de type propriété.

La question est de dénombrer un ensemble : à_dénombrer(Pb,Ens).

1.3 Problèmes additifs

Le langage de description des problèmes additifs est défini par 18 prédicats dont :

6 prédicats de type objet,
11 prédicats de type relation,
1 prédicat de type propriété.

La question est de calculer la taille d'un ensemble de billes : à_calculer(Pb,Ens).

1.4 Tarot

Le langage de description des problèmes de tarot est défini par 9 prédicats dont :

5 prédicats de type objet,
4 prédicats de type relation,
0 prédicats de type propriété.

La question est de choisir un plan de jeu, c'est donc la même question pour tous les problèmes.

1.5 Thermodynamique

Le langage de description des problèmes de thermodynamique est défini par 85 prédicats dont :

48 prédicats de type objet,
10 prédicats de type relation,
27 prédicats de type propriété.

On demande la valeur d'une grandeur thermodynamique : on_demande(Pb, masse(S)).

1.6 Conclusion

Le langage de description est riche pour les exercices de dénombrement et de thermodynamique (il y a beaucoup de prédicats). On remarque que pour les dénombrements, les prédicats de type objet, relation et propriété sont en quantité égale, alors qu'en thermodynamique on a beaucoup plus de prédicats de type objet, mais peu de type relation. Ceci est en adéquation avec notre appréciation que les modèles descriptifs des exercices de dénombrement sont plus structurés que ceux de thermodynamique, car les objets sont plus reliés entre eux.

Les problèmes additifs et de tarot ont un langage de description plus simple (peu de prédicats). Nous avons remarqué qu'il y a en fait trois situations (assez simples) possibles pour les modèles descriptifs des problèmes additifs et deux pour le tarot.

Les questions sont pour chaque domaine exprimées grâce à un unique attribut. Elles sont néanmoins variées suivant les problèmes, sauf pour le tarot où la question est toujours la même.

Nous avons défini des critères pour évaluer la complexité d'un langage de description, mais nous avons commencé à évoquer dans cette conclusion la complexité des *modèles descriptifs*. Il faudrait effectivement tenir compte du nombre moyen de prédicats par modèle descriptif, du type (objet, relation, propriété) de ces prédicats et définir des critères d'évaluation de la complexité d'*un* modèle descriptif.

2 Complexité du graphe de classification

Dans chaque domaine X, un expert donne à SYRCLAD un graphe de classification de problèmes du domaine X. Ce graphe contient des attributs discriminants et les valeurs discriminantes associées à ces attributs. Les quatre graphes donnés à SYRCLAD dans les quatre domaines d'application sont présentés en partie II ; ils sont donnés sous leur forme Prolog en annexe.

2.1 Définition des critères

Quels sont les critères que nous avons choisis pour définir la complexité d'un graphe de classification ?

• tout d'abord le nombre de classes, non-opérationnelles et opérationnelles.

• la symétrie ou non-symétrie du graphe, en terme de structure du graphe mais aussi en terme d'attributs. Par exemple, le graphe du tarot (chap. 6 § 2.2) est symétrique car chaque attribut discriminant a deux valeurs possibles, les branches sont de même longueur et ce sont les mêmes attributs discriminants qui interviennent dans toutes les branches du graphe.

• le nombre d'attributs discriminants.

• la portée des attributs définis : l'expert définit un attribut discriminant dans une classe du graphe. Si un attribut est défini à la racine, il est défini pour tous les problèmes (par exemple le nombre de systèmes en thermodynamique), ce qui est plus simple que si un attribut est défini dans une branche du graphe, car il est alors défini uniquement pour certains problèmes (par exemple la conservation d'une grandeur thermodynamique).

• le nombre (minimum, moyen et maximum) de fils par classe.

2.2 Dénombrements

Le graphe donné par l'expert à SYRCLAD-dénombrements est composé de 23 classes, dont 7 non-opérationnelles et 16 opérationnelles. Il est peu symétrique : seuls les attributs "type de l'objet formé" et "remise" interviennent dans plusieurs branches du graphe.

Il y a 6 attributs discriminants, 19 attributds sont définis à la racine, 8 apparaissent dans les branches.

Nombre de fils par classe non-feuille (14) : 6 2 1 2 1 1 1 2 2 1 2 1 1 2.
Les sept classes qui ont un seul fils sont des classes opérationnelles, leur fils correspondent aux cas particuliers.
Nombre minimum de fils par classe : 1
Nombre maximum de fils par classe : 6
Nombre moyen de fils par classe : 1,8 (2,6 sans les cas particuliers)

2.3 Problèmes additifs

Le graphe donné par l'expert à SYRCLAD-additifs est composé de 31 classes, dont 13 non-opérationnelles et 18 opérationnelles. Il est peu symétrique : seuls les opérateurs AJOUT et RET interviennent dans plusieurs branches du graphe.

Il y a 7 attributs discriminants, 6 attributs sont définis à la racine, 5 apparaissent dans les branches.

Nombre de fils par classe non-feuille : 2 3 2 3 3 2 2 2 2 2 2 2 2.
Il n'y a pas de cas particuliers.
Nombre minimum de fils par classe : 2
Nombre maximum de fils par classe : 3
Nombre moyen de fils par classe : 2,2.

2.4 Tarot

Le graphe donné par l'expert à SYRCLAD-tarot est composé de 26 classes, dont 12 non-opérationnelles et 14 opérationnelles. Il est très symétrique, aussi bien par sa structure que par rapport aux 4 attributs discriminants qui interviennent dans toutes les branches du graphe.

Les 9 attributs sont tous définis à la racine. Il y a 4 attributs discriminants.

Nombre de fils par classe non-feuille : toujours 2.
Nombre minimum de fils par classe : 2
Nombre maximum de fils par classe : 2
Nombre moyen de fils par classe : 2.

2.5 Thermodynamique

Le graphe donné par l'expert à SYRCLAD-thermodynamique est composé de 31 classes, dont 10 non-opérationnelles et 21 opérationnelles. Il est peu symétrique : seuls le nombre de systèmes et d'instants interviennent pour tous les problèmes, l'attribut conservation intervenant également dès qu'il y a deux instants.

3 attributs sont définis à la racine et 6 apparaissent dans les branches. Il y a 8 attributs discriminants.

Nombre de fils par classe non-feuille : 2 2 2 2 3 4 2 4 2 2 2 2.
Nombre minimum de fils par classe : 2
Nombre maximum de fils par classe : 4
Nombre moyen de fils par classe : 2,4.

2.6 Conclusion

Le graphe de classification du tarot est le plus simple : il est complètement symétrique car chaque attribut discriminant a deux valeurs possibles, les branches sont de même longueur et ce sont les même attributs discriminants qui interviennent dans toutes les branches du graphe. De plus tous les attributs sont définis à la racine. Ce graphe est donc facile à mémoriser et à utiliser pour un humain car la démarche est la même pour tous les problèmes : il faut déterminer la valeur de quatre attributs.

Le graphe de classification des problèmes de dénombrement n'est pas très grand, mais il contient beaucoup d'attributs, surtout à la racine ; on note aussi la présence de cas particuliers. Un grand nombre d'attributs est un indice de complexité car cela signifie qu'il va falloir déterminer la valeur de nombreux attributs pour classer un problème.

Les graphes des problèmes additifs et de la thermodynamique sont les plus grands. Ils sont semblables en terme de nombre de classes, nombre d'attributs et nombre de fils. Un graphe de classification (relativement) grand sera difficile à mémoriser pour un élève.

3 Complexité des connaissances de reformulation

Pour utiliser un graphe de classification , il faut déterminer les valeurs des attributs discriminants. L'expert du domaine X doit donner à SYRCLAD-X des règles de reformulation pour utiliser le graphe de classification du domaine X.

3.1 Définition des critères

Nous pensons qu'un critère de complexité dans la résolution d'un problème est le nombre de règles à utiliser pour reformuler ce problème. Nous avons donc choisi en première approximation de compter le nombre de règles de reformulation pour chaque domaine. Il faudra par la suite compter le nombre de règles effectivement déclenchées pour *un* problème.

Comme toutes les règles n'ont pas la même complexité, nous avons défini trois catégories de règles. La première est celle des règles calculatoires : il s'agit dans ces règles de compter quelque chose (des atouts par exemple) ou d'utiliser une procédure pour compter quelque chose (les cartes perdantes par exemple). Nous pensons que les règles calculatoires sont simples à appliquer, ou au pire procédurales. Nous avons séparé les autres règles en deux catégories : les règles de "lecture" et les règles de "déduction". Les règles de lecture sont des règles assez simples à appliquer. Il s'agit dans ces règles de chercher la présence d'un fait dans l'énoncé. Par exemple la règle "isotherme => conservation d'une grandeur" est une règle de lecture ; c'est une règle qui traduit une connaissance importante pour le domaine, mais qui n'est pas difficile à appliquer, il suffit de la connaître. Les règles de déduction sont des règles plus complexes, qui sont difficiles à comprendre et à appliquer. Leurs prémisses sont plus nombreuses et font intervenir d'autres attributs. Par exemple, les règles qui permettent de déterminer le type du problème en dénombrement sont des règles de déduction.

Nous nous intéressons également au nombre (minimum, moyen et maximum) de règles associées à un attribut et permettant d'en calculer la valeur. Par exemple, il y a 16 règles qui concluent sur la valeur du type du problème en dénombrement, ce qui montre qu'il est difficile de déterminer cet attribut.

3.2 Dénombrements

Il y a 83 règles, dont
- 0 règles de calcul,
- 28 règles de lecture,
- 55 règles de déduction.

Nombre minimum de règles par attribut : 1,
Nombre maximum de règles par attribut : 16,
Nombre moyen de règles par attribut : 4.

3.3 Problèmes additifs

Il y a 29 règles, dont
- 0 règles de calcul,
- 9 règles de lecture,
- 20 règles de déduction.

Nombre minimum de règles par attribut : 1,
Nombre maximum de règles par attribut : 8,
Nombre moyen de règles par attribut : 3,2.

3.4 Tarot

Il y a 13 règles, dont
- 11 règles de calcul,
- 2 règles de lecture,
- 0 règles de déduction.

Nombre minimum de règles par attribut : 1,
Nombre maximum de règles par attribut : 2,
Nombre moyen de règles par attribut : 1,3.

3.5 Thermodynamique

Il y a 27 règles, dont
- 0 règles de calcul,
- 19 règles de lecture,
- 8 règles de déduction.

Nombre minimum de règles par attribut : 1,
Nombre maximum de règles par attribut : 8,
Nombre moyen de règles par attribut : 3.

3.6 Conclusion

On remarque que le tarot se distingue par des règles peu nombreuses et essentiellement calculatoires, alors que pour les autres domaines les règles se partagent entre règles de lecture et de déduction, avec plus de règles de lecture pour la thermodynamique et plus de règles de déduction pour les problèmes additifs.

En dénombrement, les règles sont nettement plus nombreuses, essentiellement déductives. Le nombre de règles par attribut est élevé ; cela confirme que les énoncés se réfèrent à des situations très variées puisqu'une règle permet de passer d'une situation concrète à la valeur d'un attribut.

4 Complexité du langage d'expression des modèles opérationnels

4.1 Définition des critères

En utilisant le graphe de classification et les connaissances de reformulation, SYRCLAD construit un modèle opérationnel du problème. Ce modèle opérationnel est exprimé dans un langage de la logique du premier ordre. Les prédicats de ce langage correspondent à des attributs du problème définis par l'expert dans le graphe de classification. Nous donnons en annexe tous les prédicats issus de la classification pour chacun des domaines. Un sous-ensemble de ces prédicats définit le langage d'expression des modèles opérationnels. En effet, parmi les attributs définis dans le graphe de classification, ceux intervenant dans les modèles opérationnels sont d'une part des attributs discriminants et d'autre part des attributs qui permettent d'instancier le problème par rapport à la classe.

Par exemple le modèle opérationnel du problème m10 (chapitre 1) est :

```
Type du problème : répartition[1]
Type de l'objet formé : liste
Remise : avec

Taille de l'ensemble où l'on choisit : 26
Taille de l'objet formé : 5
Ensemble caractéristique : ((1, 2, b),(6, 2, voyelle))
```

Les trois premiers attributs sont des attributs discriminants qui définissent la classe du problème. Les trois derniers sont des attributs non-discriminants, qui instancient le problème m10 par rapport à sa classe. Ils seront utilisés par la méthode de résolution.

---

[1] à cet attribut correspond le prédicat : type_problème(m10, répartition).

4.2 Dénombrements

Il y a 35 attributs définis dans le graphe de classification. 7 d'entre eux sont discriminants et 14 permettent d'instancier le problème par rapport à sa classe.

4.6 Problèmes additifs

Il y a 13 attributs définis dans le graphe de classification. 7 d'entre deux sont discriminants et 2 permettent d'instancier le problème par rapport à sa classe.

4.4 Tarot

Il y a 9 attributs définis dans le graphe de classification, dont 4 discriminants. Les problèmes ne sont pas instanciés par rapport aux classes.

4.5 Thermodynamique

Il y a 13 attributs définis dans le graphe de classification, dont 8 discriminants. 4 attributs permettent d'instancier les problèmes particuliers par rapport aux classes.

4.6 Conclusion

La complexité du langage d'expression des modèles opérationnels est liée à la complexité du graphe de classification. On a en effet dans les deux cas tenu compte du nombre d'attributs discriminants définis dans le graphe de classification.

Cette complexité est également liée à la complexité des méthodes de résolution. En effet, le nombre d'attributs qui permettent d'instancier les problèmes interviendra également dans la complexité des méthodes de résolution, car ils permettent d'appliquer les méthodes de résolution aux problèmes.

Pour réaliser un SYRCLAD-X, un expert du domaine X doit définir un langage de description des problèmes et donner au système des bases de connaissances de classification, de reformulation et de résolution. Il ne doit pas définir de langage d'expression des modèles opérationnels, puisqu'il définit les attributs du problème dans le graphe de classification. De plus, nous avons vu que la complexité que nous avons définie pour le langage d'expression des modèles opérationnels est liée à la complexité du graphe de classification et à celle des méthodes de résolution. Nous pensons donc qu'il n'est peut-être pas nécessaire de prendre en compte cette complexité pour évaluer la complexité d'un SYRCLAD-X.

5 Complexité des méthodes de résolutions

5.1 Définition des critères

L'expert associe une méthode de résolution à chaque classe opérationnelle du graphe de classification. Ces méthodes sont présentées en partie II et certaines sont données en annexe. On s'intéresse pour chaque domaine aux différents types de méthodes de résolution : s'agit-il d'une formule, d'un plan de résolution type, d'un appel à un système expert. De plus, on distingue les résolutions composées, qui construisent des sous-problèmes que SYRCLAD classe et résout. On précise également les méthodes qui utilisent des attributs du problème afin de produire une solution adaptée au problème.

5.2 Dénombrements

Il y a 4 méthodes de résolution composées (produit, complémentaire et deux types de disjonction). Chacune de ces 4 méthodes utilise un attribut spécifique du problème. Pour les cas particuliers (combinaison, arrangement, ...), 7 règles permettent de calculer la solution avec une formule. Dans les 8 autres classes opérationnelles, une règle permet le calcul de la solution grâce à une formule et la production du plan de résolution grâce à un plan-type. Pour utiliser ces formules et ces plans-type, ces règles utilisent 7 attributs du problème.

5.3 Problèmes additifs

Une méthode de résolution composée permet de résoudre successivement suivant le point de vue de chacun des deux participants (chap. 5 § 2.2).

Pour les 18 autres classes opérationnelles, les méthodes de résolution consistent à faire une addition ou une soustraction, elles utilisent 2 attributs : donnée1 et donnée2.

5.4 Tarot

À chacune des 14 classes opérationnelles est associé un plan de jeu. Ce plan de jeu ne s'adapte pas au problème posé. C'est une solution plus qu'une méthode de résolution qui est associée à chacune des classes opérationnelles.

5.5 Thermodynamique

Il y a :

- 2 méthodes de résolution composées qui construisent un sous-problème où la question est différente de celle du problème ("cela revient à calculer le travail"),
- 12 méthodes de résolution qui sélectionnent les connaissances pertinentes parmi 6 modules de connaissances et appellent un moteur (THERMOS) pour utiliser ces connaissances,

- 2 méthodes de résolution qui sélectionnent les connaissances pertinentes avec aussi une sélection du type de grandeurs (extensives/intensives),

- 4 plans-type qui utilisent 4 attributs spécifiques du problème pour produire un plan de résolution.

5.6 Conclusion

Les méthodes de résolution utilisées par SYRCLAD sont variées suivant les domaines et même au sein d'un même domaine. Beaucoup d'entre elles utilisent des attributs du problème afin de produire des solutions adaptées aux problèmes. On trouve des méthodes de résolutions en dénombrements mais également pour les problèmes additifs et la thermodynamique. Les méthodes de résolution du tarot sont moins intéressantes, car il s'agit plutôt de solutions.

6 Complexité par domaine

Nous avons défini des critères pour évaluer dans chaque domaine d'application la complexité du langage de description des problèmes, des connaissances de classification et de reformulation, du langage d'expression des modèles opérationnels obtenus et des connaissances de résolution. En considérant un domaine, on a maintenant une idée plus précise des étapes qui nécessitent le plus de connaissances. Ce ne seront pas les mêmes étapes pour tous les domaines.

Comme nous n'avons pas encore défini une formule tenant compte des critères que nous avons présentés pour calculer une complexité du domaine, nous avons attribué une note entre 1 et 5 pour chaque complexité dont nous avons défini les critères. Cette note tient compte des résultats numériques que nous avons présentés mais elle est très subjective puisqu'elle est attribuée par l'auteur. De manière toute aussi subjective, nous avons choisi en première approximation de faire une moyenne des cinq notes ainsi obtenues, pour avoir une idée comparative de la complexité des quatre domaines d'application. Il faudra introduire des coefficients dans cette moyenne ; nous avons en effet déjà indiqué que la complexité du langage d'expression des modèles opérationnels n'est peut-être pas à prendre en compte.

6.1 Dénombrements

La difficulté de résoudre un problème de dénombrement avec SYRCLAD-dénombrements est due pour une grande part à la richesse du langage de description des problèmes. En effet, les modèles descriptifs sont très variés. Ces situations variées entraînent également une complexité des connaissances de reformulation, bien que le graphe de classification ne soit pas très grand. Le grand nombre de règles de reformulation vient également d'une grande différence entre les modèles descriptifs et les attributs qui permettent le classement. On a besoin de beaucoup de connaissances pour passer d'un exercice concret à un modèle proche de la théorie du domaine. Le modèle opérationnel ainsi construit est riche et permet une résolution bien adaptée aux problèmes grâce à des attributs spécifiques.

Complexité de 1 à 5 :
- modèles descriptifs : 5
- graphe de classification : 3
- règles de reformulation : 5
- modèles opérationnels : 5
- méthodes de résolution : 5

En faisant la moyenne, on obtient une complexité de 4,6 pour les dénombrements.

6.2 Problèmes additifs

Les modèles descriptifs des problèmes additifs ne sont pas très variés, nous avons vu que le langage pour exprimer ces modèles descriptif n'est pas aussi riche que celui des dénombrements. La complexité du domaine est plutôt due au graphe de classification, qui est assez grand. Il est difficile pour un élève de mémoriser un tel graphe. Cette taille du graphe nécessite une quantité assez grande de connaissances de reformulation, qui conduit à construire un modèle opérationnel assez riche pour que les résolutions s'adaptent aux problèmes, en utilisant leurs données numériques.

Complexité de 1 à 5 :
- modèles descriptifs : 2
- graphe de classification : 5
- règles de reformulation : 4
- modèles opérationnels : 4
- méthodes de résolution : 3

En faisant la moyenne, on obtient une complexité de 3,6 pour les problèmes additifs.

6.3 Tarot

Les modèles descriptifs des problèmes de tarot sont tous construits sur le même modèle. Nous avons en effet remarqué que le langage de description des problèmes de tarot contient peu de prédicats. Nous avons également observé que le graphe de classification est très symétrique et tous les attributs sont définis à la racine. Les règles de reformulation sont calculatoires, le modèle opérationnel construit est simple, et les méthodes de résolution ne s'adaptent pas aux problèmes puisqu'il s'agit plutôt de solutions aux problèmes. Ce domaine n'est donc pas très complexe pour SYRCLAD. En fait, la difficulté de l'expertise du choix d'un plan de jeu au tarot réside dans l'expertise qui à une classe associe un plan de jeu, cette expertise est donnée au système, et c'est l'expert (J.-M. Nigro) qui a fourni le travail.

Complexité de 1 à 5 :
- modèles descriptifs : 1
- graphe de classification : 1
- règles de reformulation : 1
- modèles opérationnels : 2

- méthodes de résolution : 1

En faisant la moyenne, on obtient une complexité de 1,2 pour le tarot.

6.4 Thermodynamique

Les modèles descriptifs en thermodynamique sont complexes par le grand nombre de prédicats qu'ils font intervenir, mais il sont moins structurés que ceux des dénombrements. Nous avons en effet noté que le langage de description des problèmes de thermodynamique est riche mais qu'il contient moins de prédicats explicitant des relations entre objets que celui des problèmes de dénombrement ; les objets présents dans les modèles descriptifs sont moins reliés entre eux. Le graphe de classification des problèmes de thermodynamique est assez grand, ce qui entraîne une difficulté de mémorisation pour un être humain. Les règles de reformulation sont nombreuses, mais elles sont moins difficile à mémoriser et à appliquer que celles du domaine des dénombrements. Les méthodes de résolution sont variées et certaines s'adaptent au problème grâce à des attributs spécifiques du problème.

Complexité de 1 à 5 :
- modèles descriptifs : 4
- graphe de classification : 5
- règles de reformulation : 3,5
- modèles opérationnels : 4
- méthodes de résolution : 4

En faisant la moyenne, on obtient une complexité de 4,1 pour la thermodynamique.

7 Conclusion, perspectives

Nous avons défini des critères d'évaluation de la complexité du langage de description des problèmes et des bases de connaissances que l'expert donne à SYRCLAD pour un domaine d'application. Ces critères nous ont aidé à préciser quelles sont les étapes nécessitant beaucoup de connaissances dans la résolution de problèmes d'un domaine par un SYRCLAD-X. Ces étapes s'avèrent difficiles pour un être humain.

Cette étude est loin d'être achevée, et pour que les résultats de cette étude soient utilisables par un tuteur intelligent, de nombreuses améliorations doivent être entreprises.

Tout d'abord, les critères doivent être améliorés. Par exemple, pour la complexité des modèles descriptifs, on compte le nombre de prédicats appartenant au langage utilisé pour exprimer ces modèles. Il faudrait également tenir compte du nombre et de la nature des prédicats dans *un* modèle.

Nous avons défini des critères d'évaluation de la complexité de certaines bases de connaissances, mais nous n'avons pas défini le calcul de complexité correspondant. Pour que ce calcul soit rigoureux et objectif, il faudrait que le système évalue lui-même ces critères (en comptant le nombre de règles, attributs, etc.). Il faudrait de plus qu'il possède une formule permettant de calculer une complexité à

partir de la valeur des critères. Enfin une autre formule est nécessaire pour évaluer la complexité d'un domaine et la nature de cette complexité à partir de la complexité des bases de connaissances.

Cette complexité par base de connaissances est intéressante, car elle permet de savoir quelles sont les étapes les plus délicates dans la résolution des problèmes d'un domaine. Un tuteur peut alors en tenir compte pour élaborer ses stratégies d'enseignement. Il serait également intéressant de définir une complexité pour chaque classe opérationnelle. Le tuteur pourrait s'en servir pour choisir le problème qu'il propose à l'élève, en fonction de la difficulté souhaitée. On peut même définir une complexité par problème, en reprenant les critères que nous avons définis ici. On pourrait alors tenir compte de la taille du modèle descriptif et du type des prédicats présents. Le nombre de règles de reformulation déclenchées par SYRCLAD et leur type (calcul, lecture, déduction) donnerait une idée de la difficulté à modéliser le problème. Enfin, la complexité de la méthode de résolution utilisée par le système pour ce problème compléterait l'évaluation de sa difficulté.

Enfin, tous ces critères d'évaluation permettent de donner une idée de la complexité du point de vue de SYRCLAD. Même si un tuteur utilise SYRCLAD comme résolveur de référence, et a pour but de faire utiliser la méthode de SYRCLAD aux élèves, il faudra confronter ces complexités à la complexité effective des problèmes *pour les élèves*. Ceci devra faire l'objet d'une étude didactique approfondie.

Par ailleurs, nous avons défini ici des critères d'évaluation de la complexité des connaissances données à SYRCLAD pour réaliser un résolveur de problèmes d'un domaine. Il serait également intéressant d'évaluer la complexité à *concevoir* ces bases de connaissances, c'est-à-dire la difficulté à expliciter les connaissances nécessaires à la réalisation d'un SYRCLAD-X. Ceci requiert un suivi complet et détaillé de plusieurs applications de SYRCLAD à des domaines variés, afin de définir des critères d'évaluation d'une telle complexité. Le bon sens et notre expérience personnelle tendent à indiquer que la complexité des bases de connaissances d'un SYRCLAD-X que nous avons définie dans ce chapitre est un premier indicateur de la complexité à modéliser le domaine X. En effet, si les connaissances sont nombreuses et complexes, il est plus difficile de les expliciter. Néanmoins, ce critère a posteriori n'est pas très utile à quelqu'un souhaitant aborder la modélisation d'un domaine.

Il est rare que l'on s'engage dans la modélisation d'un domaine sans point de départ. Les difficultés que l'on rencontrera dépendront du matériel dont on dispose au départ. Nous avons en effet eu bien plus de mal à élaborer un graphe de classification des problèmes de thermodynamique, domaine pour lequel nous disposions d'un système à base de règles n'utilisant pas de connaissances liées à une classification, qu'à expliciter un graphe de classification dans les trois autres domaines, pour lesquels nous disposions soit d'une classification sur papier, soit d'un système à base de règles conçu avec une idée de classification. Les critères de la complexité à modéliser un domaine que nous pourrions définir au vu de notre travail de thèse seraient essentiellement dépendant du matériel de départ. Construire un graphe de classification à partir de classes opérationnelles n'est pas difficile, pour autant que le nombre de classes opérationnelles soit limité. À partir d'une base de règles destinée à résoudre des problèmes et créée avec une idée de classification, on peut distinguer des connaissances de reformulation, des connaissances définissant des classes opérationnelles, et des connaissances de résolution. La complexité à expliciter des connaissances de reformulation dépendra de la variété des objets apparaissant dans les énoncés de problèmes ainsi que de la "distance" entre ces traits de surface et les attributs discriminants de la classification. La complexité à expliciter des

méthodes de résolution dépendra de la taille des connaissances théoriques du domaine (particulièrement grande par exemple pour la thermodynamique), et du type d'action pour résoudre : s'il s'agit d'appliquer des formules les unes après les autres de manière à un produire un plan de résolution (comme en thermodynamique), la combinatoire est plus grande que s'il s'agit de choisir un ensemble de buts moins dépendants les uns des autres (comme au tarot).

Chapitre 9

Résoudre en décomposant un problème en sous-problèmes

Nous présentons dans ce chapitre deux manières d'utiliser SYRCLAD pour décomposer un problème en sous-problèmes afin de le résoudre.

D'une part, un expert d'un domaine X peut associer à une classe opérationnelle du graphe de classification du domaine X une *méthode de résolution composée*, c'est-à-dire une méthode qui consiste à construire des sous-problèmes, et à appeler récursivement SYRCLAD-X pour les classer et les résoudre, puis à engendrer la solution du problème à partir des solutions des sous-problèmes.

D'autre part, pour un domaine donné, il existe des problèmes que le graphe de classification ne permet pas de résoudre. Nous avons modifié le noyau de SYRCLAD afin qu'il ait la possibilité de comparer ces problèmes nouveaux au graphe de classification, pour déterminer en quoi ils sont différents des problèmes du graphe. Il construit ainsi des sous-problèmes qu'il classe et résout afin que le problème entre dans la classification. Pour qu'il puisse déterminer ces différences et construire ces sous-problèmes, il faut donner à SYRCLAD des connaissances déclaratives et des connaissances procédurales spécifiques pour chaque domaine. Nous présenterons les résultats obtenus pour les problèmes additifs et la thermodynamique.

Plan du chapitre

1 Méthodes de résolution par décomposition

Nous avons vu que l'expert d'un domaine X donne à SYRCLAD-X un graphe de classification des problèmes du domaine. A certaines classes opérationnelles, il associe une méthode de résolution qui décompose le problème en sous-problèmes, et demande de classer puis de résoudre ces sous-problèmes.

1.1 Exemples en dénombrement

En dénombrement, la classe "complémentaire" (voir figure 1) est un exemple de classe dont la méthode de résolution décompose le problème.

On classe et on résout le problème qui compte les éléments qui *ne* vérifient *pas* les contraintes de l'énoncé (problème complémentaire), et on classe et résout également le problème sans contraintes (problème complet) ; la solution de l'exercice est alors la différence entre la solution du problème complet et la solution du problème complémentaire. C'est aussi le cas de la classe "disjonction", qui considère plusieurs sous-problèmes disjoints, les classe et les résout, puis additionne leurs solutions pour obtenir la solution du problème initial (voir chapitre 4 § 2.2).

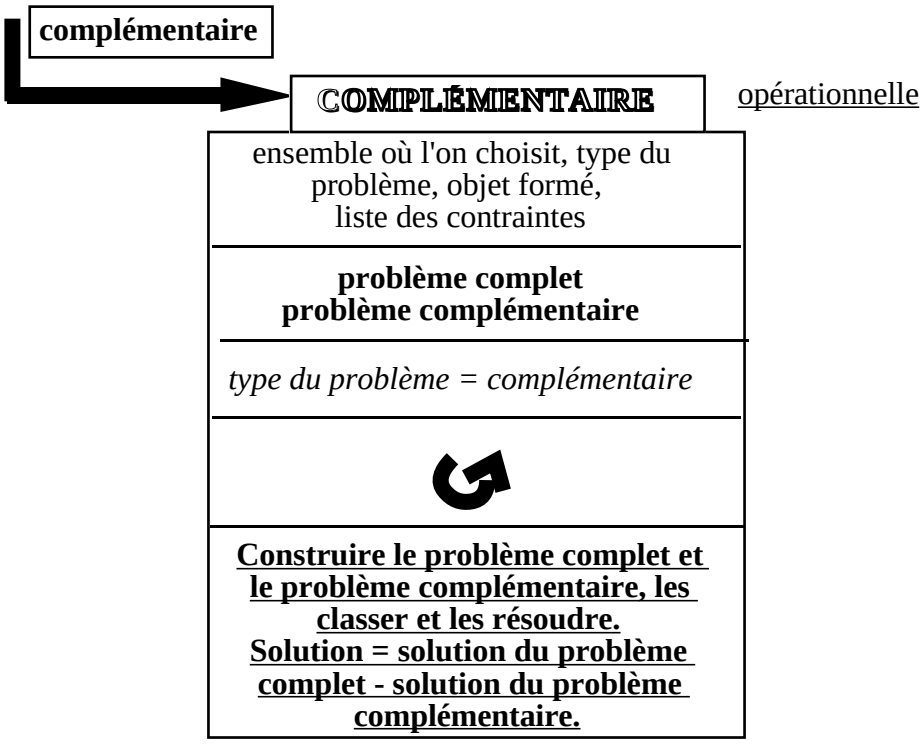


Figure 1 : la classe "complémentaire"

Voici comment la méthode de résolution associée à la classe "complémentaire" est donnée à SYRCLAD-dénombrements :

```
employer_résolution(c_complémentaire):-
            eval_prem(problème(P)),
/* pour résoudre le problème P */
            eval_prem(liste_complémentaire(P,N,M,Att,Val)),
/* on lit la valeur d'un attribut non-discriminant présent dans le modèle
opérationnel pour instancier P par rapport à la classe "complémentaire" */
            construire_pb_compl(P,N,M,Att,Val,Pc,Pall),
/* grâce aux valeurs récupérées, on construit le problème complémentaire Pc et
le problème complet Pall */
            ajouter_bf(complémentaire(P,Pall,Pc)),
            retirer_bf(problème(P)),ajouter_bf(problème(Pall)),classer(Pall),
            retirer_bf(problème(Pall)),ajouter_bf(problème(Pc)),classer(Pc),
            retirer_bf(problème(Pc),ajouter_bf(problème(P)),
/* on classe Pall et Pc */
            eval_prem(solution(Pall,S1)), /* soit S1 la solution de Pall*/
            eval_prem(solution(Pc,S2)),   /* et S2 celle de Pc */
            -(S1,S2,Sol), /* la solution Sol de P est S2-S1 */
            ajouter_bf(solution(P,Sol)),
            ajouter_bf(plan(P,[['On compte les résultats qui ne vérifient pas la
propriété demandée, puis on les soustrait du nombre total de résultats
possibles.']])),!.
```

Considérons l'exercice suivant : « Dans un sac, se trouvent 5 jetons verts et 4 jetons rouges. On tire simultanément 3 jetons dans le sac. Combien y a-t-il de tirages contenant au plus 2 jetons verts ? ».

On peut résoudre cet exercice par disjonction : on considère les trois exercices où on veut exactement 0, exactement 1, et exactement 2 jetons verts. Les règles données par l'expert à SYRCLAD-dénombrements pour déterminer la valeur du type du problème indiquent qu'il vaut mieux ici choisir la classe complémentaire, qui conduit à moins de calculs :

```
j7 : Exercice de classe "complémentaire"
Plan de résolution : On compte les résultats qui ne vérifient pas la
propriété demandée (résoudre p0 : "exactement 3 jetons verts"), puis on les soustrait du
nombre total de résultats possibles (résoudre p1 : "3 jetons").
Taille de l'ensemble où l'on choisit : 9
Type de l'objet formé : ensemble
Taille de l'objet formé : 3
Sans remise
p1 : Exercice de classe "combinaison"
Solution : C(9,3) = 84
p0 : Exercice de classe "combinaison"
Solution : C(5,3) = 10
j7 : Solution : 84 - 10 = 74
```

Quand SYRCLAD-dénombrements classe le sous-problème complémentaire, il peut trouver une classe dont la méthode de résolution effectue une décomposition du problème ; ce sera par exemple le cas pour l'exercice : « Dans un sac, se trouvent 5 jetons verts et 4 jetons rouges. On tire simultanément 5 jetons dans le sac. Combien y a-t-il de tirages contenant au plus 3 jetons verts ? »

```
j15 : Exercice de classe c_complémentaire
```

```
Plan de résolution : On compte les résultats qui ne vérifient pas la
propriété demandée (résoudre p2 : "au moins 4 jetons verts"), puis on les
soustrait du nombre total de résultats possibles (résoudre p3 : "5
jetons").
Taille de l'ensemble où l'on choisit : 9
Type de l'objet formé : ensemble
Taille de l'objet formé : 5
Sans remise
p3 : Exercice de classe "combinaison"
Solution : C(9,5) = 126
p2 : Exercice de classe "disjonction linéaire"
Plan de résolution : On envisage plusieurs cas disjoints : p4 ("exactement 5
jetons verts") et p5 ("exactement 4 jetons verts").
p4 : Exercice de classe "combinaison"
Solution : C(5,5) = 1
p5 : Exercice de classe "répartition ensemble"
Plan de résolution : On choisit 4 vert parmi 5, puis on complète en
choisissant 1 éléments parmi les 4 éléments restants.
Solution : C(5,4) * C(4,1) = 20
p2 : Solution : 20 + 1 = 21
j15 : Solution : 126 - 21 = 105
```

1.2 Exemples en thermodynamique

En thermodynamique, il existe aussi des classes dont les méthodes de résolution introduites par l'expert dans le graphe de classification décomposent le problème à résoudre. Pour calculer la variation d'énergie interne d'un système adiabatique entre deux instants, l'expert propose de calculer le travail de ce système entre ces deux instants, calcul pour lequel il a déjà explicité une méthode. Comme les échanges de chaleur sont nuls pour le système adiabatique, la variation d'énergie interne du système sera égale au travail. Cette méthode propose de résoudre une question intermédiaire pour résoudre le problème plus facilement. En effet, le sous-problème diffère du problème initial uniquement par la question posée. Il ne s'agit cependant pas uniquement d'une reformulation du problème puisqu'on a utilisé pour poser ce sous-problème des connaissances de résolution (DU = W + Q).

