Une méthode "constructive" de résolution de problèmes de dénombrement et sa mise en œuvre

Gérard TISSEAU,
Hélène GIROIRE
LAFORIA UA 1095
Boîte 169, Tour 46-0 2° étage
4 Place Jussieu
F-75252 Paris Cedex 05

email : giroire@laforia.ibp.fr
téléphone : [33-1] 44 27 70 05
fax : [33-1] 44 27 70 00

Françoise LE CALVEZ,
Marie URTASUN
Jeune Équipe Informatique et communication
UFR de Mathématique et Informatique
Université René Descartes
45 rue des Saints Pères
F-75270 Paris Cedex 06

Jacques DUMA
Lycée technique Jacquard
2 rue Bouret
75019 Paris

Résumé : Nous présentons une méthode de résolution de problèmes de dénombrement appelée "méthode constructive de dénombrement". Elle consiste à élaborer une définition de l'ensemble à dénombrer qui puisse être interprétée comme un algorithme (éventuellement non déterministe) de génération des éléments de cet ensemble. La définition est dite "constructive" si l'algorithme n'effectue aucune élimination, c'est-à-dire si tout essai conduit à coup sûr à un élément solution. Sous certaines conditions, l'étude de la structure de l'algorithme permet de déterminer le cardinal de l'ensemble à dénombrer. Dans le cadre du projet COMBIEN?, nous avons réalisé un système de résolution de problèmes de dénombrement qui utilise cette méthode. Par ailleurs, celle-ci fournit un cadre conceptuel qui permet actuellement à un utilisateur de communiquer au système un énoncé d'exercice et une proposition de solution, et au système d'étudier la validité de cette proposition.
Mots-clés : intelligence artificielle, représentation des connaissances, démonstration automatique de théorèmes, EIAO, dénombrement, méthode constructive de dénombrement.

Abstract: A method for solving problems in combinatorial mathematics called "constructive method of counting" is presented in this paper. It consists in elaborating a definition of the set to be enumerated that can be interpreted as a non deterministic algorithm capable to generate the elements of the set. The definition is said to be "constructive" if the algorithm never eliminates a proposed element by backtracking. Under certain conditions the study of the structure of the algorithm allows to calculate the number of elements in the set. Within the framework of the COMBIEN? project, we have

implemented a system that solves combinatorial problems using this method. Besides, the method yields a conceptual framework that allows the user to input the text of a problem and a proposal for its solution. Next, this conceptual framework allows the system to study the validity of this proposal.
Keywords : artificial intelligence, knowledge representation, automatic theorem proving, intelligent tutoring systems, combinatorial mathematics, constructive method of counting.

1. INTRODUCTION

Le projet Combien? a pour but de réaliser un environnement interactif d'apprentissage avec ordinateur pour des exercices de dénombrement au niveau de la classe de terminale scientifique. Il vise à familiariser les élèves avec une méthode de résolution que nous avons appelée "méthode constructive de dénombrement". Dans cet article, nous décrivons cette méthode puis ses applications. D'une part, elle a permis la réalisation d'un système capable de résoudre des exercices et de justifier ses solutions. D'autre part elle fournit un cadre conceptuel dans lequel un utilisateur peut communiquer un énoncé d'exercice et une proposition de solution. Le système est capable d'étudier la validité de cette proposition, sans toutefois pouvoir conclure dans tous les cas car le problème est complexe et ne peut pas être résolu simplement par un algorithme général.

2. LA METHODE CONSTRUCTIVE DE DENOMBREMENT

Nous décrivons d'abord la forme des exercices considérés, puis nous introduisons certains concepts : la définition constructive, le programme de construction, la multiplicité algorithmique et la propriété d'injectivité. Nous énonçons ensuite la méthode.

2.1. L'ensemble à dénombrer

Les exercices que nous considérons sont tous réductibles à la forme suivante : combien y a-t-il d'éléments d'un ensemble donné vérifiant une conjonction de contraintes données ? Les éléments sont des structures ou "configurations" (ensembles, listes, fonctions) et les contraintes portent sur les nombres d'occurrences de certaines valeurs. La figure 1 en donne un exemple. Par souci de clarté, nous avons choisi pour cet article des exemples simples. Les exercices qu'on rencontre dans les livres et qui sont traités par notre système peuvent être plus difficiles : les ensembles peuvent être définis de manière complexe et les contraintes peuvent être multiples. Nous travaillons actuellement sur un corpus d'une centaine d'exercices, dont soixante sont traités par le système.