Ce sera une méthode analogue que SYRCLAD-thermodynamique choisit pour résoudre le problème suivant : « Un récipient de volume 10 litres contient de l'air sous la pression de 80 cm de mercure à la température de 20 °C. On fait subir à ce gaz une compression isotherme jusqu'à la pression de 800 cm de mercure. Quels sont les échanges de chaleur du gaz avec le milieu extérieur ? On assimilera l'air à un gaz parfait. On donne la masse du litre d'air normal 1,293 gr et $\gamma = 1,4$. ».

Modélis considère l'air comme un système S en transformation fermée entre les instants I et F. SYRCLAD-thermodynamique classe l'exercice et donne le plan de résolution suivant :

/* résolution du problème initial */

```
Le système S est isotherme entre les instants I et F, donc l'énergie
interne est constante, et on a Q = -W. Calculons donc le travail du système
S entre les instants I et F.
```

/* résolution du sous-problème : calculer W */

```
S est un système fermé (n=cte), et la variable température est constante
entre I et F, donc il existe une fonction f(pression,volume) constante
entre I et F. Comme on connaît pression(S,I) et volume(S,I), on connaît la
valeur de cette fonction. Comme d'autre part on connaît pression(S,F), on
peut intégrer dw en pression pour obtenir le travail du système S entre les
instants I et F.
```

1.3 Différents types de décomposition

Nous avons vu dans les exemples pécédents deux types de décomposition. En thermodynamique, les sous-problèmes consistaient à répondre à une question intermédiaire. En dénombrement, il s'agissait plutôt de problèmes auxiliaires. SCARP [Willamovski 94] utilise également la décomposition, en alternance avec des phases de classification. Il décompose des tâches en sous-tâches de plus en plus simples, la classification lui servant à choisir, suivant la forme des données, la meilleure méthode pour réaliser une tâche. Il s'agit, comme en thermodynamique, d'une décomposition du but en sous-buts ; en effet, on reste dans le même cadre, avec les mêmes données, seule la question change. En dénombrement, il s'agit pultôt d'une décomposition en problèmes auxiliaires, avec un réel changement de cadre : les sous-problèmes construits sont différents du problème initial par les contraintes sur les objets à dénombrer.

Nous avons présenté dans la première partie de ce chapitre des méthodes de résolution données par l'expert d'un domaine X à SYRCLAD-X qui les exécute pour résoudre un problème. Ces méthodes indiquent explicitement quels sous-problèmes construire et résoudre. Nous allons dans la deuxième partie de ce chapitre présenter une méthode qui permet à un SYRCLAD-X de résoudre par décomposition des problèmes qui n'appartiennent pas au graphe de classification du domaine X. Dans cette méthode, plus heuristique, l'identification du sous-problème est un (méta)problème en soi, résolu par exploration.

2 Comparer pour construire un sous-problème

Pour chaque domaine d'application X, SYRCLAD-X sait résoudre les problèmes qui entrent dans le graphe de classification que l'expert du domaine X lui a donné. Il existe presque toujours (sauf au tarot) des problèmes du domaine qui ne relèvent pas de cette classification. On peut choisir d'augmenter le graphe de classification, ou de donner plus de souplesse à SYRCLAD-X pour qu'il puisse traiter certains de ces nouveaux problèmes en les comparant à ceux qu'il sait résoudre.

Prenons le cas des problèmes additifs. Dans le cas d'une partie de billes, pour chaque participant il y a trois données : le nombre de billes qu'il possède avant la partie, le nombre de billes qu'il a gagnées ou perdues pendant la partie, et le nombre de billes qu'il a après la partie. La classification donnée à SYRCLAD-additifs lui permet de déterminer une de ces trois données à partir des deux autres. S'il y a deux parties de billes successives, on considère pour un participant le nombre de billes perdues ou gagnées pendant la première partie, pendant la deuxième partie, et en tout. La classification permet de déterminer une de ces données à partir des deux autres (cf. chapitre 5).

Supposons à présent que l'on présente à SYRCLAD-additifs le problème suivant : « Jean a 5 billes. Il joue deux parties de billes. Il perd 2 billes pendant la première partie, et gagne 4 billes pendant la deuxième partie. Combien a-t-il de billes à la fin ? ». Ce problème n'entre pas dans la classification donnée, mais on attend d'un élève qu'il le résolve de la manière suivante : on calcule combien de billes Jean a gagnées ou perdues en tout au cours des deux parties. On est alors ramené, pour calculer le nombre de billes qu'il a à la fin, à un problème avec une seule partie de billes. Cette méthode consiste à résoudre un sous-problème afin de se retrouver dans un cas connu.

Nous présenterons d'abord comment nous avons modifié le noyau de SYRCLAD en y ajoutant une architecture de méthode qui permet à un SYRCLAD-X de résoudre par décomposition des problèmes qu'il ne sait pas résoudre. Nous détaillerons ensuite les connaissances déclaratives et procédurales que nous avons données à SYRCLAD-additifs et à SYRCLAD-thermodynamique afin d'appliquer cette architecture de méthode à ces deux domaines.

2.1 La méthode générale

Nous voulons permettre à un SYRCLAD-X de résoudre des problèmes que le graphe de classification ne lui a pas permis de résoudre. Il s'agit de problèmes pour lesquels le classement n'a pas débouché sur une classe opérationnelle. Nous avons donc modifié l'algorithme de classement utilisé par le noyau de SYRCLAD en ajoutant, en cas de classement terminé dans une classe non-opérationnelle, un appel à une procédure *décomposer(Problème).*

La méthode implémentée dans cette procédure consiste à introduire et à résoudre un sous-problème afin de se retrouver dans un cas connu. Il faut donc trouver un sous-problème dont la résolution permettrait de se trouver dans une situation connue. Pour cela, on compare le problème aux classes opérationnelles du graphe afin de trouver une classe proche du problème et d'établir la différence qui fait que le problème n'appartient pas à cette classe. Le sous-problème à résoudre sera de combler cette différence.

La méthode générale est donc la suivante :

Si le classement d'un problème échoue, on essaie de le décomposer :

- comparer le problème aux classes opérationnelles :
    - trouver une classe proche
    - trouver "ce qui manque" pour que le problème appartienne à la classe
- classer et résoudre le sous-problème consistant à trouver "ce qui manque"
- essayer de résoudre le problème initial ainsi modifié

La démarche consistant à chercher une différence entre la situation actuelle et la situation désirée pour poser comme sous-problème de combler cette différence est celle utilisée par les systèmes historiques GPS [Newell et al 60], STRIPS [Fikes et Nilsson 69] et leurs descendants. SYRCLAD

utilise cette heuristique dans le cadre d'une classification de problèmes : au lieu de comparer un état aux prémisses d'une règle, il compare un problème à une classe de problèmes.

Pour comparer un problème aux classes opérationnelles du graphe de classification, il faut définir dans chaque domaine d'application des *éléments de comparaison.* L'expert devra alors enrichir le graphe de classification de connaissances déclaratives sur lesquelles pourra se faire la comparaison, c'est-à-dire donner les valeurs des éléments de comparaison pour chaque classe opérationnelle.

La construction des sous-problèmes sera aussi dépendante du domaine. Cette méthode conduit à construire de nouveaux objets qui interviennent dans le sous-problème, ces objets étant eux aussi spécifiques au domaine.

Par rapport à la méthode de résolution par classification de SYRCLAD, cette méthode de décomposition travaille à un niveau méta. Il s'agit en effet pour le système de savoir ce qu'il est capable de faire grâce à la classification, et d'essayer de s'y ramener.

Voici la procédure *décomposer(Problème)* ajoutée au noyau de SYRCLAD :

```
décomposer(Problème):-
/* pour décomposer un problème dont le classement a échoué : */
 chercher_éléments_comparaison(Problème,Liste_éléments),
/* déterminer la valeur pour ce problème des éléments de comparaison
définis pour le domaine */
 chercher_classe_proche(Liste_éléments,Type_différence),
/* chercher dans le graphe de classification une classe proche par rapport
aux éléments de comparaison du problème, et établir le type de la
différence */
 construire_sous_problème(Problème,Type_différence,Sous_problème),
/* construire un sous-problème qui permette de réduire cette différence */
 classer(Sous_problème),
/* classer et résoudre ce sous-problème */
 reconstruire(Sous_problème,Problème),
/* la solution du sous-problème modifie l'énoncé du problème initial */
 classer(Problème).
/* on peut à présent essayer de classer et de résoudre le problème modifié
```

Cette procédure est indépendante du domaine, mais chacune des sous-procédures qui la constituent (à part *classer*) est spécifique au domaine. Comme elle est implémentée en Prolog, il peut y avoir retour arrière en cas d'échec : si on n'arrive pas à combler la différence avec une classe, on essaie avec une autre ; si on n'arrive pas à résoudre un sous-problème, on essaie d'en construire un autre.

Pour appliquer cette méthode générale à un domaine X, l'expert du domaine X doit d'une part enrichir le graphe de classification de connaissances déclaratives en indiquant les valeurs des éléments de comparaison qu'il a choisis pour le domaine X et d'autre part fournir des connaissances procédurales en définissant les sous-procédures qui composent la méthode.

Pour le domaine des dénombrements et du tarot, ces sous-procédures ne sont pas définies, et la procédure *décomposer(Problème)* échoue. Nous allons à présent préciser comment sont définies ces sous-procédures pour le domaine des problèmes additifs et de la thermodynamique

2.2 Application aux problèmes additifs

2.2.1 La méthode spécialisée aux problèmes additifs

Nous donnons à nouveau en figure 2 la classification des problèmes additifs que SYRCLAD-additifs utilise. Pour tous les problèmes de cette classification, il y a deux données et une question. Il peut s'agir d'un ensemble de billes possédé par quelqu'un à un moment donné, ou d'un opérateur correspondant à un gain ou une perte de billes. Lorsqu'un problème n'entre pas dans cette classification, il faut le comparer aux classes existantes. Nous avons choisi de faire cette comparaison sur le type (ensemble de billes ou opérateur) des données et de la question. Le graphe de classification a donc été enrichi des valeurs des éléments de comparaison pour les classes opérationnelles ; par exemple la classe 2.1 prend en données un ensemble et un opérateur et permet de calculer un ensemble :

```
type_résultat(c_simple_ajout_res,ensemble).
type_données(c_simple_ajout_res,[ensemble,opérateur]).
```

	Opérateurs	Types de problèmes	Solutions
1.1	ADD	ADD : (a, b) -> ?	? = a + b
1.2		ADD : (a, ?) -> b	? = b - a
2.1	AJOUT n	AJOUT b : a -> ?	? = a + b
2.2		AJOUT b : ? -> a	? = a - b
2.3		AJOUT ? : a -> b	? = b - a
3.1	RET n	RET b : a -> ?	? = a - b
3.2		RET b : ? -> a	? = a + b
3.3		RET ? : a -> b	? = a - b

	Types de problèmes	Solutions
4.1	AJOUT a o AJOUT b = ?	? = AJOUT (a + b)
4.2	? o AJOUT b = AJOUT a	
4.2.1	avec a > b	? = AJOUT (a - b)
4.2.2	avec a < b	? = RET (b - a)
5.1	RET a o RET b = ?	? = RET (a + b)
5.2	? o RET b = RET a	
5.2.1	avec a > b	? = RET (a - b)
5.2.2	avec a < b	? = AJOUT (b - a)
6.1	AJOUT a o RET b = ?	
6.1.1	avec a > b	? = AJOUT (a - b)
6.1.2	avec a < b	? = RET (b - a)
6.2.1	? o RET b = AJOUT a	? = AJOUT (a + b)
6.2.2	? o AJOUT b = RET a	? = RET (a + b)

Figure 2 : Classification des problèmes additifs

Reprenons chacune des sous-procédures de la méthode générale et voyons comment elle est appliquée aux problèmes additifs.

```
chercher_éléments_comparaison(Problème,Liste_éléments)
```

On cherche le type des données du problème et de la question : ensemble ou opérateur. Pour trouver le type des données, on utilise un paquet de règles qui permet de recenser toutes les données présentes dans un problème, et on note leur type. On dispose également d'un ensemble de règles qui concluent sur le type de la question.

Ces règles sont des règles si-alors identiques aux règles de reformulation. Elles sont utilisées par le même moteur d'inférence : les règles d'un paquet de règles sont saturées alors que dans un ensemble de règles, une seule est déclenchée.

Liste_éléments = [Liste des types des données, Type du résultat]

Exemple : Liste_éléments = [[ensemble, opérateur, opérateur], ensemble]

```
chercher_classe_proche(Liste_éléments,Type_différence)
```

Si une classe du graphe de classification permet de calculer un résultat du même type que celui de la question, on compare le type des données de la classe et le type des données du problème afin de déterminer le type de la donnée manquante.

Type_différence = opérateur ou ensemble

```
construire_sous_problème(Problème,Type_différence,Sous_problème)
```

Il faut alors construire un sous-problème dont la question est de ce type. Pour définir la question du sous-problème, il faut créer un objet qui n'est pas défini dans le problème. On suppose en effet que les objets introduits dans l'énoncé sont soit des données, soit la question. On doit donc créer un nouvel ensemble de billes, la question posée par le sous-problème sera de calculer la taille de cet ensemble.

Si le type de la donnée manquante est un ensemble, on peut créer l'ensemble de billes possédé par un ensemble de personnes à un instant donné ; il faut alors préalablement créer cet ensemble de personnes s'il n'est pas défini. Il peut aussi s'agir d'un ensemble possédé par une personne à un instant donné, cette personne faisant partie d'un ensemble de personnes. On peut également créer un ensemble de billes possédé à un instant donné par un participant à une partie ou à une succession de parties.

Si le type de la donnée manquante est un opérateur, on peut créer dans une partie ou une succession de parties l'ensemble gagné ou perdu par un participant. S'il s'agit d'une succession de parties, on peut avoir à créer préalablement une "sous-succession" de parties.

Le sous-problème est construit en créant un objet sur lequel porte la question. En introduisant cet objet, on définit également les données du sous-problème en conséquence. Ces données sont souvent une partie des données du problèmes restructurées en tenant compte du nouvel objet introduit.

```
classer(Sous_problème)
```

Le sous-problème ainsi construit est classé et résolu par SYRCLAD-additifs comme n'importe quel autre problème. S'il n'appartient pas à la classification, il peut donc à son tour être décomposé.

```
reconstruire(Sous_problème,Problème)
```

Une fois le sous-problème résolu, il faut reconstruire le problème initial. Il s'agit de restructurer l'énoncé du problème en y ajoutant la solution du sous-problème tout en enlevant les données du sous-problème, pour se retrouver dans un cas plus simple.

```
classer(Problème)
```

Enfin, on réessaie de classer et de résoudre le problème.

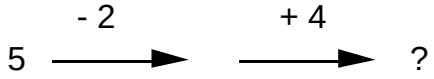
La méthode spécialisée aux problèmes additifs sera donc la suivante :

- Trouver le type des données du problème et de la question : ensemble ou opérateur
- Chercher une classe opérationnelle qui permet de trouver un résultat de ce type et trouver quel type de donnée manque pour que le problème appartienne à cette classe
- Construire un sous-problème dont la question est du type de la donnée manquante
- Classer et résoudre le sous-problème
- Reconstruire le problème
- Classer et résoudre le problème

Nous allons maintenant étudier quelques exemples de problèmes que SYRCLAD-additifs résout en utilisant cette méthode de décomposition.

2.2.2 Résultats

Exercice 2pRes : « Jacques a 5 billes. Il joue deux parties de billes. Il perd 2 billes pendant la première partie, et gagne 4 billes pendant la deuxième partie. Combien a-t-il de billes à la fin ? »




```
problème(b2pRes).
action(b2pRes,p).
est_un(p,succession_de_parties).
participants(p,jacques,i).
est_un(jacques,personne).
nombre_de_parties(p,2).
succession(p,[[p1,t1,t2],[p2,t2,t3]]).
possède(jacques,t1,b2).
est_un(b2,ensemble).
tout_élément(b2,w).
est_un(w,bille).
taille(b2,5).
perdu(jacques,p1,b5).
est_un(b5,ensemble).
tout_élément(b5,y).
est_un(y,bille).
taille(b5,2).
gagné(jacques,p2,b4).
est_un(b4,ensemble).
tout_élément(b4,x).
est_un(x,bille).
taille(b4,4).
possède(jacques,t3,bc).
est_un(bc,ensemble).
tout_élément(bc,z).
est_un(z,bille).
taille(bc,c).
à_calculer(b2pRes,c).
```

SYRCLAD-additifs essaie de classer le problème :

```
b2pRes
```

```
nature_composé op : init i2 op : ajout op : ret top2
Classification bloquée : c_composé_typediff
problème composé : deux parties, opérateurs de types différents
```

Le classement s'arrête dans une classe non opérationnelle. SYRCLAD-additifs essaie alors de décomposer le problème b2pRes :

```
Essai de décomposition
```

Il détermine que les données sont un ensemble et deux opérateurs, et que l'on veut trouver un ensemble :

```
données : init données : ens données : op données : op résultat3
données : [opérateur, opérateur, ensemble]
question : ensemble
```

Les classes permettant de trouver un ensemble sont les classes 1.1, 1.2, 2.1, 2.2, 3.1, 3.2 :

```
[c_simple_ret_opérande, c_simple_ret_res, c_simple_ajout_opérande,
c_simple_ajout_res, c_simple_add_opérande, c_simple_add_res]
```

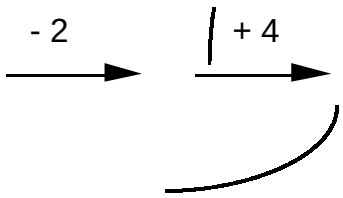
Les classes 1.1 et 1.2 ne conviennent pas, mais la classe 2.1 est plus proche, puisqu'elle permet de trouver un ensemble à partir d'un ensemble et d'un opérateur. On a déjà un ensemble, la donnée manquante est alors du type opérateur, que l'on va essayer de déterminer avec les deux opérateurs donnés.

```
c_simple_ret_opérande
donnée manquante : opérateur
```

chgt_parties

SYRCLAD-additifs crée donc l'opérateur correspondant aux deux parties et le sous-problème consistant à trouver cet opérateur :

```
problème(pr0)
action(pr0, p)
est_un(ens0, ensemble)
tout_élément(ens0, elt0)
est_un(elt0, bille)
taille(ens0, taille0)
changement(jacques, p, ens0)
```

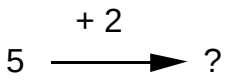


Il classe ce sous-problème qu'il trouve de classe 6.1.1 et obtient AJOUT 2. :

```
pr0
nature_composé op : init i3 op : ajout op : ret top2 at8 s1 sol15
pr0
Exercice de classe c_composé_typediff_double_pos
Solution : jacques gagné 2
```

Il reconstruit le problème en remplaçant les deux parties par la partie résultante :

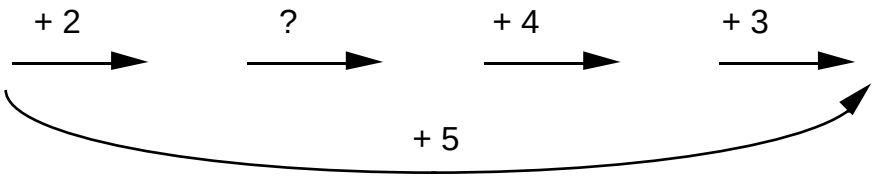
```
nettoyer_op1

problème(b2pRes)
est_un(p, partie)
partie(p, t1, t3)
gagné(jacques, p, ens0)
taille(ens0, 2)
```



puis il le classe. Il obtient un problème de classe 2.1 dont de résultat est 7 :

```
b2pRes
nature_simple1 op_ajout1 at3 sol3
b2pRes
Exercice de classe c_simple_ajout_res
Solution : 7
```

Exercice b4p2e : « Jacques joue 4 parties de billes. A la première partie il a gagné 2 billes. On ne sait pas ce qui s'est passé pendant la deuxième partie. A la troisième partie il a gagné 4 billes, et à la quatrième il a gagné 3 billes. En tout, il a gagné 5 billes. Que s'est-il passé pendant la deuxième partie ? ».




```
problème(b4p2e).
action(b4p2e,p).
est_un(p,succession_de_parties).
participants(p,jacques,i).
est_un(jacques,personne).
nombre_de_parties(p,4).
succession(p,[[p1,t1,t2],[p2,t2,t3],
[p3,t3,t4],[p4,t4,t5]]).
gagné(jacques,p1,b2).
est_un(b2,ensemble).
tout_élément(b2,w).
est_un(w,bille).
taille(b2,2).
changement(jacques,p2,bat).
est_un(bat,ensemble).
tout_élément(bat,y).
est_un(y,bille).
taille(bat,t).
gagné(jacques,p3,b4).
est_un(b4,ensemble).
tout_élément(b4,x).
est_un(x,bille).
taille(b4,4).
gagné(jacques,p4,b3).
est_un(b3,ensemble).
tout_élément(b3,v).
est_un(v,bille).
taille(b3,3).
gagné(jacques,p,b5).
est_un(b5,ensemble).
tout_élément(b5,z).
est_un(z,bille).
taille(b5,5).
à_calculer(b4p2e,t).
```

SYRCLAD-additifs échoue dans le classement du problème b4p2e. Il essaie alors de le décomposer.

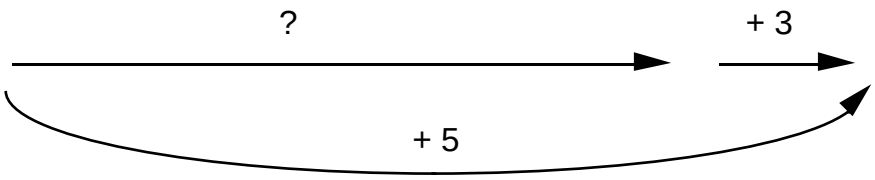
Il y a 4 données de type opérateur et le résultat est de type opérateur :

```
données : [opérateur, opérateur, opérateur, opérateur]
question : opérateur
```

Les classes qui concluent sur un opérateur sont les classes du deuxième tableau et les classes 2.3 et 3.3.

La donnée manquante est de type opérateur. Le système crée la partie [part1,t1,t4] et pose le sous-problème pr1 dont la question est de calculer le changement de billes de Jacques pendant cette partie.

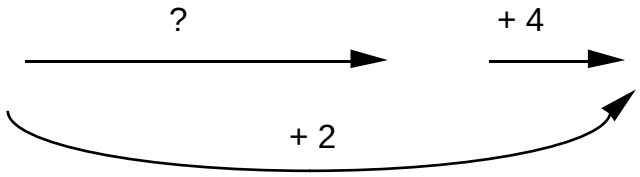
```
problème(pr1)
action(pr1, p)
nombre_de_parties(p, 2)
succession(p, [[part1, t1, t4], [p4, t4, t5]])
est_un(ens1, ensemble)
tout_élément(ens1, elt1)
est_un(elt1, bille)
taille(ens1, taille1)
changement(jacques, part1, ens1)
à_calculer(pr1, taille1)
```



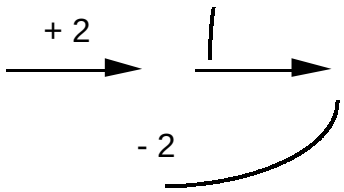
SYRCLAD-additifs classe pr1 et trouve AJOUT 2.

Il reconstruit le problème b4p2e en considérant uniquement les trois premières parties, mais échoue à nouveau dans le classement. Il décompose à nouveau en créant la partie [part2,t1,t3], le sous-problème pr2 consiste à trouver le changement de billes de Jacques pendant cette partie.

```
problème(pr2)
action(pr2, part1)
nombre_de_parties(part1, 2)
succession(part1, [[part2, t1, t3], [p3, t3, t4]])
est_un(ens2, ensemble)
tout_élément(ens2, elt2)
est_un(elt2, bille)
taille(ens2, taille2)
changement(jacques, part2, ens2)
à_calculer(pr2, taille2)
```



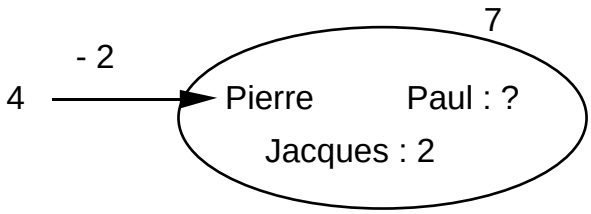
SYRCLAD-additifs classe pr2 et trouve RET 2. Il reconstruit b4p2e en considérant uniquement les deux premières parties.



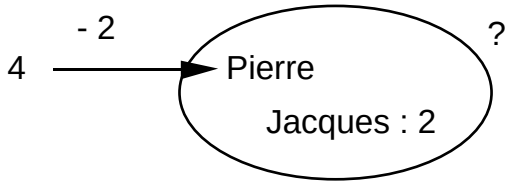
Le classement de b4p2e est alors un succès et il trouve RET 4 :

```
b4p2e
Exercice de classe c_composé_typediff_un_ret
Solution : jacques perdu 4
```

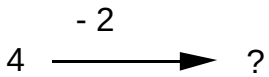
Exercice aj-3addOp : « Pierre a 4 billes. Il en perd 2. Sachant qu'alors Pierre, Jacques et Paul ont ensemble 7 billes et que Jacques en a 2, combien Paul a-t-il de billes ? ».



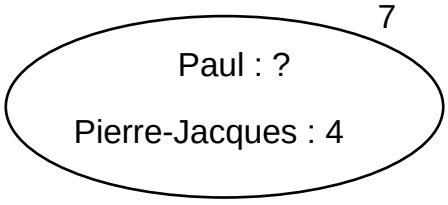
SYRCLAD-additifs cherche à résoudre le sous-problème pr1 consistant à calculer le nombre de billes qu'ont ensemble Pierre et Jacques :



Cela le conduit à poser pr2 comme sous-problème de pr1, pr2 consistant à calculer le nombre de billes de Pierre :



Le problème pr2 est de classe 3.1 et sa solution est 2. SYRCLAD-additifs peut alors résoudre pr1 qui est de classe 1.1 et dont la solution est 4. Enfin il peut résoudre le problème initial en considérant le sous-ensemble Pierre et Jacques comme une seule personne, ce problème est de classe 1.2 et sa solution est 3.



L'utilisation par SYRCLAD-additifs de cette méthode de décomposition lui permet de résoudre une vingtaine de nouveaux exercices variés.

2.3 Application à la thermodynamique

2.3.1 La méthode spécialisée à la thermodynamique

La méthode générale de décomposition de problèmes a aussi été appliquée à la thermodynamique. Dans ce domaine, la comparaison aux classes opérationnelles se fait sur le nombre de systèmes et le nombre d'instants du problème. En effet, les nouveaux exercices traités par décomposition sont des successions de transformations ou des mélanges de plusieurs gaz. On a donc ajouté au graphe de classification les connaissances déclaratives quant au nombre de systèmes et d'instants d'un problème d'une classe opérationnelle :

```
nombre_systèmes(c_1s_2i_cons,1).
nombre_instants(c_1s_2i_cons,2).
```

Reprenons chacune des sous-procédures de la méthode générale et voyons comment elle est appliquée aux problèmes additifs.

```
chercher_éléments_comparaison(Problème,Liste_éléments),
```

Les éléments de comparaison étant le nombre de systèmes et d'instants du problème, ils sont donnés dans le modèle descriptif du problème.

```
chercher_classe_proche(Liste_éléments,Type_différence)
```

On compare les nombres de systèmes et d'instants du problème avec ceux d'une classe opérationnelle et on en déduit une différence soit par le nombre de systèmes soit par le nombre d'instants (nous n'avons pas traité les cas où les deux diffèrent).

```
construire_sous_problème(Type_différence,Sous_problème)
```

On construit un sous-problème qui consiste à chercher une grandeur soit sur un système soit à un instant.

```
classer(Sous_problème)
```

On essaie de classer et résoudre le sous-problème.

```
reconstruire(Sous_problème,Problème)
```

La solution au sous-problème vient s'ajouter aux données du problème.

```
classer(Problème)
```

On essaie à nouveau de classer le problème.

La méthode spécialisée à la thermodynamique sera donc la suivante :

Si le classement d'un problème échoue, on essaie de le décomposer :

- comparer le nombre de systèmes et d'instants du problème aux classes opérationnelles

- trouver quel type de donnée manque pour que le problème appartienne à une classe : grandeur à un instant ou grandeur sur un système

- construire un sous-problème dont la question est du type déterminé

- classer et résoudre le sous-problème

- reconstruire le problème

- classer et résoudre le problème

Succession de transformations

Si le problème est une succession de deux transformations, il y a un système, les données sont des grandeurs sur un système à 3 instants et la question une grandeur sur le même système à un instant. La classe la plus proche est la classe un_système-deux_instants (avec ou sans conservation) dont les données sont des grandeurs à 2 instants et la question une grandeur à un instant. Il faut alors déterminer une donnée manquante du type : grandeur à un instant. SYRCLAD-thermodynamique considère un sous-problème où il faut trouver une grandeur thermodynamique (on se restreint en fait à P, V ou T) à un instant voisin (en fait l'instant précédant) de l'instant où la question est posée. Tout se passe dans le sous-problème comme si l'on considérait uniquement la première transformation. SYRCLAD-thermodynamique classe et résout le sous-problème en utilisant uniquement les classes débouchant sur un plan de résolution (étiquetées comme telles). La reconstruction consiste à ajouter la grandeur calculée et à considérer alors uniquement la deuxième transformation.

Cette décomposition est analogue à la décomposition d'une succession de parties de billes. Elle pourra être faite sur une succession de plus de deux transformations.

Mélanges de gaz

Il existe dans la classification des problèmes de thermodynamique une classe correspondant aux problèmes où l'on mélange deux gaz. Cette classe a comme données des grandeurs sur deux systèmes à un instant et sur un troisième système (le mélange) à un autre instant, la question est une grandeur sur un système (le mélange) à un instant. Face à un problème où l'on considère un mélange de plus de deux gaz, SYRCLAD-thermodynamique compare le problème à cette classe et considère qu'il manque une donnée du type : grandeur sur un système. Il construit alors un nouveau système : un "sous-mélange". La question du sous-problème est une grandeur sur ce sous-mélange (en fait le nombre de moles (n) ou la masse (m) ou les deux). Tout se passe dans le sous-problème comme si l'on considérait uniquement la formation du sous-mélange. Le sous-problème est classé et résolu par SYRCLAD-thermodynamique. La reconstruction consiste à ajouter la (ou les) grandeur(s) calculée(s) et à considérer alors le sous-mélange comme un seul gaz.

Cette décomposition est analogue à celle qu'utilisait SYRCLAD-additif pour les ensembles de personnes et l'opérateur ADD. Elle pourra être faite sur un mélange de plus de trois gaz.

Décomposition locale

SYRCLAD a également la possibilité en thermodynamique de décomposer des problèmes à un niveau plus local. En effet, certaines classes du graphe de classification correspondent à des cas particuliers et permettent de produire un plan de résolution. Considérons la définition d'un de ces cas particuliers : "la transformation est adiabatique, on connaît γ, parmi les grandeurs P, V, T, on en connaît deux à l'instant initial, l'une d'entre elles à l'instant final, et on demande l'autre". Supposons à présent que l'on pose à SYRCLAD-thermodynamique un exercice qui satisfait la définition de ce cas particulier, sauf que l'on ne connaît pas γ, mais Cp et Cv. Le classement donne un système en transformation entre deux instants, avec conservation d'une grandeur. Mais l'exercice n'entre pas dans le cas particulier précédent puisque l'on n'a pas γ. Or il suffit de calculer γ à partir de Cp et Cv. SYRCLAD-thermodynamique a donc la possibilité pour tous les cas particuliers de la classification de comparer le problème posé à la définition du cas particulier, pour éventuellement déterminer quelle est la donnée manquante (ici γ), puis il pose le sous-problème consistant à calculer cette donnée.

Cette méthode est l'application de la méthode générale de décomposition de problèmes à un niveau local, le (meta)problème initial étant "descendre dans un cas particulier". Il ne s'agit pas d'un problème du domaine et il n'y a pas lieu de le comparer à un graphe de classification. La procédure décomposer(Problème) n'est donc pas utilisée. La démarche relève pourtant de la même idée :

Si l'on ne descend pas dans un cas particulier :

- comparer le problème à la définition des cas particuliers pour trouver la donnée manquante
- classer et résoudre le sous-problème consistant à trouver cette donnée manquante
- si le sous-problème est résolu, descendre dans le cas particulier

Dans cette application de la méthode générale de décomposition, le sous-problème doit être résolu directement. En cas d'échec, il ne faut pas essayer de le décomposer, sous peine de bouclage possible.

Nous allons maintenant étudier quelques exemples de problèmes que SYRCLAD-thermodynamique résout en utilisant ces méthodes de décomposition.