> Combien y a-t-il de mots de longueur 2 formés avec des lettres de l'alphabet {a, b, c} et comportant exactement une occurrence de la lettre a ?

Figure 1 — Un exemple d'exercice

Nous appellerons *ensemble à dénombrer* l'ensemble des éléments qui vérifient la conjonction des contraintes et nous le noterons E_d dans la suite. Le problème est de déterminer son cardinal.

2.2. La formalisation d'un énoncé de problème

Pour représenter un énoncé, nous utilisons un formalisme de type logique [Tisseau et al 93, 94]. Nous avons préféré ne pas le détailler dans cet article et donner les formes équivalentes dans le langage Prolog. L'idée de base est la suivante : tout ensemble E peut être associé à un prédicat unaire p jouant le rôle de propriété caractéristique, c'est-à-dire que X est un élément de E si et seulement si p(X) est vérifiée. Prolog permet de définir un tel prédicat par l'intermédiaire de faits et de règles constituant un programme (équivalent à une théorie dans la logique du premier ordre). La figure 2 donne le programme associé à l'exercice de la figure 1, la propriété caractéristique étant ici le prédicat `configuration_solution`. Cette définition traduit directement l'énoncé en utilisant les prédicats `est_une_liste` et `effectif_element` pour exprimer les contraintes. Notre système comporte une quinzaine de tels prédicats prédéfinis.

```
configuration_solution(X) :-
         est_une_liste(X,2,[a,b,c]),
         effectif_element(X,exactement,1,a).

est_une_liste(X,N,A)
```

signifie que X est une liste de longueur N formée avec les éléments de l'ensemble A.

```
effectif_element(X,exactement,1,a)
```

signifie que X contient exactement une occurrence de a. Les valeurs que peuvent prendre le deuxième paramètre sont : `exactement`, `au_moins`, `au_plus`.

`[a,b,c]` est la représentation de l'ensemble {a,b,c} sous forme d'une liste Prolog.

Figure 2 - Un exemple d'énoncé formalisé

Dans notre système, le formalisme logique ne sert qu'à *définir* le problème à l'aide de contraintes. Il n'est pas destiné à être *exécuté* par un interpréteur Prolog. On pourrait très bien cependant faire exécuter la requête `configuration_solution(X)` et on obtiendrait une liste de réponses donnant les valeurs possibles de X. On compterait alors le nombre de valeurs distinctes pour déterminer le cardinal cherché. Mais cette façon de procéder ne fournirait pas une démonstration analogue à celle qu'on attend d'un élève. Elle se contenterait d'énumérer exhaustivement toutes les listes de longueur 2 (prédicat `est_une_liste`) pour ne retenir finalement que celles qui vérifient les contraintes (prédicat `effectif_element`). Nous dirons que la définition de la figure 2 est *éliminatrice* car elle procède par essais et éliminations.

2.3. La définition constructive de l'ensemble à dénombrer

Un ensemble peut être décrit par plusieurs définitions équivalentes. Nous nous intéressons ici à celles que nous appelons *constructives*. Une définition constructive se présente sous forme de programme Prolog introduisant un prédicat unaire p. Mais elle se distingue d'une définition éliminatrice en ce que, dans l'exécution de la requête p(X), tout choix de valeur d'une variable conduit effectivement à la construction d'un élément X qui convient : rien de ce qui est proposé n'est éliminé.

Avant d'illustrer cela sur l'exemple de la figure 1, précisons d'abord comment nous représentons les "mots". Un mot de longueur 2 est vu comme une application de {1, 2} (appelé ensemble des "places") dans un ensemble de lettres. Une application f est identifiée à son graphe, c'est-à-dire à l'ensemble des couples (x, f(x)). Le mot "ca" est donc vu comme l'ensemble {(1, c), (2, a)}.

Dans l'exercice de la figure 1, un élément quelconque X de l'ensemble à dénombrer peut être construit en choisissant d'abord une place pour le a parmi les deux places, puis en choisissant une lettre dans {b, c} pour la place restante. On formalise cela par l'algorithme non déterministe suivant :

> construire un ensemble A formé d'un élément dans l'ensemble {1, 2};
> construire le complémentaire B de A dans {1, 2};
> construire une application F de A dans {a};
> construire une application G de B dans {b, c};
> construire la réunion X de F et de G.

Un exemple d'exécution de ce procédé donne :

A = {2}; F = {(2, a)}; B = {1}; G = {(1, c)}; X = {(1, c), (2, a)} ce qui est vu comme le mot "ca".

Un tel procédé peut être décrit par une règle Prolog, définissant un prédicat `un_a`, propriété caractéristique de l'ensemble à dénombrer (voir la figure 3). Le système utilise une quinzaine de prédicats prédéfinis permettant d'exprimer des définitions constructives, ceux de la figure 3 étant les suivants :

- `combinaison(N,R,S)` signifie que S est une partie à N éléments de l'ensemble R. N et R sont donnés et les valeurs de S sont engendrées.
- `complementaire(A,B,C)` signifie que C est le complémentaire de l'ensemble B dans l'ensemble A (B étant inclus dans A). A et B sont donnés et C est calculé.
- `application(A,B,F)` signifie que F est une application de l'ensemble A dans l'ensemble B. A et B sont donnés et les valeurs de F sont engendrées.
- `reunion([E1,…,En],X)` signifie que X est la réunion des ensembles E1, …, En. Ceux-ci sont donnés et X est calculé.

Un ensemble est représenté sous forme de liste Prolog et une application sous forme de liste de couples.

La figure 3 donne des exemples d'autres prédicats définissant les éléments des différents ensembles de mots de longueur 2 suivant le nombre d'occurrences de a.

L'exemple `au_moins_un_a` est défini par deux règles. Il s'agit d'une définition par *disjonction* : un mot contient au moins un a s'il en contient un ou s'il en contient deux.

```
un_a(X) :-
                    combinaison(1,[1,2],A),
                    complementaire(A,[1,2],B),
                    application(A,[a],F),
                    application(B,[b,c],G),
                    reunion([F,G],X).
aucun_a(X) :-
                    application([1,2],[b,c],X).
deux_a(X) :-
                    application([1,2],[a],X).
un_mot_quelconque(X) :-
                    application([1,2],[a,b,c],X).

au_moins_un_a(X) :- un_a(X).
au_moins_un_a(X) :- deux_a(X).
```

Figure 3 – Des définitions constructives

Les littéraux qui figurent dans la partie droite de chaque règle sont conçus et ordonnés de telle sorte que chacun n'introduise qu'une seule variable nouvelle. Lorsqu'une règle possède cette propriété, nous disons qu'elle est *échelonnée*. Dans la définition de `un_a`, par exemple, les variables introduites sont dans l'ordre A, B, F, G, X. C'est une des raisons pour lesquelles la méthode est qualifiée de *constructive* : le résultat X est formé par assemblage d'éléments introduits progressivement, de telle sorte qu'aucune contrainte nouvelle ne remette en cause le choix des éléments déjà introduits.

Deux interprétations : déclarative et procédurale

Une définition constructive peut être interprétée suivant deux points de vue, l'un déclaratif et l'autre procédural.

L'interprétation *déclarative* consiste à considérer le prédicat correspondant (`un_a` par exemple) comme une propriété caractéristique des éléments de l'ensemble à dénombrer, énoncée dans le formalisme Prolog de la logique formelle. Il s'agit alors d'une définition d'ensemble en compréhension.

L'interprétation *procédurale* consiste à considérer la règle Prolog comme un algorithme destiné à être exécuté. Lorsqu'on présente à un interpréteur Prolog la requête `un_a(X)`, il répond en affichant la séquence des réponses qu'il a engendrées, dans l'ordre dans lequel il les a engendrées :

```
N°1: X = [[1,a],[2,b]]
N°2: X = [[1,a],[2,c]]
N°3: X = [[2,a],[1,b]]
N°4: X = [[2,a],[1,c]]
```

```
No more solutions
```

A chaque définition constructive on peut ainsi associer d'une part un *ensemble* (l'ensemble défini par la propriété caractéristique) et d'autre part une *séquence* de réponses. Tout élément de l'ensemble figure au moins une fois dans la séquence mais certains éléments peuvent figurer plusieurs fois.