2.3.2 Résultats

Succession de deux transformations

« Un gaz parfait subit deux transformations isochores. La première transformation le fait passer de l'état (P1,T1) à l'état (P2,T2). La deuxième de l'état (P2,T2) à l'état (P3,T3). Exprimez P3 en fonction de T1, P1, P2 et T3. »

```
problème(succ2).
systèmes(succ2,[gaz_parfait]).
instants(succ2,[i1,i2,i3]).
gp(gaz_parfait).
trans_gp_fermé(gaz_parfait,i1,i2).
trans_gp_fermé(gaz_parfait,i2,i3).
isochore(gaz_parfait,i1,i2).
isochore(gaz_parfait,i2,i3).
température(gaz_parfait,i1).
```

```
pression(gaz_parfait,i1).
pression(gaz_parfait,i2).
température(gaz_parfait,i3).
on_demande(succ2,[pression(gaz_parfait,i3)]).
```

SYRCLAD-thermodynamique essaie de classer le problème succ2 :

```
succ2
nb_sys nb_ins
Classification bloquée
c_1s
un système
```

Il compare le nombre de systèmes et d'instants du problème pour les comparer aux classes :

```
1 systèmes
3 instants
```

Le type de la donnée manquante est grandeur à un instant. Il construit pr0 :

```
sous problème pr0 : trouver volume(gaz_parfait, i2)
```

Il essaie de classer pr0 :

```
pr0
nb_sys nb_ins isochore
c_1s_2i_cons
```

Le sous-problème ne se résout pas avec un plan (on ne peut pas calculer de volume dans ce problème), SYRCLAD-thermodynamique choisit alors une autre grandeur :

```
sous problème pr0 : trouver température(gaz_parfait, i2)
```

Il classe le nouveau sous-problème pr0 :

```
pr0
cas_part_inv
c_1s_2i_cons_inv
```

et donne la solution :

```
gaz_parfait est un système fermé (n=cte), et la variable volume est
constante entre i1 et i2, donc il existe une fonction f(pression,
température) constante entre i1 et i2. Comme on connaît
pression(gaz_parfait,i1) et température(gaz_parfait,i1), on connaît la
valeur de cette fonction. Comme d'autre part on connaît
pression(gaz_parfait,i2), on en déduit température(gaz_parfait,i2).
```

SYRCLAD-thermodynamique reconstruit le problème succ2 en ajoutant T2 et en considérant uniquement la deuxième transformation. Il classe succ2 :

```
succ2
nb_sys nb_ins isochore cas_part_inv
c_1s_2i_cons_inv
```

et donne la solution :

```
gaz_parfait est un système fermé (n=cte), et la variable volume est
constante entre i2 et i3, donc il existe une fonction f(pression,
température) constante entre i2 et i3. Comme on connaît
pression(gaz_parfait,i2) et température(gaz_parfait,i2), on connaît la
valeur de cette fonction. Comme d'autre part on connaît
température(gaz_parfait,i3), on en déduit pression(gaz_parfait,i3).
```

Succession de trois transformations

« Un gaz parfait subit trois transformations successives. Ces trois transformations sont isothermes. Elles font passer le système de l'état (P1,V1) à (P2,V2) puis (P3,V3) et enfin (P4,V4). P1 = 1 atm, V1 = 3 l, V2 = 1 l, P3 = 3 atm, P4 = 2 atm. Calculez V4. »

SYRCLAD-thermodynamique essaie de classer le problème succ7 :

```
succ7
 nb_sys nb_ins
Classification bloquée
c_1s
un système
```

Pour le décomposer, il compare le nombre d'instants et le nombre de systèmes aux classes du graphe.

```
1 systèmes
4 instants
```

Il construit le sous-problème pr1 :

```
sous problème pr1 : trouver volume(gaz_parfait, i3)
```

Il essaie de classer pr1 :

```
pr1
nb_sys nb_ins
Classification bloquée
c_1s
un système
```

Il décompose pr1 :

```
1 systèmes
3 instants
sous problème  pr2 : trouver pression(gaz_parfait, i2)
```

Il classe le sous-problème pr2 :

```
pr2
nb_sys nb_ins isotherme cas_part_inv
c_1s_2i_cons_inv
```

et donne la solution de pr2 :

```
gaz_parfait est un système fermé (n=cte), et la variable température est
constante entre i1 et i2, donc il existe une fonction f(pression,volume)
constante entre i1 et i2. Comme on connaît pression(gaz_parfait,i1) et
volume(gaz_parfait,i1), on connaît la valeur de cette fonction. Comme
d'autre part on connaît volume(gaz_parfait,i2), on en déduit
pression(gaz_parfait,i2).
```

Maintenant que P2 est connue, il peut résoudre pr1 :

```
pr1
nb_sys nb_ins isotherme cas_part_inv
c_1s_2i_cons_inv
gaz_parfait est un système fermé (n=cte), et la variable température est
constante entre i2 et i3, donc il existe une fonction f(pression,volume)
constante entre i2 et i3. Comme on connaît pression(gaz_parfait,i2) et
volume(gaz_parfait,i2), on connaît la valeur de cette fonction. Comme
```

```
d'autre part on connaît pression(gaz_parfait,i3), on en déduit
volume(gaz_parfait,i3).
```

pr1 résolu, il termine la résolution de succ7 :

```
succ7
nb_sys nb_ins isotherme cas_part_inv
c_1s_2i_cons_inv
gaz_parfait est un système fermé (n=cte), et la variable température est
constante entre i3 et i4, donc il existe une fonction f(pression,volume)
constante entre i3 et i4. Comme on connaît pression(gaz_parfait,i3) et
volume(gaz_parfait,i3), on connaît la valeur de cette fonction. Comme
d'autre part on connaît pression(gaz_parfait,i4), on en déduit
volume(gaz_parfait,i4).
```

Mélange de trois gaz

«On considère 3 gaz g1, g2, g3. On connaît P1, T1, V1, P2, T2, V2, P3, T3. On mélange ces trois gaz, on obtient 3 litres en conditions normales. Calculez V3. »

```
problème(mél3).
systèmes(mél3,[s1,s2,s3]).
instants(mél3,[i,f]).
équi_gp(s1,i).
volume(s1,i).
température(s1,i).
pression(s1,i).
équi_gp(s2,i).
volume(s2,i).
température(s2,i).
pression(s2,i).
équi_gp(s3,i).
température(s3,i).
pression(s3,i).
mélange([s1,s2,s3],i,mel,f).
équi_gp(mel,f).
volume(mel,f).
température(mel,f).
pression(mel,f).
on_demande(mél3,[volume(s3,i)]).
```

SYRCLAD-thermodynamique essaie de classer le problème mél3 :

```
mél3
nb_sys
Classification bloquée
c_problème
```

Il décompose le problème en comparant le nombre de systèmes et d'instants aux classes du graphe :

```
nb_ins
3 systèmes
2 instants
```

Il en déduit que la donnée manquante est de type : grandeur sur un système, et construit le sous-problème pr3 :

```
sous problème  pr3 : mel0 = mélange de  [s1, s2], trouver n(mel0,f) ou
masse(mel0, f)
```

Il classe pr3 :

```
pr3
nb_sys nb_ins comp3 sans
c_2s_2i_mélange_sans
```

Il produit le plan suivant (réécrit):

```
P1 + V1 + T1 => n1 (PV=nRT)
n1 + n2 => nmel0 (quand on mélange deux gaz, les nombres de moles
s'ajoutent)
```

SYRCLAD-thermodynamique reconstruit le problème mél3 en considérant que l'on mélange mel0 et s3. Il classe mél3 :

```
mél3
nb_sys nb_ins comp3 sans
c_2s_2i_mélange_sans
mél3
```

Il produit le plan de résolution suivant (réécrit) :

```
Pmel + Vmel + Tmel => nmel (PV=nRT)
nmel0 + nmel => n3 (quand on mélange deux gaz, les nombres de moles
s'ajoutent)
P3 + T3 + n3 => V3 (PV=nRT)
```

Mélange de quatre gaz

« On mélange une mole d'azote avec une mole de méthane, une mole de propane et une mole d'oxygène. Quelle est la masse molaire du gaz ainsi obtenu ? »

```
systèmes(mél8,[s1,s2,s3,s4]).
instants(mél8,[i,f]).
équi_gp(s1,i).
n(s1,i).
composition(s1).
équi_gp(s2,i).
n(s2,i).
composition(s2).
équi_gp(s3,i).
n(s3,i).
composition(s3).
équi_gp(s4,i).
n(s4,i).
composition(s4).
mélange([s1,s2,s3,s4],i,mel,f).
équi_gp(mel,f).
on_demande(mél8,[masse_molaire(mel)]).
```

SYRCLAD-thermodynamique essaie de classer ce problème mél8 :

```
mél8
nb_sys
Classification bloquée
c_problème
```

Il décompose alors mél8 en comparant le nombre d'instants et le nombre de systèmes avec les clases du graphe :

```
nb_ins
4 systèmes
2 instants
```

Il construit le sous-problème pr4 :

```
sous problème pr4 : mel1 = mélange de  [s2, s3, s4] , trouver  n(mel1, f)
ou masse(mel1, f)
```

Il essaie de classer pr4 :

```
pr4
nb_sys
Classification bloquée
c_problème
```

Il décompose alors pr4 :

```
nb_ins
3 systèmes
2 instants
sous problème pr5 : mel2  = mélange de [s3, s4], trouver n(mel2, f) ou
masse(mel2, f)
pr5
nb_sys nb_ins comp3 sans
c_2s_2i_mélange_sans
pr5
n3 + n4 => nmel2
composition(s3) => M3
n3 + M3 => m3
composition(s4) => M4
n4 + M4 => m4
m3 + m4 => mmel2 (quand on mélange deux gaz, les masses s'ajoutent)
```

Il peut maintenant résoudre pr4 :

```
pr4
nb_sys nb_ins comp3 sans
c_2s_2i_mélange_sans
pr4
composition(s2) => M2
n2 + M2 => m2
m2 + mmel2 => mmel1
nmel2 + n2 => nmel1
```

Il peut enfin résoudre mél8 :

```
mél8
nb_sys nb_ins comp3 sans
c_2s_2i_mélange_sans
mél8
composition(s1) => M1
n1 + M1 => m1
mmel1 + m1 => mmel
mmel1 + n1 => nmel
nmel + mmel => Mmel
```

Décomposition locale

« On fait subir une compression adiabatique quasi-statique à 1 l d'azote à 25 °C, on obtient 0.75 l. Calculer la température finale du gaz. On suppose connus les coefficients calorimétriques de l'azote. »

```
problème(local3).
systèmes(local3,[système_C]).
instants(local3,[instant_G,instant_H]).
cp(système_C).
cv(système_C).
température(système_C,instant_G).
volume(système_C,instant_G).
volume(système_C,instant_H).
on_demande(local3,[température(système_C,instant_H)]).
gp(système_C).
trans_gp_fermé(système_C,instant_G,instant_H).
adiabatique(système_C,instant_G,instant_H).
```

SYRCLAD-thermodynamique classe local3 :

```
local3
nb_sys nb_ins adiabatique
```

Le classement s'arrête dans la classe un_système-deux_instants-conservation. SYRCLAD-thermodynamique compare le problème aux définitions des cas particuliers, et trouve que gamma est la donnée manquante par rapport au cas particulier : " la transformation est adiabatique, on connaît γ, parmi les grandeurs P, V, T, on en connaît deux à l'instant initial, l'une d'entre elles à l'instant final, et on demande l'autre". Il pose donc le sous-problème pr6 :

```
Sous-problème  pr6  : trouver gamma(système_C)
```

Il classe pr6 :

```
pr6
nb_sys nb_ins adiabatique
c_1s_2i_cons
```

et produit le plan suivant :

```
pr6
gamma = Cp/Cv
```

Il peut alors descendre dans la classe :

```
c_1s_2i_cons_gam
Le système système_C est adiabatique entre les instants instant_G et
instant_H. Comme on connaît gamma, on utilise la loi de Laplace et le fait
que PV/T=cte (système fermé), on en déduit qu'il existe une fonction
f(volume,température) constante entre instant_G et instant_H. Comme on
connaît volume(système_C,instant_G) et température(système_C,instant_G), on
connaît la valeur de cette fonction. Comme d'autre part on connaît
volume(système_C,instant_H), on en déduit température(système_C,instant_H).
```

Décomposition locale

« On fait subir une compression isotherme à 1 litre d'azote sous 2 atm, avec un rapport volumétrique de 1,9. Calculer la quantité de chaleur reçue par le gaz. »

```
problème(local7).
systèmes(local7,[système_C]).
instants(local7,[instant_F,instant_G]).
volume(système_C,instant_F).
rapport(volume,système_C,instant_F,instant_G).
pression(système_C,instant_F).
on_demande(local7,[chaleur(système_C,instant_F,instant_G)]).
trans_gp_fermé(système_C,instant_F,instant_G).
isotherme(système_C,instant_F,instant_G).
gp(système_C).
```

SYRCLAD-thermodynamique classe le problème local7 :

```
local7
nb_sys nb_ins isotherme cas_part_nrj1 cas_part_nrj_Q_it
c_1s_2i_cons_nrj_Q_it
Le système système_C est isotherme entre les instants instant_F et
instant_G, donc l'énergie interne est constante, et on a Q = -W. Calculons
donc le travail du système système_C entre les instants instant_F et
instant_G
```

Il pose le sous-problème p_travail0, dont la question est de calculer le travail. Il essaie de classer ce sous-problème.

```
p_travail0
nb_sys nb_ins isotherme cas_part_nrj1
```

Le classement s'arrête dans la classe 1système-2instants-conservation-question=nrj. SYRCLAD-thermodynamique compare le problème p_travail0 aux cas particuliers et trouve que le volume final est la donnée manquante par rapport au cas particulier "un invariant (T), deux variables initiales (V et P), une variable finale (P ou V), on demande le travail". Il pose donc le sous-problème pr7 :

```
Sous-problème pr7 : trouver volume(système_C, instant_G)
```

Il résout pr7 :

```
pr7
nb_sys nb_ins isotherme
c_1s_2i_cons
pr7
rapport volumique + volume initial => volume final
```

Il peut maintenant descendre dans la classe :

```
c_1s_2i_cons_nrj_W_inv
système_C est un système fermé (n=cte), et la variable température est
constante entre instant_F et instant_G, donc il existe une fonction
f(volume,pression) constante entre instant_F et instant_G. Comme on connaît
volume(système_C,instant_F) et pression(système_C, instant_F), on connaît
la valeur de cette fonction. Comme d'autre part on connaît
volume(système_C,instant_G), on peut intégrer dw en  volume pour obtenir le
travail du système système_C entre les instants instant_F et instant_G.
```

L'utilisation de la décomposition en thermodynamique a permis à SYRCLAD-thermodynamique de résoudre une vingtaine d'exercices supplémentaires variés.

3 Conclusion

Nous avons modifié le noyau de SYRCLAD en lui ajoutant une méthode générale de décomposition de problèmes. Cette architecture de méthode permet, si l'expert d'un domaine X fournit les connaissances déclaratives et procédurales appropriées, de résoudre des problèmes qui n'entrent pas dans la classification que l'expert a donnée à un SYRCLAD-X. Elle consiste à comparer le problème aux classes du graphe afin de déterminer une différence avec une classe la proche. On pose alors comme sous-problème de combler cette différence.

Cette méthode est appliquée aux problèmes additifs et à la thermodynamique. Dans les deux cas, on a fourni au système des connaissances déclaratives et procédurales spécifiques au domaine. SYRCLAD-additifs et SYRCLAD-thermodynamique résolvent grâce à cette méthode de nombreux exercices supplémentaires. L'intérêt de cette méthode est que l'on résout plus d'exercices, sans avoir à augmenter le graphe de classification.

Conclusion

1 Bilan du travail réalisé

Nous avons présenté le système SYRCLAD, qui est une architecture de résolveur de problèmes dans laquelle les phases de modélisation et de résolution sont séparées et où le processus de modélisation est guidé par une classification des problèmes. Nous avons ainsi proposé, d'une part, un cadre général dans lequel on peut expliciter de manière déclarative une classification de problèmes et insérer des connaissances de reformulation et de résolution, et d'autre part, un mécanisme d'exploitation de ces connaissances. Le système SYRCLAD a été testé sur quatre domaines d'application.

Si un expert du domaine X souhaite utiliser SYRCLAD pour réaliser un résolveur de problème du domaine X (SYRCLAD-X), cet expert doit définir un langage de la logique du premier ordre qui permette de décrire les problèmes du domaine X, un graphe de classification des problèmes du domaine X qui permette d'associer une méthode de résolution à certaines classes de problèmes, des règles de reformulation permettant de modéliser un problème concret afin d'obtenir un modèle plus proche de la théorie du domaine X, et des méthodes de résolution efficaces pour certaines classes de problèmes.

Un problème à résoudre est soumis à un système SYRCLAD-X sous la forme d'un modèle descriptif de ce problème, exprimé dans le langage défini par l'expert. Pour classer ce problème, SYRCLAD-X utilise le graphe de classification de problèmes que l'expert lui a donné et qui explicite les connaissances de classification. Pour utiliser le graphe de classification, SYRCLAD-X utilise les connaissances de reformulation fournies par l'expert pour déterminer la valeur des attributs discriminants intervenant dans la classification. Le système effectue une "opérationnalisation" du problème, obtenant ainsi d'une part une classe (si possible opérationnelle) et d'autre part un nouveau modèle du problème, appelé modèle opérationnel. Une expertise fournie par l'expert associe à chaque classe du graphe de classification une méthode de résolution adaptée. Pour obtenir une solution du problème, le système SYRCLAD-X applique la méthode de résolution associée à la classe du problème sur le modèle opérationnel.

En utilisant l'architecture de SYRCLAD, nous avons réalisé quatre résolveurs de problèmes SYRCLAD-X.

Le premier domaine d'application de SYRCLAD a été celui des exercices de dénombrement au niveau terminale scientifique. Par rapport au système expert réalisé pendant notre DEA [Guin et al 95], l'utilisation de SYRCLAD a permis une représentation plus déclarative des connaissances nécessaires à la résolution des exercices de dénombrement ; la résolution est plus claire et plus générale, et les résultats sont plus satisfaisants. Le système SYRCLAD-dénombrements résout 76 problèmes variés.

Par la suite, nous avons testé SYRCLAD sur le domaine des problèmes additifs proposés à l'école primaire, en nous basant sur une classification sur papier établie par des didacticiens. Le système SYRCLAD-additifs résout 50 problèmes additifs en utilisant une modélisation basée sur la notion d'opérateur, adaptée aux connaissances des élèves.

Nous avons ensuite étudié comment le système Bateleur [Nigro 95] choisit un plan de jeu au tarot, pour expliciter une classification cachée dans des règles. Le système SYRCLAD-tarot choisit un plan de jeu pour une trentaine de situations, et obtient les mêmes résultats que le système Bateleur, de manière plus explicite.

Enfin, nous avons étudié la résolution d'exercices de thermodynamique par le système Modélis [Tisseau 90] dans le but d'élaborer une classification. Le système SYRCLAD-thermodynamique résout une soixantaine d'exercices de thermodynamique en utilisant le graphe de classification que nous avons construit. Le résolveur de Modélis n'a pas de vision globale du problème et n'utilise pas de connaissances cachées concernant une classification de problèmes. Le fait d'utiliser une classification de problèmes permet à SYRCLAD-thermodynamique de reformuler le problème en interprétant les données du problème afin de le présenter de manière structurée, en faisant ressortir les points qui seront importants pour la résolution.

Dans SYRCLAD, les connaissances, et en particulier celles de classification, sont exprimées déclarativement. Cela permet d'expliciter une démarche de résolution à un niveau connaissance. De plus, le fait de séparer la résolution d'un problème en différents processus généraux (modélisation, classement, application d'une méthode) permet de mieux distinguer les démarches et les types de difficultés propres à chaque processus. L'abstraction possède également l'avantage de permettre une réutilisabilité plus aisée des connaissances contenues dans la classification, et en particulier de faire évoluer l'expertise plus facilement et avec moins de risques d'erreurs. Nous avons montré dans plusieurs domaines que l'on peut facilement ajouter une classe supplémentaire au graphe de classification.

L'architecture de SYRCLAD permet de concevoir des systèmes SYRCLAD-X destinés à servir de résolveurs de référence dans des EIAO sur chacun des domaines X. Un système destiné à enseigner une stratégie de résolution fondée sur une classification des problèmes doit à avoir accès à une représentation explicite de la classification et des processus mis en jeu. Nous souhaitons, dans chaque domaine et dans un cadre d'enseignement à un niveau donné, pouvoir expliquer à un élève la démarche de résolution d'un élève-expert, et les métaconnaissances qu'il utilise. Les explications concernant la résolution de ce problème seront d'autant plus faciles à donner que les connaissances seront explicitées. C'est pourquoi nous avons distingué dans SYRCLAD les différents types de connaissances intervenant dans une résolution : les connaissances de classification, les connaissances de reformulation et les connaissances de résolution ; ces différents types de connaissances sont exprimées de manière déclarative. SYRCLAD possède donc des (méta)connaissances explicites pour exprimer les choix stratégiques de résolution, ce qui sera important pour donner des explications.

Chacun des systèmes SYRCLAD-X est adapté à un niveau d'apprentissage donné, et son comportement peut être fourni comme exemple à un apprenant du domaine. Nous avons conçu SYRCLAD pour qu'il soit adapté à des domaines où les énoncés des problèmes correspondent à des

situations concrètes, alors que le cours est donné sous forme de connaissances théoriques. Il faut pour résoudre ces problèmes effectuer une reformulation du problème pour obtenir un modèle adapté aux connaissances du cours théorique. C'est cette phase de reformulation qui est difficile pour les élèves. SYRCLAD facilite l'expression des connaissances dans de tels domaines en permettant d'exprimer déclarativement une classification des problèmes et de distinguer connaissances de reformulation et connaissances de résolution. De plus, SYRCLAD sait utiliser ces connaissances pour résoudre un problème concret à partir d'un modèle descriptif de ce problème. La démarche utilisée par SYRCLAD pour modéliser ces problèmes et choisir une bonne méthode de résolution pourrait être enseignée aux élèves.

L'architecture de SYRCLAD semble être particulièrement adaptée pour construire un résolveur de problème utilisant une *méthode*[1] fondée sur la classification. Or, le résolveur d'un tuteur donneur de leçons de méthodes doit fonctionner selon les méthodes qu'il veut enseigner. Ce tuteur pourrait donc utiliser un système SYRCLAD-X comme résolveur de référence.

L'originalité de SYRCLAD par rapport à d'autres systèmes d'intelligence artificielle utilisant des classifications pour résoudre des problèmes vient du fait que dans les domaines d'application de SYRCLAD, les classes font intervenir un vocabulaire adapté aux méthodes de résolution, basées sur des principes abstraits, tandis que la représentation initiale du problème fait intervenir un vocabulaire décrivant une situation concrète. Il faut noter que dans la plupart des systèmes basés sur la classification, les valeurs des attributs du problème qui permettent la classification sont connues ; le problème posé est un problème "classifiable". Contrairement à ces systèmes, SYRCLAD traite des problèmes qui ne peuvent pas être classés directement à partir de leur formulation initiale, car les valeurs des attributs du problème à classer ne sont pas connues ; le système SYRCLAD ne sait même pas quels attributs seront pertinents pour le problème qu'il doit classer. Le système doit construire un autre modèle du problème afin de pouvoir le classer.

Nous avons défini des critères d'évaluation de la complexité d'un système SYRCLAD-X, c'est-à-dire de la complexité des connaissances fournies par l'expert du domaine X à SYRCLAD pour réaliser SYRCLAD-X. Ces critères sont relatifs au langage de description des problèmes et aux bases de connaissances de classification, de reformulation et de résolution. Ils permettent de repérer pour chaque domaine les étapes qui nécessitent le plus de connaissances pour résoudre des problèmes dans ce domaine. Nous avons ainsi constaté que la difficulté à résoudre des problèmes de dénombrement ne vient pas du graphe de classification, qui n'est pas grand, mais plutôt de la variété et de la richesse des modèles descriptifs, et surtout du grand nombre de règles de reformulation utilisables pour construire le modèle opérationnel d'un problème (ces règles étant, en outre, difficiles à utiliser). Dans les domaines des problèmes additifs et de la thermodynamique, les connaissances de reformulation sont moins nombreuses et moins difficiles à utiliser, mais le graphe de classification est plus grand, ce qui provoque des difficultés de mémorisation et d'utilisation du graphe.

---

[1] au sens de M. Rogalski.

Le système SYRCLAD permet d'utiliser la décomposition d'un problème en sous-problèmes afin de résoudre certains problèmes. D'une part, un expert d'un domaine X peut associer à une classe opérationnelle du graphe de classification du domaine X une méthode de résolution composée, c'est-à-dire une méthode qui consiste à construire des sous-problèmes, qui doivent être classés et résolus par le système SYRCLAD-X. La solution du problème initial dépend alors des solutions des sous-problèmes. D'autre part, pour un domaine donné, il existe des problèmes que le graphe de classification ne permet pas de résoudre. Nous avons modifié le noyau de SYRCLAD afin qu'il ait la possibilité de comparer ces problèmes nouveaux au graphe de classification, pour déterminer en quoi ils sont différents des problèmes que le système sait résoudre directement. Il construit ainsi des sous-problèmes qu'il classe et résout afin que le problème entre dans la classification. Pour qu'il puisse déterminer ces différences et construire ces sous-problèmes, il faut donner au système SYRCLAD des connaissances déclaratives et des connaissances procédurales spécifiques pour chaque domaine. Cette méthode de décomposition de problèmes a été appliquée aux problèmes additifs et à la thermodynamique. Elle permet aux systèmes de résoudre plus de problèmes, sans avoir à augmenter le graphe de classification.

2 Utilisations possibles de SYRCLAD

Le système SYRCLAD peut être appliqué à d'autres domaines. En effet, si un projet d'élaboration d'un EIAO a besoin d'un résolveur de problèmes, il est possible que l'architecture de SYRCLAD soit bien adaptée au domaine d'application. Pour savoir si SYRCLAD peut être appliqué à un domaine, la question est de savoir si une classification de problèmes du domaine est explicite ou non. Et s'il n'existe pas de classification de problèmes du domaine, il est très difficile à moins d'être expert de dire si on va pouvoir en dégager une. La méthode de calcul de primitives qu'ELISE [Delozanne 92] tente d'enseigner repose effectivement sur une classification des problèmes. C'est pourquoi nous avons indiqué que SYRCLAD pourrait servir à réaliser un système qui calcule des primitives, qui constituerait un résolveur de référence pour l'environnement ELISE.

La difficulté de dire si SYRCLAD peut bénéficier à un EIAO existant ou être appliqué à un domaine vient du fait que nous proposons dans les domaines que nous avons abordés une "nouvelle manière de résoudre". En effet, les enseignements classiques dans ces domaines ne sont pas basés sur ce type de modélisation par classification. Ceci nous amène à proposer une utilisation de ce travail qui pourrait même avoir lieu en dehors d'un cadre EIAO dans les IUFM : proposer les méthodes utilisées par SYRCLAD aux formateurs et aux enseignants. En effet, la plupart d'entre eux n'utilisent pas ce type de raisonnement pour résoudre. Il a été mis en évidence pour les problèmes additifs que les enseignants résolvent une équation dans $\mathbb{Z}$ [Rebillard 95], alors que cela est impossible aux élèves. L'enseignement des dénombrements et de la thermodynamique a souvent du mal à passer peut-être parce que les enseignants n'explicitent pas assez les connaissances de modélisation qu'ils utilisent ou les méthodes de résolution adaptées à certaines classes de problèmes. Par contre, ils ont, pour tous les domaines considérés, les connaissance nécessaires pour une appropriation rapide de ces résolutions basées sur la modélisation et la classification. C'est d'autant plus intéressant d'expérimenter ces méthodes de raisonnement en formation initiale, pour une modification éventuelle des pratiques professionnelles des enseignants, même indépendamment de la conception d'EIAO. En effet, il semble que les tuteurs de démonstration géométrique ont eu un impact sur les pratiques

enseignantes de la démonstration géométrique en environnement papier / crayon (mise en évidence de méthodes, expliciter des faits autrefois implicites dans le discours enseignant et révélés par le fonctionnement des tuteurs...). Proposer les méthodes de raisonnement utilisées par SYRCLAD ne serait pas une application directe du système, mais une utilisation du travail de modélisation du raisonnement entrepris pendant cette thèse, et en particulier des classifications que nous avons explicitées.

Pour appliquer SYRCLAD à un domaine, l'expert du domaine doit fournir au système un langage de description des problèmes, un graphe de classification, des règles de reformulation et des méthodes de résolution. En conséquence, il ne serait pas raisonnable de proposer l'architecture SYRCLAD telle quelle à un enseignant ou un expert pour qu'il l'applique à un nouveau domaine. En effet, la construction d'une classification est déjà un travail difficile en dehors de tout système. Si cette classification est connue par l'expert, on peut lui demander de l'expliciter dans un langage qui lui convient, puis la mettre sous la forme attendue par SYRCLAD. Il faut également produire des règles de reformulation, ce dont un informaticien a l'habitude, mais pas n'importe quel expert. La définition du langage de description à base de prédicats est une tâche difficile pour un expert. Pour appliquer l'architecture SYRCLAD à un autre domaine, nous envisageons plutôt une coopération avec un expert afin de préciser le cadre du système avec lui, et éventuellement l'intervention d'un autre informaticien (par exemple un stagiaire de DEA) pour la phase d'implémentation. Deux ou trois coopérations de ce type pourraient nous amener à définir avec l'aide de ces personnes une spécification précise des connaissances à donner au système en formalisant les concepts utilisés, de manière à rendre notre intervention inutile, mais pas celle de l'informaticien-implémenteur.

Par contre, nous pensons qu'il est raisonnable de demander à un enseignant de modéliser les exercices dans le langage logique proposé. En effet, il est possible d'aider un enseignant qui modélise un exercice en lui proposant une "aide à la modélisation". En dénombrement, le groupe Combien?[1] a créé une interface permettant à un élève de décrire le problème (qu'on lui a posé en langage naturel) grâce à un menu proposant des objets (cartes, des, jetons), et lui demandant combien d'objets on tire, comment, ce qu'on forme, etc. Cette interface peut être modifiée de manière à afficher les prédicats correspondant aux choix de l'enseignant, celui-ci voyant le modèle descriptif se construire. Un choix dans le menu peut produire plusieurs prédicats, permettant par exemple à l'enseignant de comprendre la séquence définissant "un sac de 5 billes". Cette interface lui permettrait de se familiariser avec le langage, et devrait devenir inutile pour l'enseignant par la suite. Par contre, si l'enseignant veut utiliser de nouveaux objets qui ne sont pas prévus dans le langage, par exemple des courses de chevaux pour les dénombrements, il ne peut pas les ajouter au langage. En effet, cette introduction de nouveaux objets demande souvent d'ajouter des règles de reformulation correspondantes. Ceci demande donc l'intervention du concepteur (informaticien-expert) du résolveur SYRCLAD-X.

[1] H. Giroire et G. Tisseau de Paris 6, F. Le Calvez et M. Urtasun de Paris 5 et J. Duma du Lycée Jacquard.

3 Capitalisation d'expérience

Le but initial de cette thèse était de réaliser un résolveur de problèmes basé sur la classification et transversal à plusieurs domaines. La possibilité de réaliser un tel résolveur était une hypothèse a priori et il n'était pas évident qu'elle soit validée et qu'un noyau commun pourrait être dégagé. La réalisation de ce système parait être un résultat intéressant dans cette optique. Elle montre qu'il est intéressant d'essayer de dégager un processus de raisonnement indépendant du domaine, et d'appliquer ce processus à plusieurs domaines. C'est la démarche qu'a choisie G. Pecego pour réaliser SYGEP [Pecego 97], un système qui crée des exercices dans des domaines scientifiques variés : électricité, thermodynamique, chimie, mécanique. Le système SYGEP utilise une base de connaissances générale relative à la construction d'énoncés, et pour chaque domaine une base de connaissances spécifique.

Dans les trois domaines des dénombrements, des problèmes additifs et du tarot, nous nous sommes basés pour expliciter un graphe de classification soit sur une classification sur papier soit sur une classification apparaissant dans des règles, mais ces classifications explicitaient uniquement les classes opérationnelles de problèmes, avec les méthodes de résolution adaptées. Il nous a donc fallu construire les classes intermédiaires de la hiérarchie. Cette construction doit être validée par l'expert du domaine et permet de dégager des concepts intéressants. Cela peut être un travail intéressant pour quelqu'un qui utilise uniquement des classes opérationnelles, même s'il n'envisage pas d'utiliser SYRCLAD.

Pour les dénombrements et le tarot, nous avons dû modifier un système à base de règles pour distinguer les connaissances de classification, de reformulation et de résolution contenues dans ces règles. Ce travail demande de "déclarativiser" les connaissances exprimées par les règles. Cela n'a pas posé de problèmes significatifs pour les dénombrements et le tarot car les deux systèmes à base de règles que nous avons modifiés avaient été conçus avec une idée de classification des problèmes. Si quelqu'un a conçu un système à base de règles avec une idée de classification de problèmes, essayer de distinguer les connaissances de classification, de reformulation et de résolution contenues dans ses règles lui permettra de gagner en déclarativité, ce travail ne devant pas a priori être trop difficile. De plus, si quelqu'un envisage de concevoir un résolveur de problèmes dans un domaine où il faut modéliser les problèmes pour les résoudre, distinguer les connaissances de classification, de reformulation et de résolution devrait lui faciliter la tâche, même si ces connaissances restent exprimées par des règles.

L'application de SYRCLAD à la thermodynamique a été basée sur l'étude de Modélis [Tisseau 90]. Nous avons tenté de construire une classification de problèmes de thermodynamique en suivant l'avis de l'expert (G. Tisseau) selon lequel "on devrait pouvoir dégager une classification". Ce travail de construction d'un graphe de classification a été long et vraiment difficile, d'une part parce que c'est un travail difficile quel que soit le domaine et d'autre part parce que le domaine de la thermodynamique est complexe. Nous avons décrit dans le chapitre 7 la démarche que nous avons utilisée pour construire cette classification. La méthode que nous avons employée peut s'avérer efficace pour construire un graphe de classification dans un autre domaine. L'idée générale est

d'étudier de nombreuses résolutions d'exercices. Un conseil possible pourrait être : pour classer des problèmes, commencer par classer des résolutions. Dans ces résolutions, repérer les connaissances qui sont utilisées ensemble, et si possible plus précisément les séquences d'applications de connaissances. Dans le premier cas, on obtiendra des "modules de connaissances" et dans l'autre des "plans". Classer alors les problèmes en fonction des modules et des plans qu'on utilise pour les résoudre. Cela devrait permettre de repérer les classes opérationnelles. La difficulté consiste ensuite à décrire ces classes non plus en fonction des résolutions, mais en fonction des énoncés. On peut au moins essayer de repérer si certains traits de surface sont caractéristiques de certaines classes. En général, cela ne suffit pas. Il faut alors introduire des concepts plus abstraits liés aux conditions d'application des modules ou des plans, ce qui est difficile. Il faut également trouver comment reformuler un problème en fonction de ces concepts.

Par ailleurs, pour concevoir une classification de problèmes, un enseignant aurait sans doute intérêt à analyser la façon dont il invente des problèmes, car il procède alors souvent en partant d'un type de résolution qu'il veut faire appliquer (ou d'une classe de problèmes), puis il invente un habillage concret. Savoir comment on "habille" peut aider à savoir comment on "déshabille".

Une des principales difficultés apparues pendant la réalisation de ce travail a été de trouver des domaines d'application. En effet, il est difficile de caractériser le type de domaines d'application de SYRCLAD, c'est-à-dire le type de domaine dont les problèmes peuvent être classifiés pour la résolution. Une catégorisation connue : prédiction, diagnostic, contrôle, etc. ne semble pas adaptée ou suffisante. Une catégorisation de domaines pourrait également être faite en fonction des démarches à utiliser pour construire une classification de problèmes. En effet, la classification des problèmes de thermodynamique a été construite en distinguant d'abord des classes de résolution, puis en définissant ensuite des attributs discriminants correspondant à ces classes de résolution. Par contre, la classification des problèmes additifs s'appuie essentiellement sur des attributs du problème, les méthodes de résolution découlant des classes de problèmes définies par ces attributs.

4 Perspectives de recherche

Une première perspective de recherche consiste à achever l'étude que nous avons entreprise sur la complexité des systèmes SYRCLAD-X. Tout d'abord, les critères que nous avons définis doivent être améliorés. Ensuite, il faudra définir le calcul de complexité correspondant à ces critères. Enfin, ces critères de complexité pourraient être adaptés à un niveau plus local : à chaque classe du graphe de classification, et même à chaque résolution d'un problème par le système. La définition de la complexité d'une classe de problèmes ou d'un problème peut être utile à un tuteur pour choisir le problème qu'il propose à l'élève, en fonction de la difficulté souhaitée. Mais avant qu'un tuteur utilise ces critères de complexité relatifs à la résolution de problèmes par un système SYRCLAD-X, il faudra les confronter à la complexité effective des problèmes pour les élèves. Ceci devra faire l'objet d'une étude didactique approfondie.

Une deuxième perspective de recherche consiste à essayer d'automatiser certaines étapes de la réalisation d'un résolveur SYRCLAD-X. En effet, c'est un expert humain qui doit donner au système un graphe de classification et des connaissances de reformulation et de résolution. Certaines de ces

connaissances pourraient être produites par un système. Par exemple, on aurait pu automatiser la construction du graphe de classification de SYRCLAD-tarot à partir de la base de règles de Bateleur. On peut également envisager de définir uniquement les classes opérationnelles d'une classification de problèmes, laissant le système construire des classes intermédiaires. Plusieurs graphes pouvant ainsi être construits, l'expert pourrait choisir celui qui lui semble le plus pertinent. L'aboutissement d'une telle approche serait la réalisation d'un système auquel on donne pour un domaine X des connaissances opérationnelles pour résoudre des problèmes du domaine, des connaissances conceptuelles concernant ce domaine, des exemples d'exercices résolus à l'aide des connaissances opérationnelles, et qui engendre automatiquement un résolveur SYRCLAD-X, c'est-à-dire le graphe de classification des problèmes, les règles de reformulation, et les méthodes de résolution. Pour mener ce travail, il serait utile d'étudier une complexité à modéliser un domaine, en fonction de la variété des problèmes proposés, de la quantité de connaissances opérationnelles disponibles, et de la présence éventuelle de connaissances liées à un début de classification.

Une troisième perspective de recherche est l'utilisation par des EIAO des résolveurs SYRCLAD-X que nous avons élaborés. Pour cela, il est nécessaire de travailler avec une équipe pluridisciplinaire. Le groupe de travail Combien? a pour but la réalisation d'un EIAO dans le domaine des dénombrements et utilise le système SYRCLAD-dénombrements pour résoudre des problèmes. Pour utiliser un système SYRCLAD-X dans le but de produire des explications, il faudra dans un premier temps associer des explications aux connaissances déclaratives données à SYRCLAD-X. Les explications associées au connaissances de classification seront destinées à justifier les choix stratégiques ; celles associées aux connaissances de reformulation seront destinées à justifier les choix de modélisation.

Nous avons déjà associé à chacune des règles de reformulation du système SYRCLAD-dénombrements une phrase d'explication qui est instanciée au problème résolu. Par exemple :

> « Le résultat r de l'action former f est un mot ; comme il n'y a aucune précision dans l'énoncé sur les répétitions, on suppose par défaut qu'on forme avec remise.»
>
> « L'ensemble où l'on choisit j est un jeu de 32 cartes, il y a donc 8 catégories de taille 4 pour l'attribut hauteur, et 4 catégories de taille 8 pour l'attribut couleur. »

Dans les autres domaines, on pourrait associer de la même manière des explications à chaque règle de reformulation. Il faudrait également définir le contenu des explications relatives à la classification et la manière de les associer au graphe : ces deux phases sont les premières étapes nécessaires à la conception d'environnements d'apprentissage utilisant des systèmes SYRCLAD-X comme résolveurs de référence.

Le projet d'environnement interactif que nous avons serait d'utiliser SYRCLAD pour accompagner la démarche de résolution d'un élève en effectuant du diagnostic et en fournissant des explications. Un problème serait proposé à l'élève. Pour le guider dans la résolution de ce problème, le système proposerait à l'élève de déterminer la valeur de certains attributs du problème (les attributs discriminants intervenant dans le graphe de classification du domaine). Le système comparerait les réponses de l'élève à celles de SYRCLAD et en cas de désaccord expliquerait la valeur trouvée par SYRCLAD grâce à la règle de reformulation utilisée, comme illustré ci-dessus. Dans un premier

temps, pour déterminer la valeur d'un attribut, l'élève pourrait avoir à choisir une règle dans l'ensemble des règles de reformulation associées à cet attribut. Quand tous les attributs nécessaires auraient été déterminés, le système proposerait alors une "machine" spécifique à la classe du problème, pour que l'élève complète la description de ce qui est pour SYRCLAD le modèle opérationnel, et le système proposerait alors à l'élève de suivre l'application par SYRCLAD de la méthode de résolution adaptée à la classe. Dans un deuxième temps, l'élève choisirait lui-même la méthode à appliquer parmi un ensemble de méthodes de résolution.

Cet environnement serait proposé à l'élève après que celui-ci ait déjà utilisé un environnement tel que celui mis au point par le projet Combien? : on propose à l'élève un exercice et la "machine" adaptée à la classe de l'exercice [Le Calvez et al 97]. L'environnement que nous proposons permettrait dans un premier temps à l'élève de choisir la bonne machine (ou classe), puis dans un deuxième temps de choisir la méthode de résolution adaptée à cette classe. En utilisant une démarche analogue à celle que nous avons utilisée dans ce travail de thèse, il serait intéressant de tenter de dégager des invariants transversaux aux différents domaines, pour concevoir une architecture d'"environnement interactif d'aide à la résolution par modélisation", applicable comme SYRCLAD à plusieurs domaines, en insérant dans cette architecture des connaissances relatives à chacun des domaines.

ANNEXES

Annexe A
Les règles de reformulation déclenchées pendant la résolution de m10 (chapitre 1)

Nous avons décrit dans le chapitre 1 le classement du problème de dénombrement m10. SyrCLAD-dénombrements délenche pendant ce classement des règles de reformulation, nous les détaillons ici.

Nous rappelons le modèle descriptif de m10 :

```
problème(m10).
action(m10,f).
à_dénombrer(m10,d).
est_un(f,action_former).
type_des_objets_formés(f,mot).
au_moyen_de(f,a).
tout_résultat(f,r).
est_un(r,mot).
taille(r,5).
répétitions_possibles(r).
effectif_attribut(r,exactement,2,type_lettre,voyelle).
effectif_attribut(r,exactement,2,lettre,b).
est_un(a,ensemble).
taille(a,26).
tout_élément(a,xa).
est_un(xa,lettre).
est_un(d,ensemble).
ens_des_résultats(d,f).
```

Voici les règles que SYRCLAD-dénombrements déclenche pendant le classement :

```
former,
obs_ens : tout_élément, obs_ens : élément_générique,
résultat_de_action,
contraintes : rapporter5, contraintes : rapporter5, contraintes : liste_vide,
contraintes : établir5, contraintes : établir5,
type_action, taille_objet, même_type,
attributs : liste_att, attributs : cons_att, attributs : cons_att,
former_au_moyen_de, alphabet, taille_alphabet, att_alphabet, cat_alphabet,
effectifs_disj, mot, former_défaut
```

Pour chacune de ces règles, nous la donnons en Prolog telle qu'elle est donnée à SYRCLAD-dénombrements, puis nous la traduisons en français en l'instanciant au problème m10.

```
règle(former):-
      si(   [problème(P),
             action(P,F),
             est_un(F,action_former),
             tout_résultat(F,R)]),
      alors([ajouter(objet_formé(P,R))]).
```

```
Si le problème est m10
Si l'action du problème est f
Si f est une action_former
Si tout résultat de f est r
Alors r est l'objet formé de m10
```

```
règle(tout_élément,obs_ens):-
      si(   [problème(P),
            à_dénombrer(P,D),
            est_un(D,ensemble),
            eval_prolog(not(fait(tout_élément(D,_))))]), /* absence */
      alors([eval_prolog(gensym(y,Y)),ajouter(tout_élément(D,Y))]).

Si le problème est m10
Si l'ensemble à dénombrer dans m10 est d
Si d est un ensemble
Si tout_élément de d n'est pas défini
Alors créer y0 comme tout_élément de d
```

```
/* on nomme l'élément générique Y de l'ensemble D à dénombrer dans le problème
P, c'est juste pour y avoir plus facilement accès, il est souvent utilisé */
règle(élément_générique,obs_ens):-
      si(   [problème(P),
            à_dénombrer(P,D),
            est_un(D,ensemble),
            tout_élément(D,Y)]),
      alors([ajouter(élément_générique(P,Y))]).

Si le problème est m10
Si l'ensemble à dénombrer dans m10 est d
Si d est un ensemble
Si tout_élément de d est y0
Alors y0 est l'élément générique de m10
```

```
/* on observe que l'ensemble à dénombrer D est déterminé à partir des résultats
de l'action A du problème P */
règle(résultat_de_action):-
      si(   [problème(P),
            à_dénombrer(P,D),
            est_un(D,ensemble),
            action(P,F),
            eval_prolog(member(M,[ens_des_résultats,inclus_dans_résultats])),
            M(D,F)]),
      alors([ajouter(résultat_de_action(P))]).

Si le problème est m10
Si l'ensemble à dénombrer dans m10 est d
Si d est un ensemble
Si l'action de m10 est f
Si d est soit l'ensemble des résultats de f, soit inclus dans les résultats de f
Alors dans m10, l'ensemble à dénombrer est le résultat de l'action
```

```
règle(rapporter5,contraintes):-
      si(   [problème(P),
            objet_formé(P,R),
            eval_prolog(est_une_contrainte(M,5)),
            M(R,E,N,T,L),
            résultat_de_action(P),
            élément_générique(P,Y)]),
      alors([ajouter(M(Y,E,N,T,L))]).

/* 2 déclenchements pour m10 */
```

```
Si le problème est m10
Si l'objet formé dans m10 est r
Si effectif_attribut(r,exactement,2,type_lettre,voyelle)
Si dans m10 l'ensemble à dénombrer est le résultat de l'action
Si y0 est l'élément générique de m10
Alors effectif_attribut(y0,exactement,2,type_lettre,voyelle)

Si le problème est m10
Si l'objet formé dans m10 est r
Si effectif_attribut(r,exactement,2,lettre,b)
Si dans m10 l'ensemble à dénombrer est le résultat de l'action
Si y0 est l'élément générique de m10
Alors effectif_attribut(y0,exactement,2,lettre,b)
```

```
/* on initialise à la liste vide */
règle(liste_vide,contraintes):-
      si(   [problème(P)]),
      alors([eval_prolog(retirer_tous(contraintes(P,_))),
            ajouter(contraintes(P,[]))]).

Si le problème est m10
Alors détruire toutes les listes de contraintes de la base de faits
et créer une liste de contrainte vide pour m10
```

```
/* les prédicats de contrainte d'arité 5 */
règle(établir5,contraintes):-
      si(   [problème(P),
            élément_générique(P,Y),
            eval_prolog(est_une_contrainte(M,5)),
            M(Y,E,N,T,L)]),
      alors([modifier(contraintes(P,Liste),
            contraintes(P,[M(E,N,T,L)|Liste]))]).

/* 2 déclenchements pour m10 */

Si le problème est m10
Si l'élément générique de m10 est y0
Si effectif_attribut(y0,exactement,2,type_lettre,voyelle)
Alors ajouter effectif_attribut(exactement,2,type_lettre,voyelle) à la liste des
contraintes

Si le problème est m10
Si l'élément générique de m10 est y0
Si effectif_attribut(y0,exactement,2,lettre,b)
Alors ajouter effectif_attribut(exactement,2,lettre,b) à la liste des
contraintes
```

```
règle(type_action):-
      si(   [problème(P),
            action(P,A),
            est_un(A,Type)]),
      alors([ajouter(type_action(P,Type))]).

Si le problème est m10
Si l'action de m10 est f
Si m10 est une action_former
Alors le type de l'action de P est action_former
```

```
règle(taille_objet):-
      si(   [problème(P),
            objet_formé(P,Y),
            taille(Y,N)]),
      alors([ajouter(taille_objet_formé(P,N))]).

Si le problème est m10
Si l'objet formé de m10 est r
Si la taille de r est 5
Alors la taille de l'objet formé de m10 est 5
```

```
/* les contraintes sont toutes de même type (même prédicat) */
règle(même_type):-
      si(   [problème(P),
            contraintes(P,L),
            eval_prolog(même_prédicat(L))]),
      alors([ajouter(contraintes_même_type(P))]).

Si le problème est m10
Si la liste de contraintes de m10 est [effectif_attribut(exactement,2,lettre,b),
effectif_attribut(exactement,2,type_lettre,voyelle)]
Si toutes les contraintes de cette liste sont définies par le même prédicat
Alors toutes les contraintes de m10 sont de même type
```

```
/* on initialise à la liste vide */
règle(liste_att,attributs):-
      si(   [problème(P),
            contraintes(P,L)]),
      alors([eval_prolog(retirer_tous(attributs(P,_))),
            ajouter(attributs(P,[]))]).

Si le problème est m10
Si la liste des contraintes de m10 est :
      [effectif_attribut(exactement,2,lettre,b),
      effectif_attribut(exactement,2,type_lettre,voyelle)]
Alors détruire toutes les listes d'attributs présentes dans la base de faits et
créer une liste d'attribut vide pour m10
```

```
/* repérer tous attributs pouvant figurer dans les prédicats de la liste des
contrainte, et en construire la liste */
règle(cons_att,attributs):-
      si(   [problème(P),
            contraintes(P,L),
            eval_prolog(member(X,L)),
            eval_prolog(contrainte_avec_attribut(X,Y)),
            eval_prolog(fait(attributs(P,K))),
            eval_prolog(not(member(Y,K)))]),
      alors([modifier(attributs(P,K),
            attributs(P,[Y|K]))]).

/* 2 déclenchements pour m10 */

Si le problème est m10
Si la liste des contraintes de m10 est :
      [effectif_attribut(exactement,2,lettre,b),
      effectif_attribut(exactement,2,type_lettre,voyelle)]
Si effectif_attribut(exactement,2,lettre,b) appartient à cette liste
Si lettre est l'attribut présent dans cette contrainte
Si lettre n'est pas encore présent dans la liste des attributs
Alors ajouter lettre à la liste des attributs
```

```
Si le problème est m10
Si la liste des contraintes de m10 est :
      [effectif_attribut(exactement,2,lettre,b),
      effectif_attribut(exactement,2,type_lettre,voyelle)]
Si effectif_attribut(exactement,2,type_lettre,voyelle) appartient à cette liste
Si type_lettre est l'attribut présent dans cette contrainte
Si type_lettre n'est pas encore présent dans la liste des attributs
Alors ajouter type_lettre à la liste des attributs
```

```
/* quand on forme au moyen d'un ensemble, c'est cet ensemble */
règle(former_au_moyen_de):-
      si(   [problème(P),
            action(P,A),
            est_un(A,action_former),
            au_moyen_de(A,E),
            résultat_de_action(P)]),
      alors([ajouter(ens_où_on_choisit(P,E))]).

Si le problème est m10
Si f est l'action de m10
Si f est une action_former
Si f a lieu au_moyen_de a
Si dans m10 l'ensemble à dénombrer est le résultat de l'action
Alors l'ensemble où l'on choisit de m10 est a
```

```
/* reconnaître l'alphabet */
règle(alphabet):-
      si(   [problème(P),
            ens_où_on_choisit(P,E),
            est_un(E,ensemble),
            taille(E,26),
            tout_élément(E,X),
            est_un(X,lettre)]),
      alors([ajouter(est_un(E,alphabet)),
            ajouter(ensemble_prototype(P,alphabet))]).

Si le problème est m10
Si l'ensemble où l'on choisit de m10 est a
Si a est un ensemble
Si la taille de a est 26
Si tout élément de a est xa
Si xa est une lettre
Alors a est l'alphabet et l'ensemble où l'on choisit de m10 est l'ensemble
prototype alphabet
```

```
règle(taille_alphabet):-
      si(   [problème(P),
            ensemble_prototype(P,alphabet)]),
      alors([ajouter(taille_ens_où_on_choisit(P,26))]).

Si le problème est m10
Si l'ensemble où l'on choisit de m10 est l'ensemble prototype alphabet
Alors la taille de l'ensemble où l'on choisit de m10 est 26
```

```
règle(att_alphabet):-
      si(   [problème(P),
            ensemble_prototype(P,alphabet)]),
      alors([ajouter(attributs_possibles(P,[lettre,type_lettre]))]).

Si le problème est m10
Si l'ensemble où l'on choisit de m10 est l'ensemble prototype alphabet
Alors les attributs possibles dans les contraintes de m10 sont lettre et
type_lettre
```

```
/* catégories de l'alphabet */
règle(cat_alphabet):-
      si(   [problème(P),
            ensemble_prototype(P,alphabet)]),
      alors([ajouter(taille_catégorie(P,type_lettre,voyelle,6)),
            ajouter(taille_catégorie(P,type_lettre,consonne,20)),
            ajouter(taille_catégorie(P,lettre,_,1)),
            ajouter(nombre_catégories(P,type_lettre,2)),
            ajouter(nombre_catégories(P,lettre,26))]).

Si le problème est m10
Si l'ensemble où l'on choisit de m10 est l'ensemble prototype alphabet
Alors il y a pour m10 :
      2 catégories pour l'attribut type_lettre de taille 6 pour les voyelles et
20 pour les consonnes
      26 catégories de taille 1 pour l'attribut lettre
```

```
/* on a plusieurs contraintes de même type effectif_attribut(exactement,....)
portant sur des attributs différents. C'est une répartition dont on construit la
liste en vérifiant que les catégories sont disjointes */
règle(effectifs_disj):-
      si(   [problème(P),
            eval_prolog(not(fait(une_seule_contrainte(P)))), /* absence */
            contraintes(P,[effectif_attribut(exactement,_,_,_)|_]),
            contraintes_même_type(P),
            eval_prolog(not(fait(un_seul_attribut(P)))),
            eval_prolog(construire_liste_nb_att_val(P,L)), /* "exactement" */
            eval_prolog(cat_disjointes(P,L))]) ,/* pour ttes les contraintes */
      alors([ajouter(type_problème(P,répartition)),
            eval_prolog(construire_liste2(P,L,Lr)),
            ajouter(liste(P,Lr))]).

Si le problème est m10
Si il y a plusieurs contraintes dans m10
Si elles sont toutes de type effectif_attribut(exactement,...)
Si il y a plusieurs attributs intervenant dans ces contraintes
Si les catégories définies par ces contraintes sont disjointes (b n'est pas une
voyelle)
Alors le type de m10 est répartition et l'ensemble caractéristique de m10 est
[[1,2,b],[6,2,voyelle]]
```

```
règle(mot):-
      si(   [problème(P),
            objet_formé(P,R),
            est_un(R,mot)]),
      alors([ajouter(type_objet_formé(P,liste))]).

Si le problème est m10
Si l'objet formé de m10 est r
Si r est un mot
Alors le type de l'objet formé de m10 est liste
```

```
/* on forme par défaut avec remise*/
règle(former_défaut):-
      si(   [problème(P),
            objet_formé(P,R),
            est_un(R,M),
            eval_prolog(member(M,[liste_chiffres,mot])),
            résultat_de_action(P)]),
      alors([ajouter(remise(P,avec))]).

Si le problème est m10
Si l'objet formé de m10 est r
Si r est un liste_cbiffres ou un mot
Si dans m10 l'ensemble à dénombrer est le résultat de l'action
Alors on tire avec remise
```

Annexe B
Quelques prédicats du noyau de SYRCLAD

Nous présentons ici des prédicats indépendants du domaine et qui sont utilisés par le noyau de SYRCLAD. Nous présentons d'abord des prédicats exprimant des faits puis des prédicats exprimant des actions.

Faits

problème(P) P est le problème à résoudre

racine_arbre(C) C est la classe racine du graphe de classification

classe_courante(P,C) pendant le classement de P, C est la classe courante (la classe la plus spécifique auquel le problème appartient)

test_fils(Classe,Pb,Att) Att est l'attribut qui discrimine les sous-classes de Classe

valeur_fils(Classe,Pb,Classe_fils) le problème Pb appartient à la sous-classe Classe_fils de la classe courante Classe

statut(Classe,opérationnelle/non_opérationnelle).

attribut(Classe,Pb,Att,_,Règles,Règles_défaut) l'attribut Att est défini dans la classe Classe et les règles Règles permettent de déterminer sa valeur pour le problème Pb

règle(Nom_règle):-si([Prémisses]),alors([Actions]) une règle

règle(Nom_règle,Paquet):-si([Prémisses]),alors([Actions]) une règle d'un paquet

fait(F) F est un fait de la base de fait

Actions

lire_bf(F), ajouter_bf(F), retirer_bf(F) lire, ajouter ou retirer F de la base de fait

employer_résolution(Classe) utiliser la méthode de résolution associée à la classe Classe

descendre_arbre(P) faire descendre le problème vers une sous-classe de la classe courante

eval_prem(F) chercher F dans la base de faits ou essayer de déterminer sa valeur grâce à des règles de reformulation

classer(Problème) classer (et résoudre) le problème Problème

but(P,Att) déterminer la valeur de l'attribut Att en utilisant les règles de reformulation

utiliser(Liste_règles) essayer de déclencher toutes les règles de Liste_règles jusqu'à ce qu'une d'entre elles se déclenche

saturer(Paquet) saturer le paquet de règles Paquet en essayant de déclencher toutes les règles avec toutes les instanciations possibles

décomposer(Problème) décomposer un problème dont le classement a échoué

chercher_éléments_comparaison(Problème,Liste_éléments) déterminer la valeur pour ce problème des éléments de comparaison définis pour le domaine

chercher_classe_proche(Liste_éléments,Type_différence) chercher dans le graphe de classification une classe proche par rapport aux éléments de comparaison du problème, et établir le type de la différence

construire_sous_problème(Type_différence,Sous_problème) construire un sous-problème qui permette de réduire cette différence */

reconstruire(Sous_problème,Problème) la solution du sous-problème modifie l'énoncé du problème initial

Annexe C
Connaissances données à SYRCLAD-dénombrements

C1/ Prédicats du langage de description et prédicats issus de la classification

Prédicats utilisés dans les modèles descriptifs

est_un(X,*objet*) action_former, action_tirer, action_jeter_dés, action_tourner_roues, action_attribuer_rôles, ensemble, liste_chiffres, plaquette, chiffre, dé_cubique, collection, attribution, sac, jeton, roue_de_loterie, mot, lettre, concaténation, jeu_de_32_cartes, récipient, liste, urne, carton

action(P,A) : A est l'action de P *relation*

à_dénombrer(P,D) : dans le problème P c'est l'ensemble D qu'il faut dénombrer *relation + inconnue*

au_moyen_de(A,E) : l'action(former) A a lieu au moyen de l'ensemble E *relation*

avec_remise(A) : l'action A a lieu avec remise *propriété*

contenu(R,C) : le contenu du récipient R est C *relation*

effectif_chaque_élément(X,au_plus/exactement,N) : dans X chaque élément apparait au plus/exactement N fois *propriété*

éléments(E,L) : les éléments de E sont ceux de la liste L éléments(E,[1,2,C]) *relation + objet*

éléments_numérotés_séquentiellement(E,N,M) : les éléments de E sont numérotés séquentiellement de N jusqu'à M *propriété*

en_effectuant(A,T) : l'action(former) A a lieu en effectuant l'action(tirer)T *relation*

ensemble(A,E) : E est l'ensemble de départ de l'action_attribuer_rôles A *relation*

ensemble_des_lettres_de(X,M) : X est l'ensemble des lettre du mot M *relation + objet*

ens_des_résultats(X,A) : X est l'ensemble des résultats de l'action A *relation*

inclus_dans(X,Y) : X est inclus dans Y *relation*

inclus_dans_résultats(E,A) : l'ensemble E est inclus dans l'ensemble des résultats de l'action A *relation*

nb_éléments_tirés(A,N) : on tire N éléments dans l'action A
propriété

on_dispose_de(P,J) : on dispose pour le problème P de l'objet J
relation

problème(P) : P est le problème *objet*

récipient(A,R) : R est le récipient de A *relation*

répétitions_possibles(X) : dans x les répétitions sont possibles
propriété

réunion_de(X,L) : X est la réunion des ensembles de la liste L
relation

rôles(A,R) : R est l'ensemble des rôles de l'action_attribuer_rôles A
relation

sans_remise(A) : l'action A a lieu sans remise *propriété*

simultanément(A) : L'action_tirer A a lieu simultanément
propriété

successivement(A) : L'action_tirer A a lieu successivement
propriété

taille(X,N) : X est de taille N *propriété*

tout_résultat(A,X) : tout résultat X de l'action A *relation*

tout_élément(E,X) : tout_élément X de l'ensemble E *relation*

type_des_objets_formés(A,T) : le type des objets formés par l'action(former) F est T
propriété

porte_inscription_dans(X,Ens) *relation*

sans_répétitions(X) *propriété*

ensemble_des_dés(action,ensemble) *relation*

effectif_valeur(X,exactement,N,R) *propriété*

désigné_par(X,des) *propriété*

couleur(X,coul) *propriété*

ensemble_des_roues(action,ens) *relation*

nombre_de_secteurs(roue,N) *propriété*

secteurs_numérotés_séquentiellement(X,N,P) *propriété*

concaténation(X,L) *relation*

porte_inscription_dans(X,ens) *relation*

<u>dont prédicats de contraintes:</u>

classes_effectif_attribut(X,N,K,Att) : dans X on veut N classes de K éléments pour l'attribut Att
propriété

effectif_attribut(X,exactement/au_plus/au_moins,N,Att,Val) : dans X on veut exactement/au_plus/au_moins N éléments dont l'attribut Att a la valeur Val
propriété

position(M,N,Att,Val) : dans M à la position N, on veut un élément dont l'attribut Att a la valeur Val *propriété*

valeurs_différentes_pour(X,Att) : les éléments de X doivent avoir des valeurs différentes pour l'attribut Att *propriété*

valeurs_identiques_pour(X,Att) : les éléments de X doivent avoir des valeurs identiques pour l'attribut Att *propriété*

Prédicats issus de la classification

+ nombre de règles concluant sur ce prédicat

A la racine

élément_générique(P,X) : dans le problème P, on dénombre l'ensemble des X *paquet de 2*
résultat_de_action(P) : dans le problème P, l'ensemble à dénombrer est lié (égalité ou inclusion) à l'ensemble des résultats de l'action du problème *1*
type_action(P,Type) *1*
ens_où_on_choisit(P,E) : pour le problème P, l'ensemble où l'on choisit est E *7*
ensemble_prototypique(P,Prototype) *2*
taille_ens_où_on_choisit(P,T) *3*
attribut_possible(P,Att) : Att est un attribut possible pour le problème P *5*
objet_formé(P,O) *3*
type_objet_formé(P,T) : le type de la collection formée dans le problème P est T *8*
taille_objet_formé(P,N) : la taille de la collection formée dans le problème P est T *6*
remise(P,avec/sans) : le problème P est avec/sans remise *9*
contraintes(P,L) : la liste des contraintes du problème P est la liste L *paquet de 7*
une_seule_contrainte(P) : le problème P a une seule contrainte *1*
contraintes_même_type(P) : toutes les contraintes de P sont du même type *1*
attributs(P,L) : les attributs utilisés dans les contraintes du problème P sont ceux de la liste L *paquet de 2*
un_seul_attribut(P) : le problème P a un seul attribut *1*
taille_catégorie(P,Att,Val,T) : dans P une catégorie pour l'attribut Att valant la valeur Val est de taille T *5*
nombre_catégories(P,Att,N) : pour l'attribut Att, il y a N catégories *5*
type_problème(P,T) : le type du problème P est T (répartition, spectre, liste avec conditions imposées à certaines positions, cas disjoints, complémentaire, produit) *16*

Dans les branches

ensemble_caractéristique(P,L) : L est l'ensemble caractéristique (répartition ou spectre) du problème P
ensemble_caractéristique_part(P) *1*
ensemble_caractéristique_plus_part(P) *1*
nombre_catégories(P,N) : le nombre de catégorie pour le spectre P est N
taille_commune_catégories(P,N) : dans le spectre P, la taille commune de toutes les catégories est T
type_disjonction(P,linéaire/ddp)
cas_part(P) *1*

Les cas particuliers

arrangement(P,N,K) : P est un arrangement de K éléments parmi N
combinaison(P,N,K) : P est une combinaison de K éléments parmi N
permutation(P,N) : le problème P est une permutation de N éléments
produit_cartésien(P,N,K) : P est le produit cartésien N^K

Attributs discriminants

+ *nombre de valeurs possibles*	
type_problème(P,T)	*6*
type_objet_formé(P,T)	*2*
ensemble_caractéristique_part(P)	*booléen*
ensemble_caractéristique_plus_part(P)	*booléen*
remise(P,avec/sans)	*2*
type_disjonction(P,linéaire/ddp)	*2*
cas_part(P)	*booléen*

Attributs utilisés pour la solution et le plan

ensemble caractéristique, taille_ens_où_on_choisit, taille_objet_formé, nombre_catégories, taile_commune_catégories
complémentaire(P,Pall,Pc) : P est le complémentaire de Pc par rapport à Pall
disjonction(P,L)

Résultats

solution(P,N) : N est la solution du problème P
plan(P,Plan)

C2/ Graphe de classification

```
racine_arbre(c_problème).

/*********** LA CLASSE PROBLEME ************************/
```

Le statut de la classe : opérationnelle ou non opérationnelle

```
statut(c_problème,non_opérationnelle).
```

L'attribut discriminant

```
test_fils(c_problème,P,type_problème(P,Type)):-eval_prem(problème(P)).
```

Les sous-classes suivant les valeurs de l'attribut discriminant

```
valeur_fils(c_problème,P,c_répartition):-
            eval_prem(problème(P)),eval_prem(type_problème(P,répartition)).
valeur_fils(c_problème,P,c_liste_position):-
            eval_prem(problème(P)),eval_prem(type_problème(P,liste_position)).
valeur_fils(c_problème,P,c_spectre):-
            eval_prem(problème(P)),eval_prem(type_problème(P,spectre)).
valeur_fils(c_problème,P,c_disjonction):-
            eval_prem(problème(P)),eval_prem(type_problème(P,disjonction)).
valeur_fils(c_problème,P,c_complémentaire):-
            eval_prem(problème(P)),eval_prem(type_problème(P,complémentaire)).
valeur_fils(c_problème,P,c_jointure):-
            eval_prem(problème(P)),eval_prem(type_problème(P,jointure)).
```