On sait que la possibilité d'utiliser la logique comme langage de programmation découle de l'existence de cette double perspective [Kowalski 74]. Nous l'exploitons ici à un niveau méta [Pitrat 90]: nous ne cherchons pas à faire exécuter l'algorithme, mais nous le considérons comme objet d'étude pour raisonner à son sujet et en déduire des propriétés de l'ensemble qu'il représente.

Multiplicité algorithmique, programme injectif

Nous appellerons *multiplicité algorithmique* du programme le nombre de réponses qu'il affiche. Dans l'exemple précédent ce nombre est 4 et dans ce cas il est égal au cardinal de l'ensemble à dénombrer, qui est {ab, ac, ba, ca}.

Nous dirons qu'un programme de construction est *injectif* si chaque valeur qu'il engendre n'apparaît qu'une seule fois dans la séquence des réponses, c'est-à-dire si sa multiplicité algorithmique est égale au cardinal de l'ensemble qu'il définit. Dans ce cas, nous dirons aussi que la définition constructive associée est injective.

Mais cela n'est pas toujours le cas. Considérons par exemple une autre définition de l'ensemble des mots de deux lettres construits avec l'alphabet {a, b, c} et contenant au moins un a : on place d'abord un a puis on complète avec une lettre quelconque. Cela se traduit par la définition `au_moins_un_a_version_2` de la figure 4.

```
au_moins_un_a_version_2(X) :-
                   combinaison(1,[1,2],A),
                   complementaire(A,[1,2],B),
                   application(A,[a],F),
                   application(B,[a,b,c],G),
                   reunion([F,G],X).
```

Figure 4 — Une définition constructive non injective

Ce prédicat est effectivement une propriété caractéristique de l'ensemble à dénombrer, mais l'interpréteur Prolog engendre la séquence de réponses suivante à la requête `au_moins_un_a_version_2(X)` :

```
N°1 X = [[1,a],[2,a]]
N°2 X = [[1,a],[2,b]]
N°3 X = [[1,a],[2,c]]
N°4 X = [[2,a],[1,a]]
N°5 X = [[2,a],[1,b]]
N°6 X = [[2,a],[1,c]]
```

```
No more solutions
```

La multiplicité algorithmique est 6, les réponses correspondant respectivement aux mots aa, ab, ac, aa, ba, ca. La réponse N°1 donne le même mot que la réponse N°4. L'ensemble à dénombrer est {aa, ab, ac, ba, ca} et son cardinal est 5 et non 6.

2.4. Le principe de la méthode

L'idée de la méthode constructive de dénombrement provient des propriétés suivantes, que nous avons vérifiées sur de nombreux exemples :
— il est possible sous certaines conditions de déterminer la multiplicité algorithmique du programme sans le faire exécuter (simplement en raisonnant sur sa structure);
— il est possible sous certaines conditions de prouver que la multiplicité algorithmique du programme est égale au cardinal de l'ensemble à dénombrer, là-aussi sans faire exécuter le programme.
Ces conditions sont vérifiées par exemple pour le programme `un_a`. On peut montrer que la multiplicité algorithmique est le produit des nombres de possibilités engendrées pour chaque littéral de la règle, soit $C_2^1 \times 1 \times 1^1 \times 2^1 = 4$. De plus, comme F et G sont des applications dont les ensembles d'arrivée sont constants et disjoints ({a} et {b,c}), on peut affirmer (grâce à un théorème que nous énonçons plus loin) que la multiplicité algorithmique est égale au cardinal de l'ensemble construit.
La figure 5 énonce la méthode, précédée des principes qui la fondent. Il s'agit en fait d'une formalisation du type de solution que peut fournir un élève et qui se présente sous la forme suivante : "On choisit d'abord une place pour le a : 2 possibilités; puis on remplit la place restante avec une autre lettre différente de a : 2 possibilités; donc en tout : $2 \times 2 = 4$ possibilités".

• *Propriété d'une définition constructive injective*
Soit D_c une définition constructive d'un ensemble E_c,
P_c le programme de construction associé à D_c,
m la multiplicité algorithmique de P_c.
Si P_c est injectif, alors $m = card(E_c)$

• *Solution constructive valide d'un problème de dénombrement*
Soit D_d une définition d'un ensemble à dénombrer E_d.
Un couple (D_c, k) est appelé une solution constructive valide du problème de dénombrement D_d si :
D_c est une définition constructive (on appelle E_c l'ensemble défini par D_c);
D_c est équivalente à D_d (c'est-à-dire $E_c = E_d$);
le programme P_c de construction associé à D_c est injectif;
k est égal à la multiplicité algorithmique de P_c .
Dans ce cas $k = card(E_d)$

• *Méthode constructive de dénombrement*
Pour déterminer le cardinal d'un ensemble E_d de définition D_d, il suffit d'en donner une définition constructive injective D_c et de calculer la multiplicité algorithmique m du programme associé P_c. (D_c, m) est alors une solution constructive valide du problème D_d.

Figure 5 — La méthode constructive de dénombrement

L'idée de base de la méthode est assez accessible et les élèves l'utilisent naturellement sur les problèmes simples (sans la formaliser). Le besoin d'une formalisation s'est fait sentir pour plusieurs raisons : d'abord pour pouvoir implémenter un système capable de résoudre des exercices de dénombrement, ensuite pour pouvoir analyser une solution produite par un élève, et enfin pour pouvoir engendrer de véritables démonstrations des solutions. Il est en effet significatif qu'une des difficultés habituellement ressenties devant ce genre de problèmes est qu'on n'est jamais vraiment sûr de l'exactitude de la solution [Antibi 88]. Cela tient à ce qu'on ne développe jamais entièrement les arguments mathématiques qui la sous-tendent. Une autre difficulté ressentie est qu'on ne sait jamais comment aborder ces problèmes. Les comportements des élèves ont été étudiées dans différents travaux de didactique ([Fischbein & Gazit 88], [Batanero et al 95], [Green 81]). Les indications que nous donnons dans cet article sur la façon d'appliquer la méthode constructive de dénombrement offrent justement différents moyens d'aborder ces problèmes.

2.5. Méthode constructive et constructions géométriques

Le type de problème abordé ici se retrouve dans les problèmes de constructions géométriques, où un objet inconnu (la figure à construire) est défini par des contraintes. Dans notre contexte nous retrouvons effectivement des éléments analogues à ceux qui ont été introduits dans différents travaux concernant les constructions géométriques : langage d'énoncé, langage de solution, programme de construction, vérification de correction [Schreck 93], [Desmoulins 94], [Allen et al 90].

3. L'APPLICATION DE LA METHODE PAR UN SYSTEME D'IA

Nous décrivons ici comment un système peut utiliser la méthode pour résoudre un exercice à partir d'une représentation mathématisée de l'énoncé. Nous n'abordons pas les problèmes posés par l'analyse du langage naturel et par la modélisation d'un énoncé concret.

3.1. La génération d'une définition constructive

La première étape consiste à analyser l'énoncé et à engendrer une définition équivalente constructive et injective. Il ne semble pas que cette tâche puisse être effectuée à l'aide d'un algorithme général applicable à tout énoncé. En tout cas, les solutions d'exercices données dans les livres ne donnent aucune indication sur cette étape et fournissent directement la définition constructive sans dire comment elle a été obtenue ni pourquoi elle est valide. Nous considérons cependant qu'il s'agit d'une étape essentielle et difficile et qu'il faut expliciter les connaissances mises en jeu.
Nous présentons ici deux méthodes susceptibles d'engendrer une définition constructive. La première (classification de problèmes) a été implémentée. L'autre est une direction de recherche en cours.