Les attributs définis dans la classe, avec les règles de reformulation associées

```
/* l'énoncé */
attribut(c_problème,P,élément_générique(P,Y),élément_générique,paquet(obs_ens),[
]):-        eval_prem(problème(P)).
attribut(c_problème,P,résultat_de_action(P),résultat_de_action,[résultat_de_acti
on],[]):-   eval_prem(problème(P)).

/* l'action */
attribut(c_problème,P,type_action(P,Ta),type_action,[type_action],[]):-
      eval_prem(problème(P)).

/* l'ensemble où l'on choisit */
attribut(c_problème,P,ens_où_on_choisit(P,E),ens_où_on_choisit,[former_au_moyen_
de,attrib_rôles,jeton_former,former_tirer_jeu,jeton_tirer,tirer_jeu,incl_jeu_car
te,jeter_dés],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,ensemble_prototype(P,Type),ensemble_prototype,[alphabet,je
u_de_32_cartes],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,taille_ens_où_on_choisit(P,T),taille_ens_où_on_choisit,[ta
ille_alphabet,taille_jeu,taille_ens],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,attributs_possibles(P,Latt),attributs_possibles,[numérotat
ion,ens_lettres,att_jeu_carte,réunion,att_alphabet,att_dés],[]):-
      eval_prem(problème(P)).

/* l'objet formé */
attribut(c_problème,P,objet_formé(P,O),objet_formé,[former,attribuer,dés,objet],
[]):- eval_prem(problème(P)).
```

```
attribut(c_problème,P,type_objet_formé(P,T),type_objet_formé,[type_précisé,type_
objet,mot,type_formés,type_former_tirer,type_tirer,attrib_ensemble,attrib_liste]
,[]):-      eval_prem(problème(P)).
attribut(c_problème,P,taille_objet_formé(P,T),taille_objet_formé,[taille_précisé
e,taille_objet,taille_former_tirer,taille_tirer,taille_attrib_ens,taille_attrib_
liste],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,remise(P,R),remise,[former_au_plus_exactement,former_avec_
répétitions,former_sans_répétitions,remise_attribution,former_permutation,remise
_former_tirer,remise_tirer,remise_ensemble],[former_défaut]):-
      eval_prem(problème(P)).

/* la liste des contraintes */
attribut(c_problème,P,contraintes(P,Lc),contraintes,paquet(contraintes),[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,une_seule_contrainte(P),une_seule_contrainte,[une_contrain
te],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,contraintes_même_type(P),contraintes_même_type,[même_type]
,[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,attributs(P,Latt),attributs,paquet(attributs),[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,un_seul_attribut(P),un_seul_attribut,[un_att],[]):-
      eval_prem(problème(P)).

/* le type du problème */
attribut(c_problème,P,taille_catégorie(P,Att,Val,T),taille_catégorie,[cat_alphab
et,cat_jeu,cat_numérotation,cat_mot,cat_jetons,cat_dés],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,nombre_catégories(P,Att,T),nombre_catégories,[cat_alphabet
,cat_jeu,cat_numérotation,cat_mot,cat_jetons,cat_dés],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,type_problème(P,Type),type_problème,[concaténation,aucune_
contrainte,attribution,val_id,val_diff,classes,un_effectif,effectifs,effectifs_d
isj,une_position,positions,effectif_au_plus_compl,effectif_au_plus_disj,effectif
_au_moins_compl,effectif_au_moins_disj,effectifs_inter],[]):-
      eval_prem(problème(P)).

/*************** LA CLASSE REPARTITION *********************/

statut(c_répartition,non_opérationnelle).

test_fils(c_répartition,P,type_objet_formé(P,Type)):-eval_prem(problème(P)).

valeur_fils(c_répartition,P,c_répartition_ensemble):-
            eval_prem(problème(P)),eval_prem(type_objet_formé(P,ensemble)).
valeur_fils(c_répartition,P,c_répartition_liste):-
            eval_prem(problème(P)),eval_prem(type_objet_formé(P,liste)).

/* un seul nouvel attribut, déjà déterminé : l'ensemble caractéristique */
attribut(c_répartition,P,liste(P,L),liste,[],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_répartition,c_problème).

infos(c_répartition,P):-write('type du problème : répartition').

/*************** LA CLASSE REPARTITION_ENSEMBLE *********************/

statut(c_répartition_ensemble,opérationnelle).
```

```
test_fils(c_répartition_ensemble,P,ens_carac_part(P)):-eval_prem(problème(P)).

valeur_fils(c_répartition_ensemble,P,c_combinaison):-
            eval_prem(problème(P)),eval_prem(ens_carac_part(P)).

/* nouvel attribut */
attribut(c_répartition_ensemble,P,ens_carac_part(P),ens_carac_part,[ens_carac_pa
rt],[]):-         eval_prem(problème(P)).

/* le père */
père(c_répartition_ensemble,c_répartition).

infos(c_répartition_ensemble,P):-write('type du problème : répartition, type de
l''objet formé : ensemble').

/* la méthode de résolution associée à la classe */
employer_résolution(c_répartition_ensemble):-utiliser([rep_e]).

/*************** LA CLASSE COMBINAISON *********************/

statut(c_combinaison,opérationnelle).

/* pas de fils */

/* pas de nouvel attribut */

/* le père */
père(c_combinaison,c_répartition_ensemble).

infos(c_combinaison,P):-write('type du problème : répartition, type de l''objet
formé : ensemble, combinaison').

/* la méthode de résolution associée à la classe */
employer_résolution(c_combinaison):-utiliser([comb1,comb2]).

/*************** LA CLASSE REPARTITION_LISTE *********************/

statut(c_répartition_liste,non_opérationnelle).

test_fils(c_répartition_liste,P,remise(P,R)):-eval_prem(problème(P)).

valeur_fils(c_répartition_liste,P,c_répartition_liste_avec):-
            eval_prem(problème(P)),eval_prem(remise(P,avec)).
valeur_fils(c_répartition_liste,P,c_répartition_liste_sans):-
            eval_prem(problème(P)),eval_prem(remise(P,sans)).

/*pas de nouveaux attributs */

/* le père */
père(c_répartition_liste,c_répartition).

infos(c_répartition_liste,P):-write('type du problème : répartition, type de
l''objet formé : liste').


/*************** LA CLASSE REPARTITION_LISTE_SANS ****************/

statut(c_répartition_liste_sans,opérationnelle).

test_fils(c_répartition_liste_sans,P,ens_carac_part(P)):-eval_prem(problème(P)).

valeur_fils(c_répartition_liste_sans,P,c_arrangement):-
            eval_prem(problème(P)),eval_prem(ens_carac_part(P)).
```

```
/* nouvel attribut */
attribut(c_répartition_liste_sans,P,ens_carac_part(P),ens_carac_part,[ens_carac_
part],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_répartition_liste_sans,c_répartition_liste).

infos(c_répartition_liste_sans,P):-write('type du problème : répartition, type
de l''objet formé : liste, sans remise').

employer_résolution(c_répartition_liste_sans):-utiliser([rep_l_s]).

/*************** LA CLASSE ARRANGEMENT *****************/

statut(c_arrangement,opérationnelle).

test_fils(c_arrangement,P,ens_carac_plus_part(P)):-eval_prem(problème(P)).

valeur_fils(c_arrangement,P,c_permutation):-
            eval_prem(problème(P)),eval_prem(ens_carac_plus_part(P)).

/* nouvel attribut */
attribut(c_arrangement,P,ens_carac_plus_part(P),ens_carac_plus_part,[ens_carac_p
lus_part],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_arrangement,c_répartition_liste_sans).

infos(c_arrangement,P):-write('type du problème : répartition, type de l''objet
formé : liste, sans remise, arrangement').

/* la méthode de résolution associée à la classe */
employer_résolution(c_arrangement):-utiliser([arr1,arr2]).

/*************** LA CLASSE PERMUTATION **********************/

statut(c_permutation,opérationnelle).

/* pas de fils */

/* pas de nouvel attribut */

/* le père */
père(c_permutation,c_arrangement).

infos(c_permutation,P):-write('type du problème : répartition, type de l''objet
formé : liste, sans remise, arrangement, permutation').

/* la méthode de résolution associée à la classe */
employer_résolution(c_permutation):-utiliser([per]).

/*************** LA CLASSE REPARTITION_LISTE_AVEC *****************/

statut(c_répartition_liste_avec,opérationnelle).

test_fils(c_répartition_liste_avec,P,ens_carac_part(P)):-eval_prem(problème(P)).

valeur_fils(c_répartition_liste_avec,P,c_produit_cartésien):-
            eval_prem(problème(P)),eval_prem(ens_carac_part(P)).
```

```
/* nouvel attribut */
attribut(c_répartition_liste_avec,P,ens_carac_part(P),ens_carac_part,[ens_carac_
part],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_répartition_liste_avec,c_répartition_liste).

infos(c_répartition_liste_avec,P):-write('type du problème : répartition, type
de l''objet formé : liste, avec remise').

employer_résolution(c_répartition_liste_avec):-utiliser([rep_l_a]).

/*************** LA CLASSE PRODUIT CARTESIEN *********************/

statut(c_produit_cartésien,opérationnelle).

/* pas de fils */

/* pas de nouvel attribut */

/* le père */
père(c_produit_cartésien,c_répartition_liste_avec).

infos(c_produit_cartésien,P):-write('type du problème : répartition, type de
l''objet formé : liste, avec remise, produit cartésien').

/* la méthode de résolution associée à la classe */
employer_résolution(c_produit_cartésien):-utiliser([p_c1,p_c2]).

/*************** LA CLASSE LISTE POSITION *********************/

statut(c_liste_position,non_opérationnelle).

test_fils(c_liste_position,P,remise(P,R)):-eval_prem(problème(P)).

valeur_fils(c_liste_position,P,c_liste_position_sans):-
            eval_prem(problème(P)),eval_prem(remise(P,sans)).
valeur_fils(c_liste_position,P,c_liste_position_avec):-
            eval_prem(problème(P)),eval_prem(remise(P,avec)).

/* un seul nouvel attribut, déjà déterminé : l'ensemble caractéristique */
attribut(c_liste_position,P,liste(P,L),liste,[],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_liste_position,c_problème).

infos(c_produit_cartésien,P):-write('type du problème : liste avec conditions de
position').

/*************** LA CLASSE LISTE POSITION SANS *********************/

statut(c_liste_position_sans,opérationnelle).

/* pas de fils */

/* pas de nouvel attribut */

/* le père */
père(c_liste_position_sans,c_liste_position).
```

```
infos(c_liste_position_sans,P):-write('type du problème : liste avec conditions
de position, sans remise').

employer_résolution(c_liste_position_sans):-utiliser([l_p_s]).

/*************** LA CLASSE LISTE POSITION AVEC ********************/

statut(c_liste_position_avec,opérationnelle).

/* pas de fils */

/* pas de nouvel attribut */

/* le père */
père(c_liste_position_avec,c_liste_position).

infos(c_liste_position_avec,P):-write('type du problème : liste avec conditions
de position, avec remise').

employer_résolution(c_liste_position_avec):-utiliser([l_p_a]).

/*************** LA CLASSE SPECTRE  **********************/

statut(c_spectre,non_opérationnelle).

test_fils(c_spectre,P,type_objet_formé(P,Type)):-eval_prem(problème(P)).

valeur_fils(c_spectre,P,c_spectre_ensemble):-
            eval_prem(problème(P)),eval_prem(type_objet_formé(P,ensemble)).
valeur_fils(c_spectre,P,c_spectre_liste):-
            eval_prem(problème(P)),eval_prem(type_objet_formé(P,liste)).

/* 3 nouveaux attributs, déjà déterminés */
/* l'ensemble caractéristique */
attribut(c_spectre,P,liste(P,L),liste,[],[]):-
            eval_prem(problème(P)).
attribut(c_spectre,P,nombre_catégories(P,N),nombre_catégories,[],[]):-
            eval_prem(problème(P)).
attribut(c_spectre,P,taille_commune_catégories(P,T),taille_commune_catégories,[]
,[]):-      eval_prem(problème(P)).

/* le père */
père(c_spectre,c_problème).

infos(c_spectre,P):-write('type du problème : spectre').

/*************** LA CLASSE SPECTRE ENSEMBLE  **********************/

statut(c_spectre_ensemble,opérationnelle).

test_fils(c_spectre_ensemble,P,cas_part(P)):-eval_prem(problème(P)).

valeur_fils(c_spectre_ensemble,P,c_combinaison):-
            eval_prem(problème(P)),eval_prem(cas_part(P)).

/* un nouvel attribut */
attribut(c_spectre_ensemble,P,cas_part(P),cas_part,[cas_part],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_spectre_ensemble,c_spectre).
```

```
infos(c_spectre_ensemble,P):-write('type du problème : spectre, type de l''objet
formé : ensemble').

employer_résolution(c_spectre_ensemble):-utiliser([s_e]).

/*************** LA CLASSE SPECTRE  LISTE *********************/

statut(c_spectre_liste,non_opérationnelle).

test_fils(c_spectre_liste,P,remise(P,R)):-eval_prem(problème(P)).

valeur_fils(c_spectre_liste,P,c_spectre_liste_sans):-
            eval_prem(problème(P)),eval_prem(remise(P,sans)).
valeur_fils(c_spectre_liste,P,c_spectre_liste_avec):-
            eval_prem(problème(P)),eval_prem(remise(P,avec)).

/* pas de nouveaux attributs */

/* le père */
père(c_spectre_liste,c_spectre).

infos(c_spectre_liste,P):-write('type du problème : spectre, type de l''objet
formé : liste').

/*************** LA CLASSE SPECTRE  LISTE SANS *******************/

statut(c_spectre_liste_sans,opérationnelle).

test_fils(c_spectre_liste_sans,P,cas_part(P)):-eval_prem(problème(P)).

valeur_fils(c_spectre_liste_sans,P,c_arrangement):-
            eval_prem(problème(P)),eval_prem(cas_part(P)).

/* un nouvel attribut */
attribut(c_spectre_liste_sans,P,cas_part(P),cas_part,[cas_part],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_spectre_liste_sans,c_spectre_liste).

infos(c_spectre_liste_sans,P):-write('type du problème : spectre, type de
l''objet formé : liste, sans remise').

employer_résolution(c_spectre_liste_sans):-utiliser([s_l_s]).

/*************** LA CLASSE SPECTRE  LISTE AVEC *******************/

statut(c_spectre_liste_avec,opérationnelle).

test_fils(c_spectre_liste_avec,P,cas_part(P)):-eval_prem(problème(P)).

valeur_fils(c_spectre_liste_avec,P,c_arrangement):-
            eval_prem(problème(P)),eval_prem(cas_part(P)).

/* un nouvel attribut */
attribut(c_spectre_liste_avec,P,cas_part(P),cas_part,[cas_part],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_spectre_liste_avec,c_spectre_liste).

infos(c_spectre_liste_avec,P):-write('type du problème : spectre, type de
l''objet formé : liste, avec remise').
```

```
employer_résolution(c_spectre_liste_avec):-utiliser([s_l_a]).

/*************** LA CLASSE JOINTURE  *********************/

statut(c_jointure,opérationnelle).

/* pas de fils */

/* pas de nouveaux attributs */

/* le père */
père(c_jointure,c_problème).

infos(c_jointure,P):-write('type du problème : jointure').

employer_résolution(c_jointure):-
            eval_prem(problème(P)),
            eval_prem(liste_jointure(P,L)),
            construire_pbs_joints(P,L,Lp),
            ajouter_bf(jointure(P,Lp)),
            classer_liste(Lp),
            ajouter_bf(problème(P)),
            construire_liste_sol(Lp,Ls),
            produit(Ls,Sol),
            ajouter_bf(solution(P,Sol)),
            ajouter_bf(plan(P,[['On compte les possibilités pour certains
éléments de l''objet formé suivant une méthode, et les possibilités pour les
autres éléments suivant une autre méthode.']])),!.


/*************** LA CLASSE COMPLEMENTAIRE  *********************/

statut(c_complémentaire,opérationnelle).

/* pas de fils */

/* pas de nouveaux attributs */

/* le père */
père(c_complémentaire,c_problème).

infos(c_complémentaire,P):-write('type du problème : complémentaire').

employer_résolution(c_complémentaire):-
            eval_prem(problème(P)),
            eval_prem(liste_complémentaire(P,N,M,Att,Val)),
            construire_pb_compl(P,N,M,Att,Val,Pc,Pall),
            ajouter_bf(complémentaire(P,Pall,Pc)),
            classer_liste([Pall,Pc]),
            ajouter_bf(problème(P)),
            eval_prem(solution(Pall,S1)),
            eval_prem(solution(Pc,S2)),
            -(S1,S2,Sol),
            ajouter_bf(solution(P,Sol)),
            ajouter_bf(plan(P,[['On compte les résultats qui ne vérifient pas la
propriété demandée, puis on les soustrait du nombre total de résultats
possibles.']])),!.


/*************** LA CLASSE DISJONCTION  *********************/

statut(c_disjonction,non_opérationnelle).
```

```
test_fils(c_disjonction,P,type_disj(P,T)):-eval_prem(problème(P)).

valeur_fils(c_disjonction,P,c_disjonction_linéaire):-
            eval_prem(problème(P)),eval_prem(type_disj(P,linéaire)).
valeur_fils(c_disjonction,P,c_disjonction_ddp):-
            eval_prem(problème(P)),eval_prem(type_disj(P,ddp)).

/* nouvel attribut, déjà déterminé */
attribut(c_disjonction,P,type_disj(P,T),type_disj,[],[]):-
      eval_prem(problème(P)).

/* le père */
père(c_disjonction,c_problème).

infos(c_disjonction,P):-write('type du problème : disjonction').

/*************** LA CLASSE DISJONCTION  LINEAIRE ******************/

statut(c_disjonction_linéaire,opérationnelle).

/* pas de fils */

/* pas de nouveaux attributs */

/* le père */
père(c_disjonction_linéaire,c_disjonction).

infos(c_disjonction_linéaire,P):-write('type du problème : disjonction, type de
la disjonction : linéaire').

employer_résolution(c_disjonction_linéaire):-
            eval_prem(problème(P)),
            eval_prem(liste_disjonction(P,N,M,Att,Val)),
            construire_pb_disj(P,N,M,Att,Val,Lp),
            ajouter_bf(disjonction(P,Lp)),
            classer_liste(Lp),
            ajouter_bf(problème(P)),
            construire_liste_sol(Lp,Ls),
            somme(Ls,Sol),
            ajouter_bf(solution(P,Sol)),
            ajouter_bf(plan(P,[['On envisage plusieurs cas disjoints.']])),!.

/*************** LA CLASSE DISJONCTION  DDP ******************/

statut(c_disjonction_ddp,opérationnelle).

/* pas de fils */

/* pas de nouveaux attributs */

/* le père */
père(c_disjonction_ddp,c_disjonction).

infos(c_disjonction_ddp,P):-write('type du problème : disjonction, type de la
disjonction : dame de pique').
```

```
employer_résolution(c_disjonction_ddp):-
            eval_prem(problème(P)),
            eval_prem(liste_ddp(P,L)),
            construire_pb_avec_sans(P,L,Pavec,Psans),
            ajouter_bf(disjonction(P,[Pavec,Psans])),
            classer_liste([Pavec,Psans]),
            ajouter_bf(problème(P)),
            construire_liste_sol([Pavec,Psans],Ls),
            somme(Ls,Sol),
            ajouter_bf(solution(P,Sol)),
            ajouter_bf(plan(P,[['On envisage plusieurs cas disjoints.']])),!.

/*******************************************************/

/*aller chercher les attributs chez le père*/
attribut(Classe,P,Att,Pred,Liste,Def):-
      père(Classe,Père),attribut(Père,P,Att,Pred,Liste,Def).
```

C3/ Quelques règles de reformulation

Certaines règles de reformulation données à SYRCLAD-dénombrements ont été présentées dans l'annexe A. Nous en présentons ici quelques autres. Le nombre total de règles de reformulation en dénombrements est 85.

```
/************************************************/
/*  Déterminer l'ensemble où l'on choisit */
/************************************************/

/*si on forme en tirant, c'est le contenu du récipient dans lequel on tire */
règle(jeton_former):-
   si(    [problème(P),
           action(P,A),
           est_un(A,action_former),
           en_effectuant(A,T),
           est_un(T,action_tirer),
           récipient(T,S),
           est_un(S,récipient),
           contenu(S,E),
           résultat_de_action(P)]),
   alors([ajouter(ens_où_on_choisit(P,E))]).

/* quand on dispose d'un jeu de cartes dans lequel l'élément générique est
inclus, c'est le jeu de cartes */
règle(incl_jeu_carte):-
   si(    [problème(P),
           on_dispose_de(P,J),
           est_un(J,jeu_de_32_cartes),
           élément_générique(P,Y),
           inclus_dans(Y,J)]),
   alors([ajouter(ens_où_on_choisit(P,J))]).

/************************************************/
/* Observer l'ensemble où l'on choisit   */
/************************************************/

/* pour un ensemble d'éléments numérotés séquentiellement, les catégories
peuvent être formées sur l'attribut numéro ou sur la parité */
règle(numérotation):-
   si(    [problème(P),
           ens_où_on_choisit(P,E),
           taille_ens_où_on_choisit(P,N),
           éléments_numérotés_séquentiellement(E,1,N)]),
   alors([ajouter(attributs_possibles(P,[numéro,parité]))]).

règle(cat_numérotation):-
   si(    [problème(P),
           ens_où_on_choisit(P,E),
           taille_ens_où_on_choisit(P,N),
           éléments_numérotés_séquentiellement(E,1,N)]),
   alors([ajouter(nombre_catégories(P,numéro,N)),
           ajouter(taille_catégorie(P,numéro,_,1)),
           ajouter(nombre_catégories(P,parité,2))]).
```

```
/* pour l'ensemble des lettres d'un mot les attributs peuvent être la lettre ou
voyelle/consonne */
règle(ens_lettres):-
   si(    [problème(P),
           ens_où_on_choisit(P,E),
           est_un(E,ensemble),
           ensemble_des_lettres_de(E,M),
           lettres_toutes_différentes(M)]),
   alors([ajouter(attributs_possibles(P,[lettre,type_lettre]))]).

règle(cat_mot):-
   si(    [problème(P),
           ens_où_on_choisit(P,E),
           est_un(E,ensemble),
           ensemble_des_lettres_de(E,M),
           lettres_toutes_différentes(M),
           taille(M,T)]),
   alors([ajouter(nombre_catégories(P,lettre,T)),
           ajouter(taille_catégorie(P,lettre,_,1)),
           eval_prolog(calcul_type_lettre(P,M))]).

/************************************************/
/*    Déterminer l'objet formé     */
/************************************************/

/* cela peut être le résulat d'une action_former */
règle(former):-
   si(    [problème(P),
           action(P,F),
           est_un(F,action_former),
           tout_résultat(F,R)]),
   alors([ajouter(objet_formé(P,R))]).

/************************************************/
/*    Déterminer le type de l'objet formé      */
/************************************************/

/* si on forme en tirant successivement, c'est une liste */
/*               simultanément, c'est un ensemble */
règle(type_former_tirer):-
   si(    [problème(P),
           action(P,F),
           est_un(F,action_former),
           en_effectuant(F,T),
           est_un(T,action_tirer),
           eval_prolog(mode_tirage(T,M)),
           eval_prolog(type_collection_associé(M,Ta)),
           résultat_de_action(P)]),
   alors([ajouter(type_objet_formé(P,Ta))]).

/************************************************/
/*    Déterminer la taille de l'objet formé    */
/************************************************/
/* si on forme en tirant, c'est le nombre d'éléments tirés */
règle(taille_former_tirer):-
   si(    [problème(P),
           action(P,F),
           est_un(F,action_former),
           en_effectuant(F,T),
           est_un(T,action_tirer),
           nb_éléments_tirés(T,N),
           résultat_de_action(P)]),
   alors([ajouter(taille_objet_formé(P,N))]).
```

```
/*************************************************/
/*    On détermine la remise             */
/*************************************************/

/* si on forme un résultat où chaque élément apparaît au plus/exactement une
fois, c'est sans */
règle(former_au_plus_exactement):-
   si(    [problème(P),
           objet_formé(P,X),
           eval_prolog(member(Z,[au_plus,exactement])),
           effectif_chaque_élément(X,Z,1),
           résultat_de_action(P)]),
   alors([ajouter(remise(P,sans))]).

/*************************************************/
/*     Etablir la liste des contraintes        */
/*************************************************/

/* on traduit cette contrainte que tous les éléments aient la couleur C par le
prédicat effectif_attribut utilisé habituellement */
règle(j5,contraintes):-
   si(    [problème(P),
           élément_générique(P,Y),
           tout_élément(Y,X),
           couleur(X,C),
           taille_objet_formé(P,N)]),
   alors([ajouter(effectif_attribut(Y,exactement,N,couleur,C))]).

/*** Rapporter les contraintes aux éléments à dénombrer ***/

règle(rapporter4,contraintes):-
   si(    [problème(P),
           objet_formé(P,R),
           eval_prolog(est_une_contrainte(M,4)),
           M(R,N,T,L),
           résultat_de_action(P),
           élément_générique(P,Y)]),
   alors([ajouter(M(Y,N,T,L))]).

/*** Etablir la liste des contraintes ***/
/***Toutes les contraintes sont maintenant "attachées" à Y, il suffit d'en faire
la liste ***/

/* on initialise à la liste vide */
règle(liste_vide,contraintes):-
   si(    [problème(P)]),
   alors([eval_prolog(retirer_tous(contraintes(P,_))),
           ajouter(contraintes(P,[]))]).

/* les prédicats de contrainte d'arité 5 */
règle(établir5,contraintes):-
/* les prédicats de contrainte d'arité 4 */
règle(établir4,contraintes):-

/* les prédicats de contrainte d'arité 2 */
règle(établir2,contraintes):-
   si(    [problème(P),
           élément_générique(P,Y),
           eval_prolog(est_une_contrainte(M,2)),
           M(Y,A)]),
   alors([modifier(contraintes(P,Liste),
           contraintes(P,[M(A)|Liste]))]).
```

```
/*************************************************/
/*     Observer la liste des contraintes */
/*************************************************/

/* la liste des contraintes est réduite à un seul élément */
règle(une_contrainte):-
   si(    [problème(P),
           contraintes(P,[_])]),
   alors([ajouter(une_seule_contrainte(P))]).

/* les contraintes sont toutes de même type (même prédicat) */
règle(même_type):-
   si(    [problème(P),
           contraintes(P,L),
           eval_prolog(même_prédicat(L))]),
   alors([ajouter(contraintes_même_type(P))]).

/***  Construire la liste des attributs concernés    *****/

/* on initialise à la liste vide */
règle(liste_att,attributs):-
   si(    [problème(P),
           contraintes(P,L)]),
   alors([eval_prolog(retirer_tous(attributs(P,_))),
           ajouter(attributs(P,[]))]).

/* repérer tous attributs pouvant figurer dans les prédicats de la liste des
contrainte, et en construire la liste */
règle(cons_att,attributs):-

/**** Observer la liste des attributs concernés *****/

/* la liste des attributs est réduite à un seul élément */
règle(un_att):-
   si(     [problème(P),
            attributs(P,[_])]),
   alors(  [ajouter(un_seul_attribut(P))]).

/*************************************************/
/*     Conclure sur le type du tirage          */
/*************************************************/

/* lorsque le résultat de l'action est une concaténation, le problème est une
jointure */
règle(concaténation):-
   si(    [problème(P),
           objet_formé(P,R),
           est_un(R,concaténation),
           concaténation(R,L),
           résultat_de_action(P)]),
   alors([ajouter(type_problème(P,jointure)),
           ajouter(liste_jointure(P,L))]).
```

```
/* il y a une seule contrainte : effectif_attribut(au_plus,....). dans le cas où
c'est le plus avantageux, on utilise le complémentaire */
règle(effectif_au_plus_compl):-
   si(    [problème(P),
           une_seule_contrainte(P),
           contraintes(P,[effectif_attribut(au_plus,N,Att,Val)]),
           eval_prolog(attribut_possible(P,Att)),
           taille_objet_formé(P,M),
           eval_prolog(-(M,N,MmN)),
           eval_prolog(<(MmN,N+1))]),
   alors([ajouter(type_problème(P,complémentaire)),
           eval_prolog(+(N,1,Np1)),
           ajouter(liste_complémentaire(P,Np1,M,Att,Val))]).

/* il y a une seule contrainte : effectif_attribut(au_plus,....). dans le cas où
c'est le plus avantageux, on utilise la disjonction */
règle(effectif_au_plus_disj):-
/* il y a une seule contrainte : effectif_attribut(au_moins,....). dans le cas
où c'est le plus avantageux, on utilise le complémentaire */
règle(effectif_au_moins_compl):-
/* il y a une seule contrainte : effectif_attribut(au_moins,....). dans le cas
où c'est le plus avantageux, on utilise la disjonction*/
règle(effectif_au_moins_disj):-

/* l'exercice dit de "la dame de pique" : les contraintes portent sur deux
attributs différents, les catégories formées par les contraintes ont une
intersection réduite à un élément */
règle(effectifs_inter):-
   si(    [problème(P),
           eval_prolog(not(fait(une_seule_contrainte(P)))),
           contraintes(P,[effectif_attribut(exactement,_,_,_)|_]),
           contraintes_même_type(P),
           eval_prolog(not(fait(un_seul_attribut(P)))),
           eval_prolog(construire_liste_nb_att_val(P,L)),
           eval_prolog(dame_de_pique(P,L))]),
   alors([ajouter(type_problème(P,disjonction)),
           ajouter(type_disj(P,ddp)),
           ajouter(déterminé(P,type_disj)),
           ajouter(liste_ddp(P,L))]).

/* il n'y a aucune contrainte : c'est une répartition */
règle(aucune_contrainte):-
   si(    [problème(P),
           eval_prolog(not(fait(type_problème(P,_)))),
           contraintes(P,[]),
           taille_ens_où_on_choisit(P,T),
           taille_objet_formé(P,N)]),
   alors([ajouter(type_problème(P,répartition)),
           ajouter(liste(P,[[T,N]]))]).

/* il y a une seule contrainte : valeur identique pour un attribut. C'est un spectre
règle(val_id):-
   si(    [problème(P),
           taille_objet_formé(P,K),
           une_seule_contrainte(P),
           contraintes(P,[valeurs_identiques_pour(Att)]),
           eval_prolog(attribut_possible(P,Att)),
           nombre_catégories(P,Att,N),
           taille_catégorie(P,Att,_,T)]),
   alors([ajouter(type_problème(P,spectre)),
           ajouter(nombre_catégories(P,N)),
           ajouter(taille_commune_catégories(P,T)),
           ajouter(liste(P,[[1,K]]))]).
```

```
/* il y a une seule contrainte : valeurs différentes pour un attribut. C'est un
spectre */
règle(val_diff):-
   si(    [problème(P),
           taille_objet_formé(P,K),
           une_seule_contrainte(P),
           contraintes(P,[valeurs_différentes_pour(Att)]),
           eval_prolog(attribut_possible(P,Att)),
           nombre_catégories(P,Att,N),
           taille_catégorie(P,Att,_,T)]),
   alors([ajouter(type_problème(P,spectre)),
           ajouter(nombre_catégories(P,N)),
           ajouter(taille_commune_catégories(P,T)),
           ajouter(liste(P,[[K,1]]))]).

/* on a plusieurs contraintes de même type classes_effectif_attribut, concernant
un seul attribut, c'est un spectre dont on construit l'ensemble caractéristique
*/
règle(classes):-
si(     [problème(P),
           contraintes(P,[classes_effectif_attribut(X,Y,Att)|_]),
           eval_prolog(attribut_possible(P,Att)),
           eval_prolog(not(fait(une_seule_contrainte(P)))),
           contraintes_même_type(P),
           un_seul_attribut(P),
           nombre_catégories(P,Att,N),
           taille_catégorie(P,Att,_,T)]),
   alors([ajouter(type_problème(P,spectre)),
           ajouter(nombre_catégories(P,N)),
           ajouter(taille_commune_catégories(P,T)),
           eval_prolog(construire_liste1(P,L)),      /* construit la liste des
couples (X,Y) */
           ajouter(liste(P,L))]).

/*il y a une seule contrainte de type effectif_attribut,c'est une répartition */
règle(un_effectif):-
   si(    [problème(P),
           une_seule_contrainte(P),
           contraintes(P,[effectif_attribut(exactement,N,Att,Val)]),
           eval_prolog(attribut_possible(P,Att)),
           taille_catégorie(P,Att,Val,T)]),
   alors([ajouter(type_problème(P,répartition)),
           ajouter(liste(P,[[T,N]]))]).

/* on a plusieurs contraintes de même type effectif_attribut(exactement,....)
portant sur le même attribut. C'est une répartition dont on construit l'ensemble
caractéristique */
règle(effectifs):-
   si(    [problème(P),
           eval_prolog(not(fait(une_seule_contrainte(P)))),
           contraintes(P,[effectif_attribut(exactement,_,_,_)|_]),
           contraintes_même_type(P),
           un_seul_attribut(P),
           eval_prolog(construire_liste_nb_att_val(P,L))]),
   alors([ajouter(type_problème(P,répartition)),
           eval_prolog(construire_liste2(P,L,Lr)),
           ajouter(liste(P,Lr))]).
```

```
/* on a plusieurs contraintes de même type effectif_attribut(exactement,....)
portant sur des attributs différents. C'est une répartition dont on construit
l'ensemble caractéristique en vérifiant que les catégories sont disjointes */
règle(effectifs_disj):-
   si(    [problème(P),
           eval_prolog(not(fait(une_seule_contrainte(P)))),
           contraintes(P,[effectif_attribut(exactement,_,_,_)|_]),
           contraintes_même_type(P),
           eval_prolog(not(fait(un_seul_attribut(P)))),
           eval_prolog(construire_liste_nb_att_val(P,L)), /* "exactement" */
           eval_prolog(cat_disjointes(P,L))]) ,/* pour ttes les contraintes */
   alors([ajouter(type_problème(P,répartition)),
           eval_prolog(construire_liste2(P,L,Lr)),
           ajouter(liste(P,Lr))]).

/* une seule contrainte de type position*/
règle(une_position):-
   si(    [problème(P),
           une_seule_contrainte(P),
           contraintes(P,[position(_,Att,Val)]),
           eval_prolog(attribut_possible(P,Att)),
           taille_catégorie(P,Att,Val,T)]),
   alors([ajouter(type_problème(P,liste_position)),
           ajouter(liste(P,[[T,1]]))]).

/* contraintes de type position*/
règle(positions):-
   si(    [problème(P),
           eval_prolog(not(fait(une_seule_contrainte(P)))),
           contraintes_même_type(P),
           un_seul_attribut(P),
           contraintes(P,[position(N,Att,_)|_]),
           eval_prolog(attribut_possible(P,Att))]),
   alors([ajouter(type_problème(P,liste_position)),
           eval_prolog(construire_liste_position(P,L)),
           ajouter(liste(P,L))]).

/************************************************/
/*     Repérer les cas particuliers      */
/************************************************/

/* ensemble caractéristique réduit*/
règle(ens_carac_part):-
   si(    [problème(P),
           taille_objet_formé(P,N),
           liste(P,[[T,N]])]),
   alors([ajouter(ens_carac_part(P))]).

/* ensemble caractéristique réduit*/
règle(ens_carac_plus_part):-
   si(    [problème(P),
           taille_objet_formé(P,N),
           liste(P,[[N,N]])]),
   alors([ajouter(ens_carac_plus_part(P))]).

/* ensemble caractéristique réduit*/
règle(cas_part):-
   si(    [problème(P),
           taille_commune_catégories(P,1),
           taille_objet_formé(P,N),
           liste(P,[[N,1]])]),
   alors([ajouter(cas_part(P))]).
```

C4/ Les plans-type et les formules associés à certaines classes opérationnelles

Nous présentons les problèmes-type correspondant aux classes du graphe de classification qui ne sont ni des classes à résolution composée ni des cas particuliers. Pour chacune de ces classes, nous présentons le plan-type associé au problème-type et la formule de calcul de la solution numérique.

Classe répartition-ensemble

Problème type

Soit un ensemble E de taille e divisé en m catégories disjointes d'effectifs c1, …, cm. On tire p éléments dans E simultanément, et on forme un ensemble. Combien y a-t-il de résultats contenant exactement x_1 éléments de catégorie 1, …, x_n éléments de catégorie n ?

Plan type

On choisit x_1 <description catégorie 1> parmi c1,
puis on choisit x_2 <description catégorie 2> parmi c_2,
etc.,
puis on complète en choisissant p - $(x_1+x_2+\ldots+x_n)$ éléments parmi les e - $(c_1+c_2+\ldots+c_n)$ éléments restants.

Formule

$$C_{c_1}^{x_1} \ldots C_{c_n}^{x_n} C_{e-(c_1+\ldots+c_n)}^{p-(x_1+\ldots+x_n)}$$

Classe répartition-liste-avec_remise

Problème type

Soit un ensemble E de taille e divisé en m catégories disjointes d'effectifs c1, …, cm. On tire p éléments dans E successivement, avec remise, et on forme une liste. Combien y a-t-il de résultats contenant exactement x_1 éléments de catégorie 1, …, x_n éléments de catégorie n ?

Plan type

On choisit x_1 places parmi p pour les <description catégorie 1>, puis on choisit un <description catégorie 1> parmi c_1 pour chacune de ces places,

puis on choisit x_2 places parmi p - x_1 pour les <description catégorie 2>, puis on choisit un <description catégorie 2> parmi c_2 pour chacune de ces places,

etc.,

puis on complète les p - $(x_1+x_2+\ldots+x_n)$ places restantes en choisissant p - $(x_1+x_2+\ldots+x_n)$ éléments parmi les e - $(c_1+c_2+\ldots+c_n)$ éléments restants.

Formule

$$\frac{p!{c_1}^{x_1}\ldots{c_n}^{x_n}(e-(c_1+\ldots+c_n))^{p-(x_1+\ldots+x_n)}}{x_1!x_2!\ldots x_n!(p-(x_1+\ldots+x_n))!}$$

Classe répartition-liste-sans_remise

Problème type

Soit un ensemble E de taille e divisé en m catégories disjointes d'effectifs c_1, …, c_m. On tire p éléments dans E successivement, sans remise, et on forme une liste. Combien y a-t-il de résultats contenant exactement x_1 éléments de catégorie 1, …, x_n éléments de catégorie n ?

Plan type

On choisit x_1 places parmi p pour les <description catégorie 1>, puis on choisit un <description catégorie 1> parmi c_1 pour chacune de ces places, tous différents,

puis on choisit x_2 places parmi p - x_1 pour les <description catégorie 2>, puis on choisit un <description catégorie 2> parmi c_2 pour chacune de ces places, tous différents,

etc.,

puis on complète les p - $(x_1+x_2+\ldots+x_n)$ places restantes en choisissant p - $(x_1+x_2+\ldots+x_n)$ éléments tous différents parmi les e - $(c_1+c_2+\ldots+c_n)$ éléments restants.

Formule

$$\frac{p!A_{c1}^{x1}\ldots A_{cn}^{xn}A_{e-(c1+\ldots+cn)}^{p-(x1+\ldots+xn)}}{x1!x2!\ldots xn!(p-(x1+\ldots+xn))!}$$

Liste_avec_conditions_de_position-avec_remise

Problème type

On tire p éléments successivement et avec remise dans un ensemble E à e éléments, partitionné en m catégories disjointes d'effectifs c_1, …, c_m. On forme une liste. Combien y a-t-il de listes telles que x_1 éléments à des positions données Ep_1 appartiennnent à la catégorie 1, x_2 éléments à des positions

données Ep_2 appartiennnent à la catégorie 2, …, x_n éléments à des positions données Ep_n appartiennnent à la catégorie n (pas de resctriction pour les autres) ?

Plan type

On choisit un <description catégorie 1> parmi c_1 pour chacune des positions Ep_1
puis on choisit un <description catégorie 2> parmi c_2 pour chacune des positions Ep_2
etc.,
puis on complète les p - $(x_1+x_2+\ldots+x_n)$ places restantes en choisissant p - $(x_1+x_2+\ldots+x_n)$ éléments parmi les e éléments de E.

Formule

$$c_1^{x1}\ldots c_n^{xn}e^{p-(x1+\ldots+xn)}$$

Liste_avec_conditions_de_position-sans_remise

Problème type

On tire p éléments successivement et sans remise dans un ensemble E à e éléments, partitionné en m catégories disjointes d'effectifs c_1, …, c_m. On forme une liste. Combien y a-t-il de listes telles que x_1 éléments à des positions données Ep_1 appartiennnent à la catégorie 1, x_2 éléments à des positions données Ep_2 appartiennnent à la catégorie 2, …, x_n éléments à des positions données Ep_n appartiennnent à la catégorie n (pas de resctriction pour les autres) ?

Plan type

On choisit x_1 <description catégorie 1> parmi c_1, tous différents, pour les positions Ep_1,
puis on choisit x_2 <description catégorie 2> parmi c_2, tous différents, pour les positions Ep_2,
etc.,
puis on complète les p - $(x_1+x_2+\ldots+x_n)$ places restantes en choisissant p - $(x_1+x_2+\ldots+x_n)$ éléments tous différents parmi les e - $(x_1+x_2+\ldots+x_n)$ éléments restants.

Formule

$$A_{c1}^{x1}\ldots A_{cn}^{xn}A_{e-(x1+\ldots+xn)}^{p-(x1+\ldots+xn)}$$

Spectre-ensemble

Problème type

Soit un ensemble E de taille e divisé en m catégories disjointes de même effectif c. On tire p éléments dans E simultanément, et on forme un ensemble. On sintéresse à la partition de cet ensemble en classes d'éléments de même catégorie. Combien y a-t-il de résultats contenant k_1 classes de x_1 éléments, …, k_n classes de x_n éléments ? ($k_1i_1 + \ldots + k_ni_n = p$)

Plan type

On choisit k_1 <description catégories> parmi m, dans chacune on choisit x_1 éléments parmi c,

puis on choisit k_2 <description catégories> parmi m, dans chacune on choisit x_2 éléments parmi c, etc.,

puis on choisit k_n <description catégories> parmi m, dans chacune on choisit x_n éléments parmi c.

Formule

$$\frac{m!}{k1!k2!\ldots kn!(m-(k1+\ldots+kn))!}(C_{c_1}^{x_1})^{k1}\ldots(C_{c_n}^{x_n})^{kn}$$

Spectre-liste-avec_remise

Problème type

Soit un ensemble E de taille e divisé en m catégories disjointes de même effectif c. On tire p éléments dans E successivement et avec remise, et on forme une liste. On sintéresse à la partition de cet ensemble en classes d'éléments de même catégorie. Combien y a-t-il de résultats contenant k_1 classes de x_1 éléments, …, k_n classes de x_n éléments ? ($k_1i_1 + \ldots + k_ni_n = p$)

Plan type

On choisit k_1 <description catégories> parmi m, pour chacun(e) on choisit x_1 places, puis on remplit chacune de ces places avec un élément pris parmi c,

puis on choisit k_2 <description catégories> parmi m, pour chacun(e) on choisit x_2 places, puis on remplit chacune de ces places avec un élément pris parmi c,

etc.,

puis on choisit k_n <description catégories> parmi pour chacun(e) on choisit x_n places, puis on remplit chacune de ces places avec un élément pris parmi c.

Formule

$$\frac{m!}{k1!k2!\ldots kn!(m-(k1+\ldots+kn))!}\frac{p!}{i1!^{k1}\,i2!^{k2}\ldots in!^{kn}}c^{p}$$

Spectre-liste-sans_remise

Problème type

Soit un ensemble E de taille e divisé en m catégories disjointes de même effectif c. On tire p éléments dans E successivement et sans remise, et on forme une liste. On sintéresse à la partition de cet ensemble en classes d'éléments de même catégorie. Combien y a-t-il de résultats contenant k_1 classes de x_1 éléments, …, k_n classes de x_n éléments ? ($k_1i_1 + \ldots + k_ni_n = p$)

Plan type

On choisit k_1 <description catégories> parmi m, pour chacun(e) on choisit x_1 places, puis on remplit ces places avec x_1 éléments distincts pris parmi c,

puis on choisit k_2 <description catégories> parmi m, pour chacun(e) on choisit x_2 places, puis on remplit ces places avec x_2 éléments distincts pris parmi c,

etc.,

puis on choisit k_n <description catégories> parmi pour chacun(e) on choisit x_n places, puis on remplit ces places avec x_n éléments distincts pris parmi c.

Formule

$$\frac{m!}{k1!k2!\ldots kn!(m-(k1+\ldots+kn))!}\frac{p!}{i1!^{k1}\,i2!^{k2}\ldots in!^{kn}}c^{p}$$

Annexe D
Connaissances données à SYRCLAD-additifs

D1/ Prédicats du langage de description et prédicats issus de la classification

Prédicats intervenant dans les modèles descriptifs

problème(P) *objet*
à_calculer(Problème,Entier) *relation + inconnue*
est_un(X,ensemble/personne/bille/partie/succession_de_parties) *objets*
possède(Pers,Ens) *relation*
tout_élément(Ens,X) *relation*
taille(Ens,N) *propriété*
ensemble(Pers1,Pers2,Pers) *relation*
action(Problème,Partie) *relation*
participants(Partie,Pers1,Pers2) *relation*
partie(Partie,Instant1,Instant2) *relation*
perdu(Pers,Partie,Ens) *relation*
gagné(Pers,Partie,Ens) *relation*
changement(Pers,Partie,Ens) *relation*
succession(Partie,Lparties) *relation*
6 objets, 11 relations, 1 propriété

Prédicats issus de la classification

+ nombre de règles concluant sur cet attribut

A la racine

nature(Problème,simple/composé) *3*
intéressé(Problème,Personne) *3*
à_trouver(Problème,résultat/opérande/opérateur/double_opérateur/un_opérateur) *8*
donnée1(Problème,Entier)
donnée2(Problème,Entier)
signe(Problème,positif/négatif) *2*

Dans les branches

opérateur(Problème,add/ajout/ret) *5*
opérateurs(Problème,Liste_op) partie double *paquet de 3*
type_opérateurs(Problème,même_type/types_différents) *2*

nom_opérateur(Problème,Op) opérateurs de même type *1*
manquant(Problème,Op) *2*
ajout(Problème,Entier) *1*
ret(Problème,Entier) *1*

Attributs discriminants

+ nombre de valeurs possibles
nature(Problème,simple/composé) *2*
opérateur(Problème,add/ajout/ret) *3*
à_trouver(Problème,résultat/opérande/opérateur/double_opérateur/un_opérateur) 5
type_opérateurs(Problème,même_type/types_différents) *2*
nom_opérateur(Problème,Op) opérateurs de même type *2*
signe(Problème,positif/négatif) *2*
manquant(Problème,Op) *2*

Résultats

solution(Problème,Sol)
plan(Problème,Plan)

Attributs utilisés pour la solution et le plan

donnée1, donnée2

D2/ Graphe de classification

```
racine_arbre(c_problème).

/*************** LA CLASSE PROBLEME ****************************/

statut(c_problème,non_opérationnelle).

test_fils(c_problème,P,nature(P,N)):-eval_prem(problème(P)).

valeur_fils(c_problème,P,c_simple):-
            eval_prem(problème(P)),eval_prem(nature(P,simple)).
valeur_fils(c_problème,P,c_composé):-
            eval_prem(problème(P)),eval_prem(nature(P,composé)).

attribut(c_problème,P,nature(P,N),nature,[nature_simple1,nature_simple2,nature_c
omposé][]):-
            eval_prem(problème(P)).
attribut(c_problème,P,intéressé(P,Pi),intéressé,[i2,i3,i4],[]):-
            eval_prem(problème(P)).
attribut(c_problème,P,à_trouver(P,Q),à_trouver,[at1,at2,at3,at4,at5,at6,at7,at8]
,[]):-
            eval_prem(problème(P)).
attribut(c_problème,P,donnée1(P,A),donnée1,[],[]):-
            eval_prem(problème(P)).
attribut(c_problème,P,donnée2(P,B),donnée2,[],[]):-
            eval_prem(problème(P)).
attribut(c_problème,P,signe(P,S),signe,[s1,s2]),[]:-
            eval_prem(problème(P)).

infos(c_problème,P):-write('aucune information sur le problème').

/*************** LA CLASSE SIMPLE ************************/

statut(c_simple,non_opérationnelle).

test_fils(c_simple,P,opérateur(P,O)):-eval_prem(problème(P)).

valeur_fils(c_simple,P,c_simple_add):-
            eval_prem(problème(P)),eval_prem(opérateur(P,add)).
valeur_fils(c_simple,P,c_simple_ajout):-
            eval_prem(problème(P)),eval_prem(opérateur(P,ajout)).
valeur_fils(c_simple,P,c_simple_ret):-
            eval_prem(problème(P)),eval_prem(opérateur(P,ret)).

attribut(c_simple,P,opérateur(P,O),opérateur,[op_add,op_ajout1,op_ajout2,op_ret1
,op_ret2],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_simple,c_problème).

infos(c_simple,P):-write('problème simple : une seule partie').

/************ LA CLASSE SIMPLE_ADD ********************/

statut(c_simple_add,non_opérationnelle).

test_fils(c_simple_add,P,à_trouver(P,Q)):-eval_prem(problème(P)).
```

```
valeur_fils(c_simple_add,P,c_simple_add_res):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,résultat)).
valeur_fils(c_simple_add,P,c_simple_add_opérande):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,opérande)).

/* le père */
père(c_simple_add,c_simple).

infos(c_simple_add,P):-write('problème simple : une seule partie, opérateur :
ADD').

/************* LA CLASSE SIMPLE_AJOUT **********************/

statut(c_simple_ajout,non_opérationnelle).

test_fils(c_simple_ajout,P,à_trouver(P,Q)):-eval_prem(problème(P)).

valeur_fils(c_simple_ajout,P,c_simple_ajout_res):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,résultat)).
valeur_fils(c_simple_ajout,P,c_simple_ajout_opérande):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,opérande)).
valeur_fils(c_simple_ajout,P,c_simple_ajout_opérateur):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,opérateur)).

/* le père */
père(c_simple_ajout,c_simple).

infos(c_simple_ajout,P):-write('problème simple : une seule partie, opérateur :
AJOUT').

/************* LA CLASSE SIMPLE_RET ****************************/

statut(c_simple_ret,non_opérationnelle).

test_fils(c_simple_ret,P,à_trouver(P,Q)):-eval_prem(problème(P)).

valeur_fils(c_simple_ret,P,c_simple_ret_res):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,résultat)).
valeur_fils(c_simple_ret,P,c_simple_ret_opérande):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,opérande)).
valeur_fils(c_simple_ret,P,c_simple_ret_opérateur):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,opérateur)).

/* le père */
père(c_simple_ret,c_simple).

infos(c_simple_ret,P):-write('problème simple : une seule partie, opérateur :
RET').

/************** LA CLASSE SIMPLE_ADD_RES *********************/

statut(c_simple_add_res,opérationnelle).
type_résultat(c_simple_add_res,ensemble).
type_données(c_simple_add_res,[ensemble,ensemble]).

/* le père */
père(c_simple_add_res,c_simple_add).

infos(c_simple_add_res,P):-write('problème simple : une seule partie, opérateur
: ADD, à trouver : le résultat').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_add_res):-utiliser([sol1]).
```

```
/************** LA CLASSE SIMPLE_ADD_OPERANDE ********************/

statut(c_simple_add_opérande,opérationnelle).
type_résultat(c_simple_add_opérande,ensemble).
type_données(c_simple_add_opérande,[ensemble,ensemble]).

/* le père */
père(c_simple_add_opérande,c_simple_add).

infos(c_simple_add_opérande,P):-write('problème simple : une seule partie,
opérateur : ADD, à trouver : l''opérande').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_add_opérande):-utiliser([sol2]).

/************** LA CLASSE SIMPLE_AJOUT_RES ********************/

statut(c_simple_ajout_res,opérationnelle).
type_résultat(c_simple_ajout_res,ensemble).
type_données(c_simple_ajout_res,[ensemble,opérateur]).

/* le père */
père(c_simple_ajout_res,c_simple_ajout).

infos(c_simple_ajout_res,P):-write('problème simple : une seule partie,
opérateur : AJOUT, à trouver : le résultat').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_ajout_res):-utiliser([sol3]).

/************** LA CLASSE SIMPLE_AJOUT_OPERANDE ********************/

statut(c_simple_ajout_opérande,opérationnelle).
type_résultat(c_simple_ajout_opérande,ensemble).
type_données(c_simple_ajout_opérande,[ensemble,opérateur]).

/* le père */
père(c_simple_ajout_opérande,c_simple_ajout).

infos(c_simple_ajout_opérande,P):-write('problème simple : une seule partie,
opérateur : AJOUT, à trouver : l''opérande').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_ajout_opérande):-utiliser([sol4]).

/************** LA CLASSE SIMPLE_AJOUT_OPERATEUR********************/

statut(c_simple_ajout_opérateur,opérationnelle).
type_résultat(c_simple_ajout_opérateur,opérateur).
type_données(c_simple_ajout_opérateur,[ensemble,ensemble]).

/* le père */
père(c_simple_ajout_opérateur,c_simple_ajout).

infos(c_simple_ajout_opérateur,P):-write('problème simple : une seule partie,
opérateur : AJOUT, à trouver : l''opérateur').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_ajout_opérateur):-utiliser([sol5]).
```

```
/************** LA CLASSE SIMPLE_RET_RES********************/

statut(c_simple_ret_res,opérationnelle).
type_résultat(c_simple_ret_res,ensemble).
type_données(c_simple_ret_res,[ensemble,opérateur]).

/* le père */
père(c_simple_ret_res,c_simple_ret).

infos(c_simple_ret_res,P):-write('problème simple : une seule partie, opérateur
: RET, à trouver : le résultat').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_ret_res):-utiliser([sol6]).

/************** LA CLASSE SIMPLE_RET_OPERANDE********************/

statut(c_simple_ret_opérande,opérationnelle).
type_résultat(c_simple_ret_opérande,ensemble).
type_données(c_simple_ret_opérande,[ensemble,opérateur]).

/* le père */
père(c_simple_ret_opérande,c_simple_ret).

infos(c_simple_ret_opérande,P):-write('problème simple : une seule partie,
opérateur : RET, à trouver : l''opérande').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_ret_opérande):-utiliser([sol7]).

/************** LA CLASSE SIMPLE_RET_OPERATEUR ******************/

statut(c_simple_ret_opérateur,opérationnelle).
type_résultat(c_simple_ret_opérateur,opérateur).
type_données(c_simple_ret_opérateur,[ensemble,ensemble]).

/* le père */
père(c_simple_ret_opérateur,c_simple_ret).

infos(c_simple_ret_opérateur,P):-write('problème simple : une seule partie,
opérateur : RET, à trouver : l''opérateur').

/* la méthode de résolution associée à la classe */
employer_résolution(c_simple_ret_opérateur):-utiliser([sol8]).

/*************** LA CLASSE COMPOSE   ************************/

statut(c_composé,non_opérationnelle).

test_fils(c_composé,P,type_opérateurs(P,T)):-eval_prem(problème(P)).

valeur_fils(c_composé,P,c_composé_mtype):-
            eval_prem(problème(P)),eval_prem(type_opérateurs(P,même_type)).
valeur_fils(c_composé,P,c_composé_typediff):-

      eval_prem(problème(P)),eval_prem(type_opérateurs(P,types_différents)).

attribut(c_composé,P,opérateurs(P,T),opérateurs,paquet(op),[]):-
            eval_prem(problème(P)).
attribut(c_composé,P,type_opérateurs(P,T),type_opérateurs,[top1,top2],[]):-
            eval_prem(problème(P)).
```

```
/* le père */
père(c_composé,c_problème).

infos(c_composé,P):-write('problème composé : deux parties').

/*************** LA CLASSE COMPOSE_MTYPE   ***********************/

statut(c_composé_mtype,non_opérationnelle).

test_fils(c_composé_mtype,P,nom_opérateur(P,T)):-eval_prem(problème(P)).

valeur_fils(c_composé_mtype,P,c_composé_mtype_ajout):-
            eval_prem(problème(P)),eval_prem(nom_opérateur(P,ajout)).
valeur_fils(c_composé_mtype,P,c_composé_mtype_ret):-
            eval_prem(problème(P)),eval_prem(nom_opérateur(P,ret)).

attribut(c_composé,P,nom_opérateur(P,T),nom_opérateur,[nop],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_composé_mtype,c_composé).

infos(c_composé_mtype,P):-write('problème composé : deux parties, opérateurs de
même type').

/*************** LA CLASSE COMPOSE_MTYPE_AJOUT   ***********************/

statut(c_composé_mtype_ajout,non_opérationnelle).

test_fils(c_composé_mtype_ajout,P,à_trouver(P,T)):-eval_prem(problème(P)).

valeur_fils(c_composé_mtype_ajout,P,c_composé_mtype_ajout_double):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,double_opérateur)).
valeur_fils(c_composé_mtype_ajout,P,c_composé_mtype_ajout_un):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,un_opérateur)).

/* le père */
père(c_composé_mtype_ajout,c_composé_mtype).

infos(c_composé_mtype_ajout,P):-write('problème  composé  :  deux  parties,
opérateurs de même type, type : AJOUT').

/*************** LA CLASSE COMPOSE_MTYPE_AJOUT_DOUBLE  ***********************/

statut(c_composé_mtype_ajout_double,opérationnelle).
type_résultat(c_composé_mtype_ajout_double,opérateur).
type_données(c_composé_mtype_ajout_double,[opérateur,opérateur]).

/* le père */
père(c_composé_mtype_ajout_double,c_composé_mtype_ajout).

infos(c_composé_mtype_ajout_double,P):-write('problème composé : deux parties,
opérateurs de même type, type : AJOUT, à trouver : l''opérateur résultant').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_mtype_ajout_double):-utiliser([sol9]).

/*************** LA CLASSE COMPOSE_MTYPE_AJOUT_UN   ***********************/

statut(c_composé_mtype_ajout_un,non_opérationnelle).

test_fils(c_composé_mtype_ajout_un,P,signe(P,T)):-eval_prem(problème(P)).
```

```
valeur_fils(c_composé_mtype_ajout_un,P,c_composé_mtype_ajout_un_pos):-
            eval_prem(problème(P)),eval_prem(signe(P,positif)).
valeur_fils(c_composé_mtype_ajout_un,P,c_composé_mtype_ajout_un_neg):-
            eval_prem(problème(P)),eval_prem(signe(P,négatif)).

/* le père */
père(c_composé_mtype_ajout_un,c_composé_mtype_ajout).

infos(c_composé_mtype_ajout_un,P):-write('problème composé : deux parties,
opérateurs de même type, type : AJOUT, à trouver : un des opérateurs').

/*************** LA CLASSE COMPOSE_MTYPE_AJOUT_UN_POS ************************/

statut(c_composé_mtype_ajout_un_pos,opérationnelle).
type_résultat(c_composé_mtype_ajout_un_pos,opérateur).
type_données(c_composé_mtype_ajout_un_pos,[opérateur,opérateur]).

/* le père */
père(c_composé_mtype_ajout_un_pos,c_composé_mtype_ajout_un).

infos(c_composé_mtype_ajout_un_pos,P):-write('problème composé : deux parties,
opérateurs de même type, type : AJOUT, à trouver : un des opérateurs, signe
positif ou nul').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_mtype_ajout_un_pos):-utiliser([sol10]).

/*************** LA CLASSE COMPOSE_MTYPE_AJOUT_UN_NEG ************************/

statut(c_composé_mtype_ajout_un_neg,opérationnelle).
type_résultat(c_composé_mtype_ajout_un_neg,opérateur).
type_données(c_composé_mtype_ajout_un_neg,[opérateur,opérateur]).

/* le père */
père(c_composé_mtype_ajout_un_neg,c_composé_mtype_ajout_un_pos).

infos(c_composé_mtype_ajout_un_neg,P):-write('problème composé : deux parties,
opérateurs de même type, type : AJOUT, à trouver : un des opérateurs, signe
négatif').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_mtype_ajout_un_neg):-utiliser([sol11]).

/*************** LA CLASSE COMPOSE_MTYPE_RET   ************************/

statut(c_composé_mtype_ret,non_opérationnelle).

test_fils(c_composé_mtype_ret,P,à_trouver(P,T)):-eval_prem(problème(P)).

valeur_fils(c_composé_mtype_ret,P,c_composé_mtype_ret_double):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,double_opérateur)).
valeur_fils(c_composé_mtype_ret,P,c_composé_mtype_ret_un):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,un_opérateur)).

/* le père */
père(c_composé_mtype_ret,c_composé_mtype).

infos(c_composé_mtype_ret,P):-write('problème composé : deux parties, opérateurs
de même type, type : RET').
```

```
/*************** LA CLASSE COMPOSE_MTYPE_RET_DOUBLE   ************************/

statut(c_composé_mtype_ret_double,opérationnelle).
type_résultat(c_composé_mtype_ret_double,opérateur).
type_données(c_composé_mtype_ret_double,[opérateur,opérateur]).

/* le père */
père(c_composé_mtype_ret_double,c_composé_mtype_ret).

infos(c_composé_mtype_ret_double,P):-write('problème composé : deux parties,
opérateurs de même type, type : RET, à trouver : l''opérateur résultant').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_mtype_ret_double):-utiliser([sol12]).

/*************** LA CLASSE COMPOSE_MTYPE_RET_UN   ************************/

statut(c_composé_mtype_ret_un,non_opérationnelle).

test_fils(c_composé_mtype_ret_un,P,signe(P,T)):-eval_prem(problème(P)).

valeur_fils(c_composé_mtype_ret_un,P,c_composé_mtype_ret_un_pos):-
            eval_prem(problème(P)),eval_prem(signe(P,positif)).
valeur_fils(c_composé_mtype_ret_un,P,c_composé_mtype_ret_un_neg):-
            eval_prem(problème(P)),eval_prem(signe(P,négatif)).

/* le père */
père(c_composé_mtype_ret_un,c_composé_mtype_ret).

infos(c_composé_mtype_ret_un,P):-write('problème composé : deux parties,
opérateurs de même type, type : RET, à trouver : un des opérateurs').

/*************** LA CLASSE COMPOSE_MTYPE_RET_UN_POS   ************************/

statut(c_composé_mtype_ret_un_pos,opérationnelle).
type_résultat(c_composé_mtype_un_pos,opérateur).
type_données(c_composé_mtype_ret_un_pos,[opérateur,opérateur]).

/* le père */
père(c_composé_mtype_ret_un_pos,c_composé_mtype_ret_un).

infos(c_composé_mtype_ret_un_pos,P):-write('problème composé : deux parties,
opérateurs de même type, type : RET, à trouver : un des opérateurs, signe
positif').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_mtype_ret_un_pos):-utiliser([sol13]).

/*************** LA CLASSE COMPOSE_MTYPE_RET_UN_NEG   ************************/

statut(c_composé_mtype_ret_un_neg,opérationnelle).
type_résultat(c_composé_mtype_un_neg,opérateur).
type_données(c_composé_mtype_ret_un_neg,[opérateur,opérateur]).

/* le père */
père(c_composé_mtype_ret_un_neg,c_composé_mtype_ret_un).

infos(c_composé_mtype_ret_un_neg,P):-write('problème composé : deux parties,
opérateurs de même type, type : RET, à trouver : un des opérateurs, signe
négatif').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_mtype_ret_un_neg):-utiliser([sol14]).
```

```
/*************** LA CLASSE COMPOSE_TYPEDIFF   ************************/

statut(c_composé_typediff,non_opérationnelle).

test_fils(c_composé_typediff,P,à_trouver(P,T)):-eval_prem(problème(P)).

valeur_fils(c_composé_typediff,P,c_composé_typediff_double):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,double_opérateur)).
valeur_fils(c_composé_typediff,P,c_composé_typediff_un):-
            eval_prem(problème(P)),eval_prem(à_trouver(P,un_opérateur)).

/* le père */
père(c_composé_typediff,c_composé).

infos(c_composé_typediff,P):-write('problème composé : deux parties, opérateurs
de types différents').

/*************** LA CLASSE COMPOSE_TYPEDIFF_DOUBLE  ************************/

statut(c_composé_typediff_double,non_opérationnelle).

test_fils(c_composé_typediff_double,P,signe(P,S)):-eval_prem(problème(P)).

valeur_fils(c_composé_typediff_double,P,c_composé_typediff_double_pos):-
            eval_prem(problème(P)),eval_prem(signe(P,positif)).
valeur_fils(c_composé_typediff_double,P,c_composé_typediff_double_neg):-
            eval_prem(problème(P)),eval_prem(signe(P,négatif)).

/* le père */
père(c_composé_typediff_double,c_composé_typediff).

infos(c_composé_typediff_double,P):-write('problème composé : deux parties,
opérateurs de types différents, à_trouver : l''opérateur résultant').

/*************** LA CLASSE COMPOSE_TYPEDIFF_DOUBLE_POS ************************/

statut(c_composé_typediff_double_pos,opérationnelle).
type_résultat(c_composé_typediff_double_pos,opérateur).
type_données(c_composé_typediff_double_pos,[opérateur,opérateur]).

/* le père */
père(c_composé_typediff_double_pos,c_composé_typediff_double).

infos(c_composé_typediff_double_pos,P):-write('problème composé : deux parties,
opérateurs de types différents, à_trouver : l''opérateur résultant, signe
positif').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_typediff_double_pos):-utiliser([sol15]).

/*************** LA CLASSE COMPOSE_TYPEDIFF_DOUBLE_NEG ************************/

statut(c_composé_typediff_double_neg,opérationnelle).
type_résultat(c_composé_typediff_double_neg,opérateur).
type_données(c_composé_typediff_double_neg,[opérateur,opérateur]).

/* le père */
père(c_composé_typediff_double_neg,c_composé_typediff_double).

infos(c_composé_typediff_double_neg,P):-write('problème composé : deux parties,
opérateurs de types différents, à_trouver : l''opérateur résultant, signe
négatif').
```

```
/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_typediff_double_neg):-utiliser([sol16]).

/*************** LA CLASSE COMPOSE_TYPEDIFF_UN  ************************/

statut(c_composé_typediff_un,non_opérationnelle).

test_fils(c_composé_typediff_un,P,manquant(P,O)):-eval_prem(problème(P)).

valeur_fils(c_composé_typediff_un,P,c_composé_typediff_un_ajout):-
            eval_prem(problème(P)),eval_prem(manquant(P,ajout)).
valeur_fils(c_composé_typediff_un,P,c_composé_typediff_un_ret):-
            eval_prem(problème(P)),eval_prem(manquant(P,ret)).

attribut(c_composé_typediff_un,P,manquant(P,O),manquant,[m1,m2],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_composé_typediff_un,c_composé_typediff).

infos(c_composé_typediff_un,P):-write('problème composé : deux parties,
opérateurs de types différents, à_trouver :  un des opérateurs').

/*************** LA CLASSE COMPOSE_TYPEDIFF_UN_AJOUT   ************/

statut(c_composé_typediff_un_ajout,opérationnelle).
type_résultat(c_composé_typediff_un_ajout,opérateur).
type_données(c_composé_typediff_un_ajout,[opérateur,opérateur]).

/* le père */
père(c_composé_typediff_un_ajout,c_composé_typediff_un).

infos(c_composé_typediff_un_ajout,P):-write('problème composé : deux parties,
opérateurs de types différents, à_trouver :  un des opérateurs, opérateur
manquant : AJOUT').

/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_typediff_un_ajout):-utiliser([sol17]).

/*************** LA CLASSE COMPOSE_TYPEDIFF_UN_RET   ************/

statut(c_composé_typediff_un_ret,opérationnelle).
type_résultat(c_composé_typediff_un_ret,opérateur).
type_données(c_composé_typediff_un_ret,[opérateur,opérateur]).

/* le père */
père(c_composé_typediff_un_ret,c_composé_typediff_un).

infos(c_composé_typediff_un_ret,P):-write('problème composé : deux parties,
opérateurs de types différents, à_trouver :  un des opérateurs, opérateur
manquant : RET').
/* la méthode de résolution associée à la classe */
employer_résolution(c_composé_typediff_un_ret):-utiliser([sol18]).

/*************************************************************/

/* aller chercher les attributs chez le père */
attribut(Classe,P,Att,Pred,Liste,Def):-
      père(Classe,Père),attribut(Père,P,Att,Pred,Liste,Def).
```

D3/ Quelques règles de reformulation

Nous présentons ici quelques règles de reformulation. Il y a pour le domaine des problèmes additifs 29 règles de reformulation. Nous ne donnons pas les 25 règles utilisées pour la décomposition de problèmes (chapitre 9).

```
/* LA NATURE DU PROBLEME : SIMPLE OU COMPOSE
*/

règle(nature_simple1):-
  si(     [problème(P),
           action(P,A),
           est_un(A,partie)]),
  alors([ajouter(nature(P,simple))]).

règle(nature_simple2):-
  si(     [problème(P),
           est_un(Pi,ensemble_de_personnes),
           éléments(Pi,[P1,P2]),
           est_un(P1,personne),
           est_un(P2,personne),]),
  alors([ajouter(nature(P,simple))]).

règle(nature_composé):-
  si(     [problème(P),
           action(P,A),
           est_un(A,succession_de_parties)]),
  alors([ajouter(nature(P,composé))]).

/* LA PERSONNE INTERESSEE */

règle(i2):-
  si(     [problème(P),
           action(P,A),
           participants(A,P1,P2),
           à_calculer(P,T),
           possède(Pi,Ti,Ei),
           est_un(Ei,ensemble),
           tout_élément(Ei,Xi),
           est_un(Xi,bille),
           taille(Ei,T),
          eval_prolog(member_s(Pi,[P1,P2]))]),
  alors([ajouter(intéressé(P,Pi))]).

/* LE TYPE DE CE QU'IL FAUT CALCULER :
RESULTAT, OPERANDE, OPERATEUR, UN_OPERATEUR,
DOUBLE_OPERATEUR */

règle(at1):-
  si(     [problème(P),
           à_calculer(P,T),
           possède(Pi,I,Ei),
           est_un(Ei,ensemble),
           tout_élément(Ei,Xi),
           est_un(Xi,bille),
           taille(Ei,T),
          est_un(Pi,ensemble_de_personnes),
           éléments(Pi,[P1,P2]),
           est_un(P1,personne),
           possède(P1,I,E1),
           est_un(E1,ensemble),
           tout_élément(E1,X1),
           est_un(X1,bille),
           taille(E1,A),
           est_un(P2,personne),
           possède(P2,I,E2),
           est_un(E2,ensemble),
           tout_élément(E2,X2),
           est_un(X2,bille),
           taille(E2,B),
           eval_prolog(integer(A)),
           eval_prolog(integer(B))]),
  alors([ajouter(à_trouver(P,résultat)),
           ajouter(donnée1(P,A)),
           ajouter(donnée2(P,B))]).

règle(at4):-
  si(     [problème(P),
           à_calculer(P,T),
           action(P,Ac),
           est_un(Ac,partie),
           partie(Ac,T1,T2),
           intéressé(P,Pi),
           possède(Pi,T1,E1),
           est_un(E1,ensemble),
           tout_élément(E1,X1),
           est_un(X1,bille),
           taille(E1,T),
           possède(Pi,T2,E2),
           est_un(E2,ensemble),
           tout_élément(E2,X2),
           est_un(X2,bille),
           taille(E2,A),
           eval_prolog(var_opérateur(P,B)),
           eval_prolog(integer(A)),
           eval_prolog(integer(B))]),
  alors([ajouter(à_trouver(P,opérande)),
           ajouter(donnée1(P,A)),
           ajouter(donnée2(P,B))]).

règle(at6):-
  si(     [problème(P),
           à_calculer(P,C),
           action(P,Ac),
          est_un(Ac,succession_de_parties),
           nombre_de_parties(Ac,2),
           type_opérateurs(P,même_type),
           participants(Ac,P1,P2),
           intéressé(P,Pi),
       succession(Ac,[[A1,T1,T2],[A2,T2,T3]]),
           changement(Pi,Ac,Ec),
           est_un(Ec,ensemble),
           tout_élément(Ec,Xc),
           est_un(Xc,bille),
           taille(Ec,C),
           changement(Pi,A1,Ea),
           est_un(Ea,ensemble),
           tout_élément(Ea,Xa),
           est_un(Xa,bille),
           taille(Ea,A),
           changement(Pi,A2,Eb),
           est_un(Eb,ensemble),
           tout_élément(Eb,Xb),
           est_un(Xb,bille),
           taille(Eb,B),
           eval_prolog(integer(A)),
           eval_prolog(integer(B))]),
  alors([ajouter(à_trouver(P,double_opérateur
)),
           ajouter(donnée1(P,A)),
           ajouter(donnée2(P,B))]).

/* SIGNE POSITIF OU NEGATIF */
règle(s1):-
  si(     [problème(P),
           donnée1(P,A),
           donnée2(P,B),
           eval_prolog(=<(B,A))]),
  alors([ajouter(signe(P,positif))]).

règle(s2):-
  si(     [problème(P),
           donnée1(P,A),
           donnée2(P,B),
           eval_prolog(<(A,B))]),
  alors([ajouter(signe(P,négatif))]).
```

```
/* TROUVER L'OPERATEUR : ADD, AJOUT OU RET
*/

règle(op_add):-
  si(     [problème(P),
           est_un(P1,personne),
           possède(P1,I,E1),
           est_un(E1,ensemble),
           tout_élément(E1,X1),
           est_un(X1,bille),
           est_un(P2,personne),
           possède(P2,I,E2),
           est_un(E2,ensemble),
           tout_élément(E2,X2),
           est_un(X2,bille),
           est_un(P3,ensemble_de_personnes),
           éléments(P3,[P1,P2]),
           possède(P3,I,E3),
           est_un(E3,ensemble),
           tout_élément(E3,X3),
           est_un(X3,bille),
           taille(E1,N1),
           taille(E2,N2),
           taille(E3,N3),
        eval_prolog(deux_connus(N1,N2,N3))]),
  alors([ajouter(opérateur(P,add))]).

règle(op_ajout1):-
  si(     [problème(P),
           action(P,A),
           est_un(A,partie),
           participants(A,P1,P2),
           intéressé(P,Pi),
           gagné(Pi,A,En),
           est_un(En,ensemble),
           tout_élément(En,Xn),
           est_un(Xn,bille),
           taille(En,N)]),
  alors([ajouter(opérateur(P,ajout)),
           ajouter(ajout(P,N))]).

règle(op_ret2):-
  si(     [problème(P),
           action(P,A),
           est_un(A,partie),
           participants(A,P1,P2),
           intéressé(P,Pi),
           partie(A,T1,T2),
           possède(Pi,T1,E1),
           est_un(E1,ensemble),
           tout_élément(E1,X1),
           est_un(X1,bille),
           taille(E1,N1),
           possède(Pi,T2,E2),
           est_un(E2,ensemble),
           tout_élément(E2,X2),
           est_un(X2,bille),
           taille(E2,N2),
           eval_prolog(>(N1,N2))]),
  alors([ajouter(opérateur(P,ret))]).

/* CONSTRUIRE LA LISTE DES OPERATUERS QUAND
IL Y A PLUSIEURS PARTIES */

règle(init,op):-
  si(     [problème(P),
           action(P,A),
           est_un(A,succession_de_parties),
           nombre_de_parties(A,2)]),
  alors([eval_prolog(retirer_tous(opérateurs(
P,_))),ajouter(opérateurs(P,[]))]).

règle(ajout,op):-
  si(     [problème(P),
           action(P,A),
           est_un(A,succession_de_parties),
           nombre_de_parties(A,2),
           participants(A,P1,P2),
           intéressé(P,Pi),
         succession(A,[[A1,T1,T2],[A2,T2,T3]]),
           gagné(Pi,Ai,En),
           est_un(En,ensemble),
           tout_élément(En,Xn),
           est_un(Xn,bille),
           taille(En,N),
         eval_prolog(member_s(Ai,[A,A1,A2]))]),
  alors([modifier(opérateurs(P,L),opérateurs(
P,[ajout|L]))]).

règle(ret,op):-
  si(     [problème(P),
           action(P,A),
           est_un(A,succession_de_parties),
           nombre_de_parties(A,2),
           participants(A,P1,P2),
           intéressé(P,Pi),
         succession(A,[[A1,T1,T2],[A2,T2,T3]]),
           perdu(Pi,Ai,En),
           est_un(En,ensemble),
           tout_élément(En,Xn),
           est_un(Xn,bille),
           taille(En,N),
         eval_prolog(member_s(Ai,[A,A1,A2]))]),
  alors([modifier(opérateurs(P,L),opérateurs(
P,[ret|L]))]).

/* TYPE DES OPERATEURS */

règle(top1):-
  si(     [problème(P),
           opérateurs(P,[X,X])]),
  alors([ajouter(type_opérateurs(P,même_type)
)]).

règle(top2):-
  si(     [problème(P),
           opérateurs(P,[X,Y]),
           eval_prolog(X\=Y)]),
  alors([ajouter(type_opérateurs(P,types_diff
érents))]).

/* OPERATEUR MANQUANT */

règle(m1):-
  si(     [problème(P),
           action(P,Ac),
           est_un(Ac,succession_de_parties),
           nombre_de_parties(A,2),
           participants(Ac,P1,P2),
           intéressé(P,Pi),
           gagné(Pi,Ac,_)]),
  alors([ajouter(manquant(P,ajout))]).

règle(m2):-
  si(     [problème(P),
           action(P,Ac),
           est_un(Ac,succession_de_parties),
           nombre_de_parties(A,2),
           participants(Ac,P1,P2),
           intéressé(P,Pi),
           perdu(Pi,Ac,_)]),
  alors([ajouter(manquant(P,ret))]).
```

Annexe E
Connaissances données à SYRCLAD-tarot

E1/ Les règles de Bateleur

Bateleur utilise 14 règles pour choisir un plan de jeu. Elles sont présentées en annexe D de la thèse de J.-M. Nigro, page 224. Nous en donnons ici quelques-unes.