Classer les problèmes

L'observation de certains experts montre qu'ils connaissent des classes de problèmes classiques et qu'ils savent leur associer des schémas valides de définitions constructives et injectives. Ils procèdent en déterminant la classe du problème par l'analyse de son énoncé, puis instancient le schéma associé pour engendrer la définition constructive équivalente. Cette méthode de résolution par classement des problèmes a été mise en œuvre par N. Guin [Guin et al 95], sous forme d'un système d'IA à base de règles. Elle utilise une classification que nous avons élaborée, moins générale que celle de [Dubois 84], mais qui prend en compte plus de contraintes. Elle a été appliquée avec succès à 60 exercices et permet de résoudre la majorité des exercices posés dans les classes de terminale scientifique. L'intérêt de cette méthode est qu'elle garantit la validité des solutions lorsqu'elle est applicable.

Analyser le résultat supposé construit

La méthode usuelle pour les constructions géométriques est de raisonner sur la figure "supposée construite" pour en extraire des propriétés nécessaires, d'utiliser ces propriétés comme base d'une construction puis de montrer que ces propriétés sont suffisantes.

On peut appliquer une méthode analogue dans l'exemple de la figure 1 pour retrouver la définition de la figure 3. Pour cela, à partir de la définition, on déduit des propriétés d'une solution quelconque X. Si X comporte exactement une occurrence de a, alors l'ensemble A des antécédents de a (les positions des occurrences de a) est une partie à un élément de l'ensemble des places {1, 2}. De même, l'ensemble B des antécédents de {b, c} est le complémentaire de A dans {1, 2}. Donc X est la réunion de sa restriction F à A et de sa restriction G à B. Ces propriétés permettent de reconstituer la définition constructive correspondant au prédicat `un_a` de la figure 3. Deux difficultés principales se présentent dans cette démarche : d'abord identifier les propriétés et les éléments intéressants sans se laisser submerger par l'explosion combinatoire des possibilités et ensuite engendrer une construction à partir des éléments identifiés. Dans l'exemple précédent il faut avoir l'idée d'introduire A, B, F et G et se rendre compte que ces éléments suffisent pour construire X. On peut utiliser pour cela des heuristiques comme : si la configuration est une liste qui possède exactement n occurrences d'une valeur v, alors introduire comme objet intéressant l'ensemble des positions où figure v.

3.2. La détermination de la multiplicité algorithmique

La deuxième étape consiste à déterminer la multiplicité algorithmique du programme de construction. Le programme étant formé de règles et les règles de littéraux, la multiplicité s'obtiendra à partir des calculs partiels concernant ces éléments.

Nombre de valeurs engendrées pour une inconnue

Chaque littéral de la partie droite d'une règle échelonnée introduit une nouvelle variable, considérée comme l'*inconnue* du littéral. Au moment de la résolution du littéral, seule cette variable n'a pas encore reçu de valeur. Le problème est de déterminer le nombre de valeurs qu'elle peut prendre. C'est ce qu'on appelle la *multiplicité* du littéral dans le programme. Pour la calculer, on utilise des connaissances associées au prédicat figurant dans le littéral.

Certains prédicats sont déterministes, c'est-à-dire qu'ils n'engendrent à chaque fois qu'une possibilité pour la variable inconnue. C'est le cas pour le complémentaire, le produit cartésien, la réunion, les intervalles. Pour chacun des autres prédicats, on sait associer une formule permettant de calculer le nombre de valeurs engendrées pour l'inconnue en fonction des données. Par exemple, pour le prédicat `combinaison(P,E,A)` où A est l'inconnue, si l'on appelle N le cardinal de l'ensemble E, alors le nombre de valeurs engendrées pour A est C_N^P, où C dénote le coefficient binomial. Ces formules se fondent sur les théorèmes de dénombrement connus par les élèves. Elles permettent de traiter les listes, les arrangements, les combinaisons, les applications, les injections.