Si Hauteur d'atout >= 2
Longueur d'atout >= 2
Avec 1 d'atout
Alors le plan de jeu est :
Affranchir la longue
Jouer les perdantes
Epuiser les atouts adverses
Fermer le jeu
Couper une couleur
Amener le petit au bout

Si Hauteur d'atout < 2
Longueur d'atout < 2
Nombre de perdantes < 6
Avec 1 d'atout
Alors le plan de jeu est :
Sauver le petit dès que possible
Affranchir la longue
Jouer la longue pour faire couper
Jouer les perdantes
Fermer le jeu
Couper une couleur

Si Hauteur d'atout >= 2
Longueur d'atout < 2
Nombre de perdantes >= 6
Sans 1 d'atout
Alors le plan de jeu est :
Affranchir la longue
Jouer la longue pour faire couper
Faire couper
Jouer les perdantes
Fermer le jeu
Couper une couleur

E2/ Graphe de classification

```
racine_arbre(c_problème).

/****************** LA CLASSE PROBLEME ****************************/

statut(c_problème,non_opérationnelle).

test_fils(c_problème,P,hauteur_atout(P,H)):-eval_prem(problème(P)).

valeur_fils(c_problème,P,c_haut):-
      eval_prem(problème(P)),eval_prem(hauteur_atout(P,haut)).
valeur_fils(c_problème,P,c_bas):-
      eval_prem(problème(P)),eval_prem(hauteur_atout(P,bas)).

attribut(c_problème,P,hauteur_atout(P,H),hauteur_atout,[hauteur1,hauteur2],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,nb_atouts_manquants(P,NM),nb_atouts_manquants,[manquants],
[]):- eval_prem(problème(P)).
attribut(c_problème,P,longueur_atout(P,L),longueur_atout,[longueur1,longueur2],[
]):-  eval_prem(problème(P)).
attribut(c_problème,P,perdantes(P,N),perdantes,[perdantes1,perdantes2],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,nb_perdantes_carreau(P,Nca),nb_perdantes_carreau,[perdante
s_carreau],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,nb_perdantes_cœur(P,Nco),nb_perdantes_cœur,[perdantes_cœur
],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,nb_perdantes_trèfle(P,Nt),nb_perdantes_trèfle,[perdantes_t
rèfle],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,nb_perdantes_pique(P,Np),nb_perdantes_pique,[perdantes_piq
ue],[]):-
      eval_prem(problème(P)).
attribut(c_problème,P,petit(P,B),petit,[petit1,petit2],[]):-
      eval_prem(problème(P)).

/*************** LA CLASSE HAUT ********************************/

statut(c_haut,non_opérationnelle).

test_fils(c_haut,P,longueur_atout(P,H)):-eval_prem(problème(P)).

valeur_fils(c_haut,P,c_haut_long):-
      eval_prem(problème(P)),eval_prem(longueur_atout(P,longue)).
valeur_fils(c_haut,P,c_haut_court):-
      eval_prem(problème(P)),eval_prem(longueur_atout(P,courte)).

/* le père */
père(c_haut,c_problème).

infos(c_haut,P):-write('atout haut').

/*************** LA CLASSE HAUT_LONG ********************************/

statut(c_haut_long,non_opérationnelle).

test_fils(c_haut_long,P,petit(P,B)):-eval_prem(problème(P)).
```

```
valeur_fils(c_haut_long,P,c_haut_long_avec):-
      eval_prem(problème(P)),eval_prem(petit(P,avec)).
valeur_fils(c_haut_long,P,c_haut_long_sans):-
      eval_prem(problème(P)),eval_prem(petit(P,sans)).

/* le père */
père(c_haut_long,c_haut).

infos(c_haut_long,P):-write('atout haut, atout long').

/*************** LA CLASSE HAUT_LONG_AVEC *********************************/

statut(c_haut_long_avec,opérationnelle).

/* le père */
père(c_haut_long_avec,c_haut_long).

infos(c_haut_long_avec,P):-write('atout haut, atout long, avec le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_haut_long_avec):-ajouter_bf(plan_de_jeu(P,pl1c1)).

/*************** LA CLASSE HAUT_LONG_SANS *********************************/

statut(c_haut_long_sans,opérationnelle).

/* le père */
père(c_haut_long_sans,c_haut_long).

infos(c_haut_long_sans,P):-write('atout haut, atout long, sans le petit').


/* la méthode de résolution associée à la classe */
employer_résolution(c_haut_long_sans):-ajouter_bf(plan_de_jeu(P,pl1c2)).

/*************** LA CLASSE HAUT_COURT *********************************/

statut(c_haut_court,non_opérationnelle).

test_fils(c_haut_court,P,perdantes(P,N)):-eval_prem(problème(P)).

valeur_fils(c_haut_court,P,c_haut_court_bcp):-
      eval_prem(problème(P)),eval_prem(perdantes(P,beaucoup)).
valeur_fils(c_haut_court,P,c_haut_court_peu):-
      eval_prem(problème(P)),eval_prem(perdantes(P,peu)).

/* le père */
père(c_haut_court,c_haut).

infos(c_haut_court,P):-write('atout haut, atout court').

/*************** LA CLASSE HAUT_COURT_BCP *********************************/

statut(c_haut_court_bcp,non_opérationnelle).

test_fils(c_haut_court_bcp,P,petit(P,N)):-eval_prem(problème(P)).

valeur_fils(c_haut_court_bcp,P,c_haut_court_bcp_avec):-
      eval_prem(problème(P)),eval_prem(petit(P,avec)).
valeur_fils(c_haut_court_bcp,P,c_haut_court_bcp_sans):-
      eval_prem(problème(P)),eval_prem(petit(P,sans)).
```

```
/* le père */
père(c_haut_court_bcp,c_haut_court).

infos(c_haut_court_bcp,P):-write('atout haut, atout court, beaucoup de
perdantes').

/*************** LA CLASSE HAUT_COURT_BCP_AVEC ***************************/

statut(c_haut_court_bcp_avec,opérationnelle).

/* le père */
père(c_haut_court_bcp_avec,c_haut_court_bcp).

infos(c_haut_court_bcp_avec,P):-write('atout haut, atout court, beaucoup de
perdantes, avec le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_haut_court_bcp_avec):-ajouter_bf(plan_de_jeu(P,pl5c1)).

/*************** LA CLASSE HAUT_COURT_BCP_SANS ****************************/

statut(c_haut_court_bcp_sans,opérationnelle).

/* le père */
père(c_haut_court_bcp_sans,c_haut_court_bcp).

infos(c_haut_court_bcp_sans,P):-write('atout haut, atout court, beaucoup de
perdantes, sans le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_haut_court_bcp_sans):-ajouter_bf(plan_de_jeu(P,pl5c2)).

/*************** LA CLASSE HAUT_COURT_PEU *********************************/

statut(c_haut_court_peu,non_opérationnelle).

test_fils(c_haut_court_peu,P,petit(P,N)):-eval_prem(problème(P)).

valeur_fils(c_haut_court_peu,P,c_haut_court_peu_avec):-
      eval_prem(problème(P)),eval_prem(petit(P,avec)).
valeur_fils(c_haut_court_peu,P,c_haut_court_peu_sans):-
      eval_prem(problème(P)),eval_prem(petit(P,sans)).

/* le père */
père(c_haut_court_peu,c_haut_court).

infos(c_haut_court_peu,P):-write('atout haut, atout court, peu de perdantes').

/*************** LA CLASSE HAUT_COURT_PEU_AVEC ****************************/

statut(c_haut_court_peu_avec,opérationnelle).

/* le père */
père(c_haut_court_peu_avec,c_haut_court_peu).

infos(c_haut_court_peu_avec,P):-write('atout haut, atout court, peu de
perdantes, avec le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_haut_court_peu_avec):-ajouter_bf(plan_de_jeu(P,pl3c1)).
```

```
/*************** LA CLASSE HAUT_COURT_PEU_SANS ******************************/

statut(c_haut_court_peu_sans,opérationnelle).

/* le père */
père(c_haut_court_peu_sans,c_haut_court_peu).

infos(c_haut_court_peu_sans,P):-write('atout haut, atout court, peu de
perdantes, sans le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_haut_court_peu_sans):-ajouter_bf(plan_de_jeu(P,pl3c2)).

/*************** LA CLASSE BAS ********************************/

statut(c_bas,non_opérationnelle).

test_fils(c_bas,P,longueur_atout(P,H)):-eval_prem(problème(P)).

valeur_fils(c_bas,P,c_bas_long):-
      eval_prem(problème(P)),eval_prem(longueur_atout(P,longue)).
valeur_fils(c_bas,P,c_bas_court):-
      eval_prem(problème(P)),eval_prem(longueur_atout(P,courte)).

/* le père */
père(c_bas,c_problème).

infos(c_bas,P):-write('atout bas').

/*************** LA CLASSE BAS_LONG ********************************/

statut(c_bas_long,non_opérationnelle).

test_fils(c_bas_long,P,perdantes(P,N)):-eval_prem(problème(P)).

valeur_fils(c_bas_long,P,c_bas_long_bcp):-
      eval_prem(problème(P)),eval_prem(perdantes(P,beaucoup)).
valeur_fils(c_bas_long,P,c_bas_long_peu):-
      eval_prem(problème(P)),eval_prem(perdantes(P,peu)).

/* le père */
père(c_bas_long,c_bas).

infos(c_bas_long,P):-write('atout bas, atout long').

/*************** LA CLASSE BAS_LONG_BCP ********************************/

statut(c_bas_long_bcp,non_opérationnelle).

test_fils(c_bas_long_bcp,P,petit(P,N)):-eval_prem(problème(P)).

valeur_fils(c_bas_long_bcp,P,c_bas_long_bcp_avec):-
      eval_prem(problème(P)),eval_prem(petit(P,avec)).
valeur_fils(c_bas_long_bcp,P,c_bas_long_bcp_sans):-
      eval_prem(problème(P)),eval_prem(petit(P,sans)).

/* le père */
père(c_bas_long_bcp,c_bas_long).

infos(c_bas_long_bcp,P):-write('atout bas, atout long, beaucoup de perdantes').
```

```
/*************** LA CLASSE BAS_LONG_BCP_AVEC ******************************/

statut(c_bas_long_bcp_avec,opérationnelle).

/* le père */
père(c_bas_long_bcp_avec,c_bas_long_bcp).

infos(c_bas_long_bcp_avec,P):-write('atout bas, atout long, beaucoup de
perdantes, avec le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_long_bcp_avec):-ajouter_bf(plan_de_jeu(P,pl6c1)).

/*************** LA CLASSE BAS_LONG_BCP_SANS ******************************/

statut(c_bas_long_bcp_sans,opérationnelle).

/* le père */
père(c_bas_long_bcp_sans,c_bas_long_bcp).

infos(c_bas_long_bcp_sans,P):-write('atout bas, atout long, beaucoup de
perdantes, sans le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_long_bcp_sans):-ajouter_bf(plan_de_jeu(P,pl6c2)).

/*************** LA CLASSE BAS_LONG_PEU ***********************************/

statut(c_bas_long_peu,non_opérationnelle).

test_fils(c_bas_long_peu,P,petit(P,N)):-eval_prem(problème(P)).

valeur_fils(c_bas_long_peu,P,c_bas_long_peu_avec):-
      eval_prem(problème(P)),eval_prem(petit(P,avec)).
valeur_fils(c_bas_long_peu,P,c_bas_long_peu_sans):-
      eval_prem(problème(P)),eval_prem(petit(P,sans)).

/* le père */
père(c_bas_long_peu,c_bas_long).

infos(c_bas_long_peu,P):-write('atout bas, atout long, peu de perdantes').

/*************** LA CLASSE BAS_LONG_PEU_AVEC ******************************/

statut(c_bas_long_peu_avec,opérationnelle).

/* le père */
père(c_bas_long_peu_avec,c_bas_long_peu).

infos(c_bas_long_peu_avec,P):-write('atout bas, atout long, peu de perdantes,
avec le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_long_peu_avec):-ajouter_bf(plan_de_jeu(P,pl2c1)).

/*************** LA CLASSE BAS_LONG_PEU_SANS ******************************/

statut(c_bas_long_peu_sans,opérationnelle).

/* le père */
père(c_bas_long_peu_sans,c_bas_long_peu).
```

```
infos(c_bas_long_peu_sans,P):-write('atout bas, atout long, peu de perdantes,
sans le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_long_peu_sans):-ajouter_bf(plan_de_jeu(P,pl2c2)).

/*************** LA CLASSE BAS_COURT *********************************/

statut(c_bas_court,non_opérationnelle).

test_fils(c_bas_court,P,perdantes(P,N)):-eval_prem(problème(P)).

valeur_fils(c_bas_court,P,c_bas_court_bcp):-
      eval_prem(problème(P)),eval_prem(perdantes(P,beaucoup)).
valeur_fils(c_bas_court,P,c_bas_court_peu):-
      eval_prem(problème(P)),eval_prem(perdantes(P,peu)).

/* le père */
père(c_bas_court,c_bas).

infos(c_bas_court,P):-write('atout bas, atout court').

/*************** LA CLASSE BAS_COURT_BCP *****************************/

statut(c_bas_court_bcp,non_opérationnelle).

test_fils(c_bas_court_bcp,P,petit(P,N)):-eval_prem(problème(P)).

valeur_fils(c_bas_court_bcp,P,c_bas_court_bcp_avec):-
      eval_prem(problème(P)),eval_prem(petit(P,avec)).
valeur_fils(c_bas_court_bcp,P,c_bas_court_bcp_sans):-
      eval_prem(problème(P)),eval_prem(petit(P,sans)).

/* le père */
père(c_bas_court_bcp,c_bas_court).

infos(c_bas_court_bcp,P):-write('atout  bas,  atout  court,  beaucoup  de
perdantes').

/*************** LA CLASSE BAS_COURT_BCP_AVEC *************************/

statut(c_bas_court_bcp_avec,opérationnelle).

/* le père */
père(c_bas_court_bcp_avec,c_bas_court_bcp).

infos(c_bas_court_bcp_avec,P):-write('atout  bas,  atout  court,  beaucoup  de
perdantes, avec le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_court_bcp_avec):-ajouter_bf(plan_de_jeu(P,pl7c1)).

/*************** LA CLASSE BAS_COURT_BCP_SANS *************************/

statut(c_bas_court_bcp_sans,opérationnelle).

/* le père */
père(c_bas_court_bcp_sans,c_bas_court_bcp).

infos(c_bas_court_bcp_sans,P):-write('atout  bas,  atout  court,  beaucoup  de
perdantes, sans le petit').
```

```
/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_court_bcp_sans):-ajouter_bf(plan_de_jeu(P,pl7c2)).

/*************** LA CLASSE BAS_COURT_PEU ********************************/

statut(c_bas_court_peu,non_opérationnelle).

test_fils(c_bas_court_peu,P,petit(P,N)):-eval_prem(problème(P)).

valeur_fils(c_bas_court_peu,P,c_bas_court_peu_avec):-
      eval_prem(problème(P)),eval_prem(petit(P,avec)).
valeur_fils(c_bas_court_peu,P,c_bas_court_peu_sans):-
      eval_prem(problème(P)),eval_prem(petit(P,sans)).

/* le père */
père(c_bas_court_peu,c_bas_court).

infos(c_bas_court_peu,P):-write('atout bas, atout court, peu de perdantes').

/*************** LA CLASSE BAS_COURT_PEU_AVEC ********************************/

statut(c_bas_court_peu_avec,opérationnelle).

/* le père */
père(c_bas_court_peu_avec,c_bas_court_peu).

infos(c_bas_court_peu_avec,P):-write('atout bas, atout court, peu de perdantes,
avec le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_court_peu_avec):-ajouter_bf(plan_de_jeu(P,pl4c1)).

/*************** LA CLASSE BAS_COURT_PEU_SANS ********************************/

statut(c_bas_court_peu_sans,opérationnelle).

/* le père */
père(c_bas_court_peu_sans,c_bas_court_peu).

infos(c_bas_court_peu_sans,P):-write('atout bas, atout court, peu de perdantes,
sans le petit').

/* la méthode de résolution associée à la classe */
employer_résolution(c_bas_court_peu_sans):-ajouter_bf(plan_de_jeu(P,pl4c2)).

/*************************************************************/

/* aller chercher les attributs chez le père */
attribut(Classe,P,Att,Pred,Liste,Def):-
      père(Classe,Père),attribut(Père,P,Att,Pred,Liste,Def).
```

E3/ Quelques règles de reformulation

```
/************* le petit ******************/
règle(petit1):-
si(    [problème(P),
       joueur(P,J),
       eval_prolog(possède(P,J,[atout,1]))]),
alors([ajouter(petit(P,avec))]).

règle(petit2):-

/****** la longueur d'atout ***************/

règle(longueur1):-
si(    [problème(P),
       joueur(P,J),
       eval_prolog(nb_atout_moyen(P,Nm)),
       eval_prolog(+(Nm,5,Nr)),
       eval_prolog(nombre_atout(P,J,Nj)),
       eval_prolog(Nj<Nr)]),
alors([ajouter(longueur_atout(P,courte))]).

règle(longueur2):-

/***** la hauteur d'atout ****************/

règle(hauteur1):-
si(    [problème(P),
       joueur(P,J),
       nb_atouts_manquants(P,NM),
       eval_prolog(nb_maitres_à_tester_initial(P,NMI)),
       eval_prolog(/(NM,NMI,C)),
       eval_prolog(C=<0.4)]),
alors([ajouter(hauteur_atout(P,haut))]).

règle(hauteur2):-

règle(manquants):-
si(    [problème(P),
       joueur(P,J),
       eval_prolog(nb_maitres_à_tester_initial(P,NMi)),
       eval_prolog(nombre_atout(P,J,Na)),
       eval_prolog(min(NMi,Na,Min)),
       eval_prolog(liste_atout(La)),
       eval_prolog(premiers(La,Min,Lr)),
       eval_prolog(nb_manquantes(P,J,atout,Lr,N))]),
alors([ajouter(nb_atouts_manquants(P,N))]).

/**** Le nombre de perdantes ********************/

règle(perdantes1):-
si(    [problème(P),
       joueur(P,J),
       nb_perdantes_carreau(P,Nca),
       nb_perdantes_cœur(P,Nco),
       nb_perdantes_trèfle(P,Nt),
       nb_perdantes_pique(P,Np),
       eval_prolog(somme([Nca,Nco,Nt,Np],N)),
       eval_prolog(N>=6)]),
alors([ajouter(perdantes(P,beaucoup))]).

règle(perdantes2):-

règle(perdantes_carreau):-
si(    [problème(P),
       joueur(P,J),
       main(P,J,carreau,Lcoul_joueur),
       eval_prolog(si_écart(P,J,Lécart)),
       eval_prolog(select_coul(Lécart,carreau,Lcoul_écart)),
       eval_prolog(append(Lcoul_joueur,Lcoul_écart,Lcoul_amis)),
       eval_prolog(liste_hauteur(Lcoul)),
       eval_prolog(complémentaire(Lcoul,Lcoul_amis,Lcoul_ennemis)),
       eval_prolog(nb_perdantes(Lcoul_joueur,Lcoul_ennemis,N))]),
alors([ajouter(nb_perdantes_carreau(P,N))]).

règle(perdantes_cœur):-

règle(perdantes_trèfle):-

règle(perdantes_pique):-
```

Annexe F
Connaissances données à SYRCLAD-thermodynamique

F1/ Prédicats du langage de description et prédicats issus de la classification

Prédicats intervenant dans les modèles descriptifs

problème(Problème)	*objet*
on_demande(Liste de grandeurs)	*inconnue*
systèmes(Problème,Liste)	*objets*
instants(Problème,Liste)	*objets*
chaleur(Sys,I,F)	*objet*
composition(Sys)	*objet*
cp, cp_mol, cpt, cv, cv_mol, cvt	*objets*
débit_massique, débit_moles, débit_vol	*objets*
densité(Sys)	*objet*
déplacement_cloche	*objet*
diamètre(Gazo)	*objet*
durée(Ii,If)	*objet*
gamma(Sys)	*objet*
gradient(Grandeur,Sys,B,O)	*objet*
hauteur(Chem)	*objet*
hauteur_emergee, hauteur_immergee (Gazo,I)	*objets*
hauteur_sous_cloche (Gazo,I)	*objet*
longueur	*objet*
masse(S,I)	*objet*
masse_molaire(S,I),masse_volumique(S,I)	*objets*
max	*objet*
moles_passées	*objet*
n(S,I) (nombre de moles)	*objet*
nrj_interne(Sys,I,F)	*objet*
p_cloche(Gazo)	*objet*
p_ext	*objet*
poids(Gazo)	*objet*
pression(S,I)	*objet*
r_molaire(S,I)	*objet*
rapport	*objet*
section(Gazo)	*objet*
surpression(Sys,Atm,I)	*objet*
température(S,I)	*objet*
travail(S,I,F)	*objet*
travail_effectif	*objet*
travail_fourni_par	*objet*
variation	*objet*
vol_massique(S,I), vol_molaire(S,I)	*objets*
volume(S,I)	*objet*
adiabatique(Sys,I,F)	*propriété*
ambiance(S,A,I,F)	*relation*
compression(Sys,I,F)	*propriété*
conserve	*propriété*
débit_vol_unif	*propriété*
détendeur(D,E,S)	*relation*
diatomique, monoatomique(Sys)	*propriété*
fp, sq, gp, équi_fp, équi_fp_int, équi_gp, équi_gp_int,	*propriétés*
équi_sq, équi_sq_int	*propriétés*
équi_gazomètre	*relation*
trans_fp_fermé, trans_fp_int, trans_gp_fermé, trans_gp_int, trans_sq, trans_sq_fermé	*propriété*
trans_gazomètre,	*relation*
trans_circulateur	*relation*
trans_surface	*propriété*
état_circulateur	*relation*
état_surface	*propriété*
gazomètre(Gazo)	*propriété*
inv1, inv2, loi1, loi2	*propriétés*
isobare, isochore,isotherme	*propriétés*

ouverture_ambiance, ouverture_unique *relations*
régime *relation*
régime_moles *propriété*
régime_permanent *relation*
régime_vol *propriété*

48 objets, 10 relations, 27 propriétés

Prédicats issus de la classification

+ nombre de règles concluant sur cet attribut

A la racine

nb_systèmes(Problème,Entier) *1*
nb_instants(Problème,Entier) *1*
type_variables(Problème,int/ext) *2*

Dans les branches

conservation(Problème) *8*
gazo(Problème) *1*
surface(Problème) *2*
composé(Problème,gazomètre / surface / ambiance / mélange) *4*
cas_particulier(P,invariant/gamma/nrj) *4*
cas_particulier_nrj(P,w_inv / w_gam / q_it / du_ad) *4*

Attributs discriminants

+ nombre de valeurs possibles

nb_systèmes(Problème,Entier) *2*
nb_instants(Problème,Entier) *2*
type_variables(Problème,int/ext) *2*
conservation(Problème) *2*
composé(Problème,gazomètre/surface/ambiance/mélange)) *4*
cas_particulier(P,invariant/gamma/nrj) *4*
cas_particulier_nrj(P,w_inv/w_gam/q_it/du_ad) *4*
gazo(Problème) *booléen*

Attributs utilisés pour le plan

liste_inv, liste_gam, liste_w_inv, liste_w_gam

F2/ Graphe de classification

```
racine_arbre(c_problème).

/*************** LA CLASSE PROBLEME ***********************/

statut(c_problème,non_opérationnelle).

test_fils(c_problème,P,nb_systèmes(P,N)):-eval_prem(problème(P)).

valeur_fils(c_problème,P,c_1s):-
            eval_prem(problème(P)),eval_prem(nb_systèmes(P,1)).
valeur_fils(c_problème,P,c_2s):-
            eval_prem(problème(P)),eval_prem(nb_systèmes(P,2)).

attribut(c_problème,P,nb_systèmes(P,N),nb_systèmes,[nb_sys],[]):-
            eval_prem(problème(P)).
attribut(c_problème,P,nb_instants(P,N),nb_instants,[nb_ins]),[]:-
            eval_prem(problème(P)).
attribut(c_problème,P,type_variables(P,T),type_variables,[int,ext],[]):-
            eval_prem(problème(P)).

infos(c_problème,P).

/*************** LA CLASSE 1S ************************/

statut(c_1s,non_opérationnelle).

test_fils(c_1s,P,nb_instants(P,N)):-eval_prem(problème(P)).

valeur_fils(c_1s,P,c_1s_1i):-
            eval_prem(problème(P)),eval_prem(nb_instants(P,1)).
valeur_fils(c_1s,P,c_1s_2i):-
            eval_prem(problème(P)),eval_prem(nb_instants(P,2)).

/* le père */
père(c_1s,c_problème).

infos(c_1s,P):-write('un système').

/*************** LA CLASSE 1S1I ************************/

statut(c_1s_1i,opérationnelle).
nombre_systèmes(c_1s_1i,1).
nombre_instants(c_1s_1i,1).

test_fils(c_1s_1i,P,type_variables(P,T)):-eval_prem(problème(P)).

valeur_fils(c_1s_1i,P,c_1s_1i_int):-
            eval_prem(problème(P)),eval_prem(type_variables(P,int)).
valeur_fils(c_1s_1i,P,c_1s_1i_ext):-
            eval_prem(problème(P)),eval_prem(type_variables(P,ext)).

/* le père */
père(c_1s_1i,c_1s).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_1i):-résoudre([1]).
```

```
/*************** LA CLASSE 1S1I_int ************************/

statut(c_1s_1i_int,opérationnelle).

/* le père */
père(c_1s_1i_int,c_1s_1i).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_1i_int):-résoudre_int([1]).

/*************** LA CLASSE 1S1I_ext ************************/

statut(c_1s_1i_ext,opérationnelle).

/* le père */
père(c_1s_1i_ext,c_1s_1i).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_1i_ext):-résoudre_ext([1]).

/*************** LA CLASSE 1S2I ************************/

statut(c_1s_2i,non_opérationnelle).

test_fils(c_1s_2i,P,conservation(P,B)):-eval_prem(problème(P)).

valeur_fils(c_1s_2i,P,c_1s_2i_cons):-
            eval_prem(problème(P)),eval_prem(conservation(P,oui)).
valeur_fils(c_1s_2i,P,c_1s_2i_sans):-
            eval_prem(problème(P)),eval_prem(conservation(P,non)).

attribut(c_1s_2i,P,conservation(P,B),conservation,[isochore,isotherme,isobare,is
obare2,adiabatique,cons_inv1,cons_loi1],[sans]):-
            eval_prem(problème(P)).

/* le père */
père(c_1s_2i,c_1s).

infos(c_1s_2i,P):-write('un système, deux instants').

/*************** LA CLASSE 1S2I_cons ************************/

statut(c_1s_2i_cons,opérationnelle).
nombre_systèmes(c_1s_2i_cons,1).
nombre_instants(c_1s_2i_cons,2).

test_fils(c_1s_2i_cons,P,cas_part(P,B)):-eval_prem(problème(P)).

valeur_fils(c_1s_2i_cons,P,c_1s_2i_cons_inv):-
            eval_prem(problème(P)),eval_prem(cas_part(P,invariant)).
valeur_fils(c_1s_2i_cons,P,c_1s_2i_cons_gam):-
            eval_prem(problème(P)),eval_prem(cas_part(P,gamma)).
valeur_fils(c_1s_2i_cons,P,c_1s_2i_cons_nrj):-
            eval_prem(problème(P)),eval_prem(cas_part(P,nrj)).

attribut(c_1s_2i_cons,P,cas_part(P,B),cas_part,[cas_part_inv,cas_part_gam,cas_pa
rt_nrj1,cas_part_nrj2,presque_cas_part_inv,presque_cas_part_gam1,presque_cas_par
t_gam2],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_1s_2i_cons,c_1s_2i).
```

```
/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_cons):-résoudre([1,2,3]).

/*************** LA CLASSE 1S2I_cons_inv ************************/

statut(c_1s_2i_cons_inv,opérationnelle).
avec_plan(c_1s_2i_cons_inv).

/* le père */
père(c_1s_2i_cons_inv,c_1s_2i_cons).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_cons_inv):-
eval_prem(problème(P)),appliquer_plan1(P).

/*************** LA CLASSE 1S2I_cons_gam ************************/

statut(c_1s_2i_cons_gam,opérationnelle).
avec_plan(c_1s_2i_cons_gam).

/* le père */
père(c_1s_2i_cons_gam,c_1s_2i_cons).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_cons_gam):-
eval_prem(problème(P)),appliquer_plan2(P).

/*************** LA CLASSE 1S2I_cons_nrj ************************/

statut(c_1s_2i_cons_nrj,non_opérationnelle).

test_fils(c_1s_2i_cons_nrj,P,cas_part_nrj(P,B)):-eval_prem(problème(P)).

valeur_fils(c_1s_2i_cons_nrj,P,c_1s_2i_cons_nrj_W_inv):-
            eval_prem(problème(P)),eval_prem(cas_part_nrj(P,w_inv)).
valeur_fils(c_1s_2i_cons_nrj,P,c_1s_2i_cons_nrj_W_gam):-
            eval_prem(problème(P)),eval_prem(cas_part_nrj(P,w_gam)).
valeur_fils(c_1s_2i_cons_nrj,P,c_1s_2i_cons_nrj_Q_it):-
            eval_prem(problème(P)),eval_prem(cas_part_nrj(P,q_it)).
valeur_fils(c_1s_2i_cons_nrj,P,c_1s_2i_cons_nrj_DU_ad):-
            eval_prem(problème(P)),eval_prem(cas_part_nrj(P,du_ad)).

attribut(c_1s_2i_cons_nrj,P,cas_part_nrj(P,B),cas_part_nrj,[cas_part_nrj_w_inv,c
as_part_nrj_w_gam,cas_part_nrj_Q_it,cas_part_nrj_DU_ad,presque_cas_part_nrj_w_in
v,presque_cas_part_nrj_w_gam1,presque_cas_part_nrj_w_gam2],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_1s_2i_cons_nrj,c_1s_2i_cons).

/*************** LA CLASSE 1S2I_cons_nrj_W_inv ************************/

statut(c_1s_2i_cons_nrj_W_inv,opérationnelle).
avec_plan(c_1s_2i_cons_nrj_W_inv).

/* le père */
père(c_1s_2i_cons_nrj_W_inv,c_1s_2i_cons_nrj).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_cons_nrj_W_inv):-
eval_prem(problème(P)),appliquer_plan3(P).
```

```
/*************** LA CLASSE 1S2I_cons_nrj_W_gam ************************/

statut(c_1s_2i_cons_nrj_W_gam,opérationnelle).
avec_plan(c_1s_2i_cons_nrj_W_gam).

/* le père */
père(c_1s_2i_cons_nrj_W_gam,c_1s_2i_cons_nrj).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_cons_nrj_W_gam):-
eval_prem(problème(P)),appliquer_plan4(P).

/*************** LA CLASSE 1S2I_cons_nrj_Q_it ************************/

statut(c_1s_2i_cons_nrj_Q_it,opérationnelle).
avec_plan(c_1s_2i_cons_nrj_Q_it).

/* le père */
père(c_1s_2i_cons_nrj_Q_it,c_1s_2i_cons_nrj).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_cons_nrj_Q_it):-eval_prem(problème(P)),
            lire_bf(systèmes(P,[S])),
            lire_bf(instants(P,[I,F])),
            writeseq(['Le système ',S,' est isotherme entre les instants ',I,'
et ',F,', donc l''énergie interne est constante, et on a Q = -W. Calculons donc
le travail du système ',S,' entre les instants ',I,' et ',F]),!,
            retirer_bf(on_demande(P,[chaleur(S,I,F)])),
            retirer_bf(problème(P)),
            gensym(p_travail,Pt),
            ajouter_bf(problème(Pt)),
            ajouter_bf(systèmes(Pt,[S])),
            ajouter_bf(instants(Pt,[I,F])),
            ajouter_bf(on_demande(Pt,[travail(S,I,F)])),
            classer(Pt).

/*************** LA CLASSE 1S2I_cons_nrj_DU_ad ************************/

statut(c_1s_2i_cons_nrj_DU_ad,opérationnelle).
avec_plan(c_1s_2i_cons_nrj_DU_ad).

/* le père */
père(c_1s_2i_cons_nrj_DU_ad,c_1s_2i_cons_nrj).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_cons_nrj_DU_ad):-eval_prem(problème(P)),
            lire_bf(systèmes(P,[S])),
            lire_bf(instants(P,[I,F])),
            writeseq(['Le système ',S,' est adiabatique entre les instants ',I,'
et ',F,', donc la chaleur est nulle, et on a DU = W. Calculons donc le travail
du système ',S,' entre les instants ',I,' et ',F]),!,
            retirer_bf(on_demande(P,[variation(nrj_interne,S,I,F)])),
            retirer_bf(problème(P)),
            gensym(p_travail,Pt),
            ajouter_bf(problème(Pt)),
            ajouter_bf(systèmes(Pt,[S])),
            ajouter_bf(instants(Pt,[I,F])),
            ajouter_bf(on_demande(Pt,[travail(S,I,F)])),
            classer(Pt).
```

```
/*************** LA CLASSE 1S2I_sans ************************/

statut(c_1s_2i_sans,opérationnelle).
nombre_systèmes(c_1s_2i_sans,1).
nombre_instants(c_1s_2i_sans,2).

/* le père */
père(c_1s_2i_sans,c_1s_2i).

/* la méthode de résolution associée à la classe */
employer_résolution(c_1s_2i_sans):-résoudre([1,2]).

/*************** LA CLASSE 2S ************************/

statut(c_2s,non_opérationnelle).

test_fils(c_2s,P,nb_instants(P,N)):-eval_prem(problème(P)).

valeur_fils(c_2s,P,c_2s_1i):-
            eval_prem(problème(P)),eval_prem(nb_instants(P,1)).
valeur_fils(c_2s,P,c_2s_2i):-
            eval_prem(problème(P)),eval_prem(nb_instants(P,2)).

attribut(c_2s,P,gazo(P),gazo,[gazo],[]):-
            eval_prem(problème(P)).
attribut(c_2s,P,composé(P,T),composé,[comp1,comp2,comp3,comp4],[]):-
            eval_prem(problème(P)).
attribut(c_2s,P,surface(P),surface,[surf1,surf2],[]):-
            eval_prem(problème(P)).

/* le père */
père(c_2s,c_problème).

infos(c_2s,P):-write('deux système').

/*************** LA CLASSE 2S1I ************************/

statut(c_2s_1i,opérationnelle).
nombre_systèmes(c_2s_1i,2).
nombre_instants(c_2s_1i,1).

test_fils(c_2s_1i,P,gazo(P)):-eval_prem(problème(P)).

valeur_fils(c_2s_1i,P,c_2s_1i_gazo):-
            eval_prem(problème(P)),eval_prem(gazo(P)).

/* le père */
père(c_2s_1i,c_2s).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_1i):-résoudre([1]).

infos(c_2s_1i,P):-write('deux systèmes, un instant').

/*************** LA CLASSE 2S1I_gazo ************************/

statut(c_2s_1i_gazo,opérationnelle).

/* le père */
père(c_2s_1i_gazo,c_2s_1i).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_1i_gazo):-résoudre([1,5]).
```

```
/*************** LA CLASSE 2S2I ************************/

statut(c_2s_2i,non_opérationnelle).

test_fils(c_2s_2i,P,composé(P,T)):-eval_prem(problème(P)).

valeur_fils(c_2s_2i,P,c_2s_2i_gazo):-
            eval_prem(problème(P)),eval_prem(composé(P,gazomètre)).
valeur_fils(c_2s_2i,P,c_2s_2i_surf):-
            eval_prem(problème(P)),eval_prem(composé(P,surface)).
valeur_fils(c_2s_2i,P,c_2s_2i_ambiance):-
            eval_prem(problème(P)),eval_prem(composé(P,ambiance)).
valeur_fils(c_2s_2i,P,c_2s_2i_mélange):-
            eval_prem(problème(P)),eval_prem(composé(P,mélange)).

attribut(c_2s_2i,P,conservation(P,B),conservation,[isochore,isotherme,isobare,is
obare2,adiabatique,cons_inv1,cons_loi1],[sans]):-
            eval_prem(problème(P)).

/* le père */
père(c_2s_2i,c_2s).

infos(c_2s_2i,P):-write('deux systèmes, deux instants').

/*************** LA CLASSE 2S2I_gazo ************************/

statut(c_2s_2i_gazo,non_opérationnelle).

test_fils(c_2s_2i_gazo,P,conservation(P,B)):-eval_prem(problème(P)).

valeur_fils(c_2s_2i_gazo,P,c_2s_2i_gazo_cons):-
            eval_prem(problème(P)),eval_prem(conservation(P,oui)).
valeur_fils(c_2s_2i_gazo,P,c_2s_2i_gazo_sans):-
            eval_prem(problème(P)),eval_prem(conservation(P,non)).

/* le père */
père(c_2s_2i_gazo,c_2s_2i).

/*************** LA CLASSE 2S2I_gazo_cons ************************/

statut(c_2s_2i_gazo_cons,opérationnelle).
nombre_systèmes(c_2s_2i_gazo_cons,2).
nombre_instants(c_2s_2i_gazo_cons,2).

/* le père */
père(c_2s_2i_gazo_cons,c_2s_2i_gazo).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_gazo_cons):-résoudre([1,2,3,5]).

/*************** LA CLASSE 2S2I_gazo_sans ************************/

statut(c_2s_2i_gazo_sans,opérationnelle).

/* le père */
père(c_2s_2i_gazo_sans,c_2s_2i_gazo).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_gazo_sans):-résoudre([1,2,5]).
```

```
/*************** LA CLASSE 2S2I_surf ************************/

statut(c_2s_2i_surf,non_opérationnelle).

test_fils(c_2s_2i_surf,P,conservation(P,B)):-eval_prem(problème(P)).

valeur_fils(c_2s_2i_surf,P,c_2s_2i_surf_cons):-
            eval_prem(problème(P)),eval_prem(conservation(P,oui)).
valeur_fils(c_2s_2i_surf,P,c_2s_2i_surf_sans):-
            eval_prem(problème(P)),eval_prem(conservation(P,non)).

/* le père */
père(c_2s_2i_surf,c_2s_2i).

infos(c_2s_2i_surf,P):-write('deux systèmes, 2 instants, systèmes de surface').

/*************** LA CLASSE 2S2I_surf_cons ************************/

statut(c_2s_2i_surf_cons,opérationnelle).

/* le père */
père(c_2s_2i_surf_cons,c_2s_2i_surf).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_surf_cons):-résoudre([1,2,3,4]).

/*************** LA CLASSE 2S2I_surf_sans ************************/

statut(c_2s_2i_surf_sans,opérationnelle).

/* le père */
père(c_2s_2i_surf_sans,c_2s_2i_surf).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_surf_sans):-résoudre([1,2,4]).

/*************** LA CLASSE 2S2I_ambiance ************************/

statut(c_2s_2i_ambiance,non_opérationnelle).

test_fils(c_2s_2i_ambiance,P,conservation(P,B)):-eval_prem(problème(P)).

valeur_fils(c_2s_2i_ambiance,P,c_2s_2i_ambiance_cons):-
            eval_prem(problème(P)),eval_prem(conservation(P,oui)).
valeur_fils(c_2s_2i_ambiance,P,c_2s_2i_ambiance_sans):-
            eval_prem(problème(P)),eval_prem(conservation(P,non)).

/* le père */
père(c_2s_2i_ambiance,c_2s_2i).

infos(c_2s_2i_ambiance,P):-write('deux systèmes, 2 instants, système fermé en
contact avec un milieu extérieur').

/*************** LA CLASSE 2S2I_ambiance_cons ************************/

statut(c_2s_2i_ambiance_cons,opérationnelle).

/* le père */
père(c_2s_2i_ambiance_cons,c_2s_2i_ambiance).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_ambiance_cons):-résoudre([1,2,3]).
```

```
/*************** LA CLASSE 2S2I_ambiance_sans ************************/

statut(c_2s_2i_ambiance_sans,opérationnelle).

/* le père */
père(c_2s_2i_ambiance_sans,c_2s_2i_ambiance).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_ambiance_sans):-résoudre([1,2]).

/*************** LA CLASSE 2S2I_mélange ************************/

statut(c_2s_2i_mélange,non_opérationnelle).

test_fils(c_2s_2i_mélange,P,conservation(P,B)):-eval_prem(problème(P)).

valeur_fils(c_2s_2i_mélange,P,c_2s_2i_mélange_cons):-
            eval_prem(problème(P)),eval_prem(conservation(P,oui)).
valeur_fils(c_2s_2i_mélange,P,c_2s_2i_mélange_sans):-
            eval_prem(problème(P)),eval_prem(conservation(P,non)).

/* le père */
père(c_2s_2i_mélange,c_2s_2i).

infos(c_2s_2i_mélange,P):-write('deux systèmes, 2 instants, système mélange de
deux gaz').

/*************** LA CLASSE 2S2I_mélange_cons ************************/

statut(c_2s_2i_mélange_cons,opérationnelle).

/* le père */
père(c_2s_2i_mélange_cons,c_2s_2i_mélange).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_mélange_cons):-résoudre([1,2,3,6]).

/*************** LA CLASSE 2S2I_mélange_sans ************************/

statut(c_2s_2i_mélange_sans,opérationnelle).

/* le père */
père(c_2s_2i_mélange_sans,c_2s_2i_mélange).

/* la méthode de résolution associée à la classe */
employer_résolution(c_2s_2i_mélange_sans):-résoudre([1,2,6]).

/*****************************************************/

/* aller chercher les attributs chez le père */
attribut(Classe,P,Att,Pred,Liste,Def):-
      père(Classe,Père),attribut(Père,P,Att,Pred,Liste,Def).
```