Détermination de cardinaux d'ensembles

Pour appliquer les formules de calcul ci-dessus il faut pouvoir déterminer les cardinaux de certains ensembles. Par exemple, pour déterminer le nombre de possibilités pour A dans `combinaison(P,E,A)`, on doit pouvoir calculer le cardinal de l'ensemble E. Or les ensembles peuvent être définis non seulement par énumération de leurs éléments, mais aussi par des opérations ensemblistes (réunion, complémentaire, intervalle). Il est alors nécessaire d'appliquer des connaissances de dénombrement, comme : "si deux ensembles sont disjoints, le cardinal de leur réunion est égal à la somme de leurs cardinaux".
Ces connaissances se présentent souvent sous forme conditionnelle et il est donc nécessaire de savoir déterminer si les conditions d'application sont remplies ou non. Lorsque deux ensembles sont définis par une composition complexe d'opérations, déterminer s'ils sont disjoints ou si l'un est inclus dans l'autre n'est pas immédiat s'il n'est pas envisageable d'énumérer leurs éléments. On peut seulement se reposer sur un ensemble de conditions suffisantes sans être certain de pouvoir résoudre tous les cas. On utilise par exemple des théorèmes comme : "si A est inclus dans B alors pour que C soit disjoint de A il suffit qu'il soit disjoint de B" (la condition n'étant pas nécessaire). Nous avons implémenté dans le système une quinzaine de règles de ce type qui permettent de traiter les cas courants.

Nombre de réponses engendrées par le programme

La multiplicité d'une règle échelonnée est le produit des multiplicités de ses littéraux. Cela résulte d'un théorème de dénombrement appelé "principe multiplicatif", parfois énoncé explicitement dans les livres de classe, mais bien souvent laissé implicite ou même parfois confondu avec le dénombrement des produits cartésiens.
Lorsque le programme est formé d'une seule règle, on obtient ainsi la multiplicité du programme. Lorsque le programme correspond à un prédicat défini par plusieurs règles (disjonction), sa multiplicité est la somme des multiplicités de chaque règle.

4. L'ETUDE DE LA VALIDITE D'UNE PROPOSITION DE SOLUTION

Examinons maintenant comment le système peut étudier la validité d'une proposition de solution constituée d'une définition constructive D_c et d'un nombre k (nous utilisons dans cette section les notations de la figure 5) pour un problème de dénombrement défini par D_d. Une proposition de solution peut provenir de deux sources : ou bien du système lui-même, lorsqu'il cherche à résoudre un problème, ou bien d'un utilisateur (un élève par exemple).
Lorsque la solution est engendrée par le système, celui-ci peut contrôler la validité en s'imposant de n'utiliser que des procédés de génération qui garantissent que chaque étape est valide. C'est le cas par exemple pour la méthode de résolution par classement de problèmes déjà mise en œuvre. Cependant, d'autres méthodes pourraient être seulement

heuristiques et aboutiraient à des propositions de solution dont il faudrait étudier la validité.
Lorsque la solution est proposée par un élève, le système doit vérifier explicitement les conditions de validité.
L'étude de la validité est un problème de démonstration de propriétés pour lequel il n'existe pas à notre connaissance d'algorithme de décision utilisable. Énumérer les éléments de tous les ensembles mis en jeu est théoriquement possible, mais pratiquement inacceptable. Nous indiquons ci-dessous des méthodes qui permettent dans certains cas de décider si une proposition de solution est valide ou non.

4.1. Ensemble construit et ensemble à dénombrer

Prouver qu'une définition est constructive
Les outils fournis à l'utilisateur pour introduire une proposition de solution (langage spécialisé) ne lui permettent d'introduire que des définitions constructives, de par la syntaxe même de la représentation, qui est contrôlée à la saisie.

Prouver l'équivalence de la construction D_c et de l'énoncé D_d
Il faut démontrer que l'ensemble construit E_c est égal à l'ensemble à dénombrer E_d. Certaines méthodes permettent de prouver directement l'égalité des deux ensembles, d'autres exigent de séparer l'étude en deux inclusions réciproques.
• *E_c est-il égal à E_d ?* — On peut s'appuyer sur la classification des problèmes déjà utilisée pour engendrer une définition constructive à partir de l'énoncé. En effet, on a vu qu'à chaque classe était associé un schéma de construction. Il est alors possible d'analyser une construction pour voir si elle correspond à un schéma connu. Il s'agit d'un problème d'appariement de structures (pattern matching). Cet appariement doit tenir compte du fait qu'un même schéma peut être réalisé sous différentes formes en raison de possibilités multiples d'ordonnancement des éléments. On compare ensuite la classe ainsi obtenue à la classe reconnue à partir de l'énoncé. Si elles sont identiques et décrites par les mêmes paramètres, alors E_c est égal à E_d. Nous avons donné au système les connaissances lui permettant de reconnaître la classe d'un problème à partir de la définition constructive, ce qui lui permet de démontrer l'égalité de E_c et de E_d pour les constructions usuelles correctes. Cette méthode peut cependant échouer si la construction ne correspond pas à un schéma connu. Les méthodes suivantes (non encore implémentées) peuvent servir à résoudre certains autres cas.