F3/ Quelques règles de reformulation

```
règle(nb_sys):-
  si(     [problème(P),
           systèmes(P,L),
           eval_prolog(length(L,N))]),
  alors([ajouter(nb_systèmes(P,N))]).

règle(nb_ins):-

règle(isochore):-
  si(     [problème(P),
           isochore(Sys,Ii,If),
           instants(P,Li),
           systèmes(P,Ls),
           eval_prolog(on(Sys,Ls)),
           eval_prolog(on(Ii,Li)),
           eval_prolog(on(If,Li)),]),
  alors([ajouter(conservation(P,oui))]).

règle(isotherme):-

règle(isobare):-

règle(adiabatique):-

règle(isobare2):-

règle(cons_inv1):-

règle(cons_loi1):-

/* règle par défaut */
règle(sans):-
  si(     [problème(P),
           absent(conservation(P,oui))]),
  alors([ajouter(conservation(P,non))]).

règle(gazo):-
  si(     [problème(P),
           systèmes(P,Ls),
           gazomètre(Sys),
           eval_prolog(on(Sys,Ls))]),
  alors([ajouter(gazo(P))]).

règle(comp1):-
  si(     [problème(P),
           gazo(P)]),
  alors([ajouter(composé(P,gazomètre))]).

règle(comp2):-
  si(     [problème(P),
           surface(P)]),
  alors([ajouter(composé(P,surface))]).

règle(comp3):-
  si(     [problème(P),
           mélange([S1,S2],I,S3,F),
           systèmes(P,Ls),
           eval_prolog(on(S1,Ls)),
           eval_prolog(on(S2,Ls)),
           instants(P,Li),
           eval_prolog(on(I,Li)),
           eval_prolog(on(F,Li))]),
  alors([ajouter(composé(P,mélange))]).

règle(comp4):-
  si(     [problème(P),
           ambiance(S1,S2,I,F),
           systèmes(P,Ls),
           eval_prolog(on(S1,Ls)),
           eval_prolog(on(S2,Ls)),
           instants(P,Li),
           eval_prolog(on(I,Li)),
           eval_prolog(on(F,Li))]),
  alors([ajouter(composé(P,ambiance))]).
règle(surf1):-
  si(     [problème(P),
      état_circulateur(Sys,B,O,I,Sv,Ivi,Ivf),
           systèmes(P,Ls),
           eval_prolog(on(Sys,Ls)),
           instants(P,Li),
           eval_prolog(on(I,Li))]),
  alors([ajouter(surface(P))]).

règle(surf2):-

règle(int):-
  si(     [problème(P),
           eval_prolog(liste_grandeurs(P,L)),
           eval_prolog(toutes_intensives(L)),
           on_demande(P,Buts),

  eval_prolog(buts_intensifs(Buts))]),
  alors([ajouter(type_variables(P,int))]).

règle(ext):-

règle(cas_part_inv):-
  si(     [problème(P),
           systèmes(P,[S]),
           instants(P,[I,F]),
           trans_gp_fermé(S,I,F),
        eval_prolog(invariant(S,I,F,Var_inv)),
  eval_prolog(enlever(Var_inv,[pression,
           température,volume],[Var1,Var2])),
           Var1(S,I),
           Var2(S,I),
           Var(S,F),
       eval_prolog(member_s(Var,[Var1,Var2])),
  eval_prolog(enlever(Var,[Var1,Var2],[Vari])
),
           on_demande(P,[Vari(S,F)])]),
  alors([ajouter(cas_part(P,invariant)),
  ajouter(liste_inv(P,S,I,F,Var_inv,Var1,Var2
,Var,Vari))]).

règle(cas_part_gam):-

règle(cas_part_nrj1):-

règle(cas_part_nrj2):-

règle(cas_part_nrj_w_inv):-

règle(cas_part_nrj_w_gam):-

règle(cas_part_nrj_Q_it):-


règle(cas_part_nrj_DU_ad):-
  si(     [problème(P),
           systèmes(P,[S]),
           instants(P,[I,F]),
           trans_gp_fermé(S,I,F),
           adiabatique(S,I,F),
  on_demande(P,[variation(nrj_interne,S,I,F)]
)]),
  alors([ajouter(cas_part_nrj(P,du_ad))]).
```

F4/ Les plans-type associés aux cas particuliers

Classe cas_particulier_invariant

Parmi les grandeurs P,T,V, l'une est invariante, les deux autres sont connues à l'instant initial. Parmi ces deux, l'une est de plus connue à l'instant final, et on demande de calculer l'autre.

Soit S le système en transformation entre l'instant I et l'instant F. Soit Var_inv la variable invariante pendant la transformation. Soit Var1 et Var2 les deux autres grandeurs. Soit Vari la grandeur qu'il faut calculer (Var1 ou Var2), et Var l'autre grandeur (Var2 ou Var1).

Le plan-type est le suivant :

> S est un système fermé (n=cte), et la variable Var_inv est constante entre I et F, donc il existe une fonction f(Var1,Var2) constante entre I et F. Comme on connaît Var1(S,I) et Var2(S,I), on connaît la valeur de cette fonction. Comme d'autre part on connaît Var(S,F), on en déduit Vari(S,F).

Classe cas_particulier_gamma

La transformation est adiabatique, et on connaît γ. Parmi les grandeurs P,T,V, deux sont connues à l'instant initial. Parmi ces deux, l'une est de plus connue à l'instant final, et on demande de calculer l'autre.

Soit S le système en transformation entre l'instant I et l'instant F. Soit Var et Vari les deux grandeurs connues à l'instant I. Vari est la grandeur qu'il faut calculer et Var l'autre grandeur.

Si Var et Vari sont P et V (ou V et P), le plan-type est le suivant :

> Le système S est adiabatique entre les instants I et F. Comme on connaît gamma, grâce à la loi de Laplace on sait qu'il existe une fonction f(Var,Vari) constante entre I et F. Comme on connaît Var(S,I) et Vari(S,I), on connaît la valeur de cette fonction. Comme d'autre part on connaît Var(S,F), on en déduit Vari(S,F).

Sinon, le plan-type est le suivant :

> Le système S est adiabatique entre les instants I et F. Comme on connaît gamma, on utilise la loi de Laplace et le fait que PV/T=cte (système fermé), on en déduit qu'il existe une fonction f(Var,Vari) constante entre I et F. Comme on connaît Var(S,I) et Vari(S,I), on connaît la valeur de cette fonction. Comme d'autre part on connaît Var(S,F), on en déduit Vari(S,F).

Classe cas_particulier_invariant_W

Parmi les grandeurs P,T,V, l'une est invariante, les deux autres sont connues à l'instant initial. Parmi ces deux, l'une est connue à l'instant final, et on demande de calculer le travail reçu par le système.

Soit S le système en transformation entre l'instant I et l'instant F. Soit Var_inv la variable invariante pendant la transformation. Soit Var1 et Var2 les deux autres grandeurs. Soit Var la grandeur connue à l'instant F (Var1 ou Var2).

Si Var 1 et Var2 sont P et V (ou V et P), le plan-type est le suivant :

S est un système fermé (n=cte), et la variable Var_inv est constante entre I et F, donc il existe une fonction f(Var1,Var2) constante entre I et F. Comme on connaît Var1(S,I) et Var2(S,I), on connaît la valeur de cette fonction. Comme d'autre part on connaît Var(S,F), on peut intégrer dw en Var pour obtenir le travail du système S entre les instants I et F.

Sinon, le plan-type est le suivant :

S est un système fermé (n=cte), et la variable Var_inv est constante entre I et F, donc il existe une fonction f(Var1,Var2) constante entre I et F. Comme on connaît Var1(S,I) et Var2(S,I), on connaît la valeur de cette fonction. Comme d'autre part PV/T=cte et qu'on connaît Var(S,F), on peut intégrer dw en Var pour obtenir le travail du système S entre les instants I et F.

Classe cas_particulier_gamma_W

La transformation est adiabatique, et on connaît γ. Parmi les grandeurs P,T,V, deux sont connues à l'instant initial. Parmi ces deux, l'une est connue à l'instant final, et on demande de calculer le travail reçu par le système.

Soit S le système en transformation entre l'instant I et l'instant F. Soit Var et Vari les deux grandeurs connues à l'instant I. Var est la grandeur qui est également connue à l'instant F.

Si Var 1 et Var2 sont P et V (ou V et P), le plan-type est le suivant :

Le système S est adiabatique entre les instants I et F. Comme on connaît gamma, grâce à la loi de Laplace on sait qu''il existe une fonction f(Var,Vari) constante entre I et F. Comme on connaît Var(S,I) et Vari(S,I), on connaît la valeur de cette fonction. Comme d''autre part on connaît Var(S,F), on peut intégrer dw en Var pour obtenir le travail du système S entre les instants I et F.

Sinon, le plan-type est le suivant :

Le système S est adiabatique entre les instants I et F. Comme on connaît gamma, on utilise la loi de Laplace et le fait que PV/T=cte (système fermé), on en déduit qu''il existe une fonction f(Var,Vari) constante entre I et F. Comme on connaît Var(S,I) et Vari(S,I), on connaît la valeur de cette fonction. Comme d''autre part on connaît Var(S,F), on peut intégrer dw en Var pour obtenir le travail du système S entre les instants I et F.

Annexe G
Caractéristiques techniques

Matériel utilisé

SYRCLAD a été implémenté en MacPROLOG 3.0 sur un Macintosh LC III. Ce macintosh a un processeur MC68030, la fréquence d'horloge est 25 MHz, et il a 4 Mo de mémore vive.

Taille des systèmes

SYRCLAD-dénombrements correspond à 2500 lignes de Prolog.

SYRCLAD-additifs correspond à 2000 lignes de Prolog.

SYRCLAD-tarot correspond à 700 lignes de Prolog.

SYRCLAD-thermodynamique correspond à 1800 lignes de Prolog.

Rapidité

Notre effort n'a pas porté sur la rapidité des systèmes SYRCLAD-X. Nous avons mesuré artisanalement le temps qu'ils mettent à résoudre (affichage compris) sur le matériel que nous avons utilisé. Ces systèmes résolvent la plupart des problèmes en quelques secondes.

Dénombrement :
Les problèmes sont résolus en moins de 10 secondes, en moyenne 7.
Les problèmes dont les résolutions sont composées sont résolus en moins de 20 secondes.

Problèmes additifs :
Les problèmes sont résolus en 2 ou 3 secondes.
Les problèmes décomposés par comparaison au graphe sont résolus en environ 25 secondes.

Tarot :
Les plans de jeu sont trouvés en une seconde.

Thermodynamique :
Les problèmes débouchant sur un plan-type sont résolus en une seconde.
Les problèmes débouchant sur un plan-type mais décomposés par comparaison au graphe sont résolus en environ 5 secondes.
Les problèmes (décomposés ou non) qui demandent l'utilisation de THERMOS pour la partie résolution sont résolus beaucoup plus lentement. En effet, le classement est rapidement effectué mais

la résolution n'est pas algorithmique et THERMOS utilise les règles "quand" en marche avant, et cela peut demander de 8 secondes pour les problèmes simples à 5 minutes pour les problèmes où il y a beaucoup de faits. L'utilisation des règles "quand" par THERMOS peut être optimisée en utilisant la méthode du résolveur de Modélis.

Si on excepte le problème dû à THERMOS, les problèmes décomposés prennent plus de temps car il y a plusieurs sous-problèmes résolus et il faut parfois d'abord trouver les sous-problèmes à résoudre. Parmi les problèmes non décomposés, les différences de temps suivant les domaines viennent du nombre de règles de reformulation du domaine : s'il y a beaucoup de règles, on met plus de temps à trouver la bonne.

Bibliographie

[Anderson et al 87] J.R. Anderson, C.F. Boyle, R .Farell, B.J. Reiser : "Cognitive principles in the design of computer tutors". *Modelling Cognition*, P. Morris ed., Willey & sons Ltd, 1987.

[Antibi 88] A. Antibi : *Étude sur l'enseignement de méthodes de démonstration. Enseignement de la notion de limite : réflexions, propositions*. Thèse d'état de l'Université Paul Sabatier de Toulouse, 27 juin 1988, Chapitre 5.

[Bailly 71] M. Bailly : *Thermodynamique technique*, tome 3. Bordas, 1971.

[Baron 84] M. Baron : "D'un "SE" à un "SEIAO", les enseignements d'un cas intéressant : GUIDON, notes de lecture". *Colloque Intelligence Artificiell*e, Aix-en-Provence, septembre 1984. Publication n° 49 du GR C.F. Picard (LAFORIA), p. 27-49.

[Baron 93] M. Baron : "Connaissances, métaconnaissances et EIAO, quelques aspects". *Métaconnaissances en IA, en EIAO et en didactique des mathématiques*, groupe de travail Math & Méta (1990-1992). Rapport interne LAFORIA 93/18, juin 1993.

[Bazin 93] J.-M. Bazin : *GEOMUS : Un résolveur de problèmes de géométrie qui mobilise ses connaissances en fonction du problème posé*. Thèse de l'université Paris 6, 26 octobre 1993.

[Bellemain 92] F. Bellemain : *Conception, réalisation et expérimentation d'un logiciel d'aide à l'enseignement de la géométrie, Cabri-Géomètre.* Thèse de l'université Joseph Fourrier (Grenoble I), 1992.

[Bensimon-Darcel 93] N. Bensimon-Darcel : *ObAdE : des objets pour une modélisation cognitive*. Thèse de l'Université Paris-Nord, Villetaneuse, 1993.

[Bertin et Renault 84] M. Bertin, J. Renault : *Exercices de thermodynamique.* Dunod, 1984.

[Boutigny 85] J. Boutigny : *Thermodynamique, classe de mathématique supérieure.* Vuibert, 1985.

[Briand 93] J. Briand : *L'énumération dans le mesurage des collections. Un dysfonctionnement dans la transposition didactique*. Thèse de l'université de Bordeaux I (spécialité didactique des mathématiques), 14 décembre 1993.

[Brachman et Schmolze 85] R.J. Brachman, J.G. Schmolze : "An overview of the KL-ONE knowledge representation system". *Cognitive Science* 9(2), April-June 1985, p.171-276.

[Carpenter et Moser 83] T.P. Carpenter et J.M. Moser : "The acquisition of addition and substraction concepts". In R. Lesh & M. Landau (Eds.), *Acquisition of mathematics concepts and processes*. New-York: Academic Press, 1983.

[Cauzinille-Marmèche et Mathieu 88] E. Cauzinille-Marmèche, J. Mathieu : "Adapter les interventions tutorielles au modèle cognitif de l'étudiant". Dans J.P. Caverni, *Psychologie Cognitive, modèles et méthodes*, PUG, 1988, p. 175-189.

[Chaillot et Crochon 94] M. Chaillot, E. Crochon : "Sélection rationnelle de méthodes dans un environnement réactif de résolution de problèmes", *9ième RFIA*, Paris, 1994, Tome 2, p. 293-303.

[Chi et al 81] M. Chi, P. Feltovich, R. Glaser : "Categorization and representation of physics problems by experts and novices". *Cognitive Science* 5, 1981, p. 121-152.

[Chi 87] M. Chi : "Representing Knowledge and Metaknowledge : Implications for Interpreting Metamemory Research", in *Metacognition, motivation and understanding*, Chap. 8, F.E. Weinert, R.H. Kluwe Eds, 1987.

[Choquet et al 96] C. Choquet, P. Tchounikine, F. Trichet : *Le projet ELMER : premier rapport*.. Rapport de recherche IRIN-145, Nantes, 1996.

[Choquet et al 97] C. Choquet, P. Tchounikine, F. Trichet : "La modélisation de la méthode de résolution de problèmes dans le système EMMA". *EIAO'97*, actes des cinquièmes journées EIAO de Cachan (M. Baron, P. Mendelsohn, J.-F. Nicaud eds.), Hermès, 1997, p. 263-275.

[Clancey 83] W.J. Clancey : "The epistemology of a rule based expert system: A framework for explanation". *Artificial Intelligence* 20, 1983, p. 215-251.

[Clancey et Letsinger 84] W.J. Clancey, R. Letsinger : "NEOMYCIN, reconfiguring a rule-based expert system for application to teaching". *Medical Artificial Intelligence: The First Decade* (Clancey & Shortlife eds.), Addison-Wesley, Reading, Massachusets, 1984.

[Clancey 85] W.J. Clancey : "Heuristic classification". *Artificial Intelligence* 27, 1985, p. 289-350.

[Clancey 87] W.J. Clancey : *Knowledge-based tutoring : the GUIDON program*. Cambridge, M.A.:MIT Press, 1987.

[Delozanne 92] E. Delozanne : *Explications en EIAO. Etude à partir d'ELISE, un logiciel pour s'entraîner à une méthode de calcul de primitives*. Thèse de l'université du Maine, janvier 1992.

[Delozanne 94] E. Delozanne : "Un projet pluridisciplinaire : ELISE, un logiciel pour donner des leçons de méthode". *Recherches en Didactique des mathématiques* 14 (1.2), 1994, p. 211-250.

[Desmoulins 94] C. Desmoulins : *Conception et réalisation d'un système tuteur pour la construction de figures géométriques.* Thèse de l'université Joseph Fourrier (Grenoble I), 2 février 1994.

[Desmoulins 95] C. Desmoulins : "La détection de solutions particulières dans TALC : une approche logique basée sur des extensions de l'énoncé du problème". *EIAO* tome 2, actes des

quatrièmes journées EIAO de Cachan (D. Guin, J.-F. Nicaud, D. Py éds.), Eyrolles, 1995, p. 161-172.

[Didierjean et Cauzinille-Marmèche 97] A. Didierjean, E. Cauzinille-Marmèche : "Eliciting self-explanations improves problem solving: What processes are involved ?" *Current Psychology of Cognition* 16 (3), june 1997, p. 325-351.

[Dubois 84] J.-G. Dubois : "Une systématique des configurations combinatoires simples". *Educational Studies in Mathematics* 15, 1984, p.37-57.

[Elio et Scharf 90] R. Elio, P Scharf : "Modeling novice-to-expert shifts in problem-solving strategy and knowledge organization", *Cognitive Science* 14, 1990, p. 579-639.

[Fayol 90] M. Fayol : *L'enfant et le nombre*. Neuchâtel. Paris : Delachaux & Niestlé, 1990.

[Fikes et Nilsson 71] R.E. Fikes, N.J. Nilsson : STRIPS : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2, 1971, p. 189-208.

[Fischbein et Grossman 97] E. Fischbein et A. Grossman : "Schemata and intuitions in cominatorial reasoning". *Educational Studies in Mathematics* 34, 1997, p. 27-47.

[Gick et Holyoak 80] M. Gick, K.J. Holyoak : "Analogical problem solving". *Cognitive Psychology* 12, 1980, p. 306-355.

[Gick et Holyoak 83] M. Gick, K.J. Holyoak : "Schema induction and analogical transfer". *Cognitive Psychology* 15, 1983, p. 1-38.

[Grenier 94] D. Grenier : "Savoirs mis en jeu dans les problèmes de combinatoire". GR CNRS Didactique - Groupe "savoirs et connaissances", Colloque GDR-INRP *Différents types de savoirs et leurs articulations*, Lyon, 12-13 décembre 1994.

[Greeno et Riley 87] J.G.Greeno, M.S. Riley : "Processes ans Development of Understanding", in *Metacognition, motivation and understanding*, F.E. Weinert, R.H. Kluwe Eds, Chap. 10, p. 289-313.

[Guin 91] D. Guin : "La notion d'opérateur dans une modélisation cognitive de la compréhension des problèmes additifs". *Math. Inf. Sci. hum.*, 29e année, no 113, 1991, p. 5-33.

[Guin 94] N. Guin : *Résolution d'exercices de dénombrement par reconnaissance de leur classe.* Rapport interne du LAFORIA 94/23 , octobre 1994.

[Guin et al 95] N. Guin, H. Giroire, G. Tisseau. "Le classement de problèmes : une méthode de résolution pour le module expert d'un EIAO. Application aux problèmes de dénombrement". *EIAO* tome 2, actes des quatrièmes journées EIAO de Cachan (D. Guin, J.-F. Nicaud, D. Py éds.), Eyrolles, 1995, p. 113-124.

[Hammond 90] K.J. Hammond : "Case-based planning: A framework for planning from experience". *Cognitive Science* 14, 1990, p. 385-443.

[Hoppe et al 93] T. Hoppe, C. Kindermann, J. Quantz, J. Schmiedel, M. Fischer : *Back V5 Tutorial & Manual*, Institut fûr Software und theoretische Informatik, w-1000 Berlin 10, Allemagne, mars 1993.

[Jeannin-Naltet et Garrivet 83] E. Jeanin-Naltet, Garrivet : *Le tarot moderne ; de la partie amicale à la compétition*, éditions du Rocher, 1983.

[Kolodner 91] J.L. Kolodner : "Improving Human Decision Making through Case-Based Decision Aiding". *The AI Magazine* 12 (2), 1991, p. 52-68.

[Leblanc et Weber-Russel 96] M.D. LeBlanc, S. Weber-Russel : "Text Integration and Mathematical Connections: A Computer Model of Arithmetic Word Problem Solving". *Cognitive Science* 20, 1996, p. 357-407.

[Le Calvez et al 97] F. Le Calvez, M. Urtasun, H. Giroire, G. Tisseau, J. Duma : "Les machines à construire. Des modèles d'interaction pour apprendre une méthode constructive de dénombrement". *EIAO'97*, actes des cinquièmes journées EIAO de Cachan (M. Baron, P. Mendelsohn, J.-F. Nicaud eds.), Hermès, 1997, p. 49-60.

[Marthe 82] P. Marthe : *Problèmes de type additif et appropriation par l'élève des groupes additifs Z et D entiers relatifs et décimaux relatifs*. Thèse de 3ème cycle, École des Hautes Études en Sciences Sociales, Paris, 1982.

[Maury et Hulin 86] J.-P. Maury, M. Hulin : *Thermodynamique, les deux principes*. Armand Colin, 1986.

[Napoli 92a] A. Napoli : *Représentations à objets et raisonnement par classification en intelligence artificielle.* Thèse d'état de l'Université de Nancy I, 31 janvier 1992.

[Napoli 92b] A. Napoli : "Subsumption and classification-based reasoning in object-based representations". In *Proceedings of the Tenth European Conference on Artificial Intelligence*, B. Neumann ed., published by J. Wiley & Sons, 1992, p. 425-429.

[Napoli et Laurenço 93] A. Napoli, C. Laurenço : "Représentation à objets et classification. Conception d'un système d'aide à la planification de synthèses organiques". *Revue d'intelligence artificielle* 7(2), p. 175-221.

[Napoli et Volle 93] A. Napoli, P. Volle : *Une introduction aux logiques terminologiques*. Rapport du CRIN n° 93-R-033, novembre 1993.

[Newel et al 60] A. Newel, J.C. Shaw, H.A. Simon : Report on a general problem-solving program for a computer, Information Processing : *Proceedings of the International Conference of Information Processing*, Paris, Unesco, 1960, p. 256-264.

[Newel 82] A. Newel : "The Knowledge Level", *Artificial Intelligence* 18, 1982, p. 87-127.

[Nicaud 94] J.-F. Nicaud : "Modélisation en EIAO, les modèles d'APLUSIX". *Recherches en Didactique des mathématiques* 14 (1.2), 1994, p. 67-112.

[Nigro 95] J.-M. Nigro : *La réalisation et la conception d'un générateur automatique de commentaires : le système GénéCom. Application au jeu du Tarot.*. Thèse de l'université Paris 6, 12 janvier 1995.

[Paquette 91] G. Paquette : *Métaconnaissance dans les environnements d'apprentissage.* Thèse de l'Université du Maine, Le Mans, octobre 1991.

[Pecego 97] G. Pecego : "Auto-évaluation des énoncés par SYGEP : un Système de Génération d'Enoncés de Problèmes".*EIAO'97*, actes des cinquièmes journées EIAO de Cachan (M. Baron, P. Mendelsohn, J.-F. Nicaud eds.), Hermès, 1997, p. 159-170.

[Pintado 94] M. Pintado : *Apprentissage et démonstration automatique de théorèmes*. Thèse de l'université Paris 6, 13 décembre 1994.

[Pitrat 90] J. Pitrat : *Métaconnaissance, futur de l'intelligence artificielle.* Hermès, 1990.

[Rebillard 95] E. Rebillard : *Problèmes additifs : analyse et catégorisations d'enseignants débutants*. DEA de didactique des disciplines scientifiques, université Montpellier II, 1995.

[Rechenmann et al 90] F. Rechenmann, P. Fontanille, P. Uvietta : *Shirka, système de gestion de bases de connaissances centrées-objet*, manuel d'utilisation, INRIA Rhône-Alpes, 1990.

[Rechenmann et Rousseau 92] F. Rechenmann, B. Rousseau : "A development shell for knowledge-based systems in scientific computing". *Expert Systems for Numerical Computing*, E.N. Houstis, J.R. Rice eds, Elsevier science Publishers, New-York USA, 1992, p. 157-173.

[Renk 97] A. Renkl : "Learning from worked-out examples : a study on individual differences". *Cognitive Science* 21 (1), 1997, p. 1-29.

[Riesbeck et Schank 89] C.K. Riesbeck, R.C. Schank : *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.

[Riley et al 83] M.S Riley, J.G. Greeno, J.I. Heller : "Development of children's problem-solving ability in arithmetic". *The development of mathematical thinking*. H.P. Ginsburg ed., New-York : Academic Press, 1983.

[Robert et al 87] A. Robert, J. Rogalski, R. Samurcay : "Enseigner des méthodes". *Cahier de Didactique des Mathématiques* n° 38, Irem, Université Paris 7, 1987.

[Robert et Tenaud 89] A. Robert, I. Tenaud : "Une expérience d'enseignement de la géométrie en terminale C". *Recherche en didactique des mathématiques* 9.1, 1989.

[Robert et Robinet 96] A. Robert, J. Robinet : "Prise en compte du méta en didactiques des mathématiques", *Recherches en Didactique des Mathématique*s 16 (2), 1996, p. 145-176.

[Rogalski 88] M. Rogalski : *Comment chercher une primitive ? Question de méthode...*, Polycopié DEUG A1, université des sciences et techniques de Lille, 1988.

[Rogalski 90] M. Rogalski : "Enseigner des méthodes en mathématiques". Commission Inter-Irem Université, *Enseigner autrement les mathématiques en Deug A première année*, bulletin Inter-Irem 1990, p. 65-79.

[Rogalski 91] M. Rogalski : "Un enseignement d'Algèbre lineaire en DEUG A premiere année", *Cahier DIDIREM* n° 11, IREM de Paris VII, 1991

[Rogalski 94] Rogalski M : "Les concepts de l'EIAO sont-ils indépendants du domaine? L'exemple d'enseignement de méthodes en analyse". *Recherches en Didactiques des Mathématiques* 14 (1.2), 1994, p. 43-66.

[Rolland et Pachet 95] P.-Y. Rolland, F. Pachet : *Représentation de connaissances sur la programmation de synthétiseurs*. Rapport interne LAFORIA 95/09, mars 1995.

[Ross et Kennedy 90] B.H. Ross, P.T. Kennedy : "Generalizing from the use of earlier examples in problem solving". *Journal of Experimental Psychology: Learning, Memory, and Cognition* 16 (1), 1990, p. 42-55.

[Schreck 93] P. Schreck : *Automatisation des constructions géométriques à la règle et au compas*. Thèse de l'université Louis Pasteur de Strasbourg, 12 janvier 1993.

[Séroussi et Morice 93] B. Séroussi, V. Morice : "Problem-solving expertise acquisition : a computational model". *Revue d'intelligence artificielle* 7(2), 1993, p. 223-273.

[Shortliffe 76] E.H. Shortliffe : *Computer-based medical consultations: MYCIN*. New York: American Elsevier Publishers, 1976.

[Suardet 85] R. Suardet : *Thermodynamique, physique de la matière*. Technique et documentation (Lavoisier), 1985.

[Tisseau 90] G. Tisseau : *Modélisation à partir d'un énoncé informel : Le système Modelis. Application à des exercices de thermodynamique*. Thèse de l'université Paris 6, 29 juin 1990.

[Vergnaud 82] G. Vergnaud : "A classification of cognitive tasks and operations of thought involved in addition and substraction problems". *Addition and substraction : A cognitive perspective*. T.P. Carpenter, J.M. Moser & T.A. Romberg eds, Hillsdale: Erlbaum, 1982.

[Vergnaud 85] G. Vergnaud : "Concepts et schèmes dans une théorie opératoire de la représentation". *Psychologie française*, 1985, p. 245-251.

[Vivet 87] M. Vivet : "Systèmes experts pour enseigner : méta-connaissances et explications". *Congrès CESTA : Mari-Cognitiva 87*, Paris, 18-22 mai 1987.

[Vivet 88] M. Vivet : "Knowledge-based tutors. Towards a shell". *International Journal of Educational Research* 12 (8), 1988, p. 839-850.

[Willamowski 94] J. Willamowski : *Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur*. Thèse de l'université Joseph Fourier - Grenoble 1, 6 avril 1994.