• *E_c est-il inclus dans E_d ?* — Il faut montrer que toute configuration construite vérifie bien les contraintes de l'énoncé. On peut associer à chaque type de contrainte des méthodes de vérification dépendant des opérateurs utilisés dans la construction. Par exemple : "pour prouver que la réunion de deux configurations F et G comporte exactement n occurrences de la valeur a, déterminer les nombres d'occurrences respectifs p et q de a dans F et G, vérifier que F et G sont disjoints et que n = p + q".

• *E_d est-il inclus dans E_c ?* — Il faut s'assurer que la construction engendre effectivement tous les éléments de l'ensemble à dénombrer, ce qui est plus difficile. On peut essayer pour cela d'identifier les variables apparaissant dans la construction à des "éléments intéressants" associés à l'énoncé et introduits de manière heuristique. Reprenons l'exemple `un_a` de la figure 3. On peut identifier A à l'ensemble des antécédents de a, B à l'ensemble des antécédents de {b, c}, F à la restriction de X à A et et G à la restriction de X à B. Cela prouve que l'ensemble à dénombrer est bien inclus dans l'ensemble construit : pour toute configuration X vérifiant les contraintes, on peut trouver A, B, F et G vérifiant la définition constructive et donc X est bien engendré par la construction.

Prouver la non équivalence de la construction et de l'énoncé

La solution d'un élève n'étant pas toujours valide, il faut aussi des connaissances pour prouver que E_c n'est pas égal à E_d. On essaye de le faire lorsqu'on n'a pas réussi à prouver l'égalité.

Il arrive que les solutions des élèves ne soient pas valides parce qu'elles sont trop contraintes. Par exemple, au lieu d'engendrer tous les mots qui contiennent exactement un a, ils engendrent seulement les mots qui contiennent exactement un a et qui commencent par a. On obtiendrait pour cet exemple la définition suivante (à comparer avec la définition correcte de la figure 3) :

```
un_a(X) :-
       application([1],[a],F),
       application([2],[b,c],G),
       reunion([F,G],X).
```

Le système peut calculer la multiplicité de ce programme : elle est égale à 2 (= $1^1 2^1$). D'autre part, il sait résoudre le problème (donné sous la forme D_d de la figure 2) et trouve que le cardinal de l'ensemble à dénombrer est 4. 4 étant supérieur à 2, on peut affirmer que la solution de l'élève n'est pas valide. En revanche, on ne sait pas expliquer exactement où est l'erreur. On peut affiner l'analyse en prouvant que l'ensemble E_c est bien inclus dans l'ensemble E_d (tout X engendré vérifie les contraintes) et on peut alors dire à l'élève que sa solution est trop contrainte.

4.2. Injectivité du programme

Prouver que le programme est injectif

Là encore, cette propriété ne semble pas pouvoir être vérifiée par un algorithme général. On utilise des conditions suffisantes comme celles-ci : "Si M et N sont deux ensembles disjoints, et si Y est un ensemble inclus dans le produit cartésien M×N alors l'ensemble X des ensembles de la forme F∪G pour tout (F,G) dans Y a le même cardinal que Y". Ce théorème est essentiel pour démontrer la validité de nombreuses solutions proposées dans les livres, mais il n'est jamais énoncé explicitement ni même suggéré.
On peut l'appliquer au programme `un_a` de la figure 3 : X est obtenu comme réunion de F et G. On peut montrer que F est inclus dans le produit cartésien {1,2}×{a} et que G est inclus dans {1,2}×{b,c}, produits qui sont disjoints car leurs deuxièmes composantes le sont. Le théorème énoncé ci-dessus prouve alors qu'il y a autant de couples (F,G) que de valeurs de X, ce qui permet de démontrer que le programme engendre chaque valeur de X une fois et une seule.
La méthode mise en œuvre actuellement dans le système utilise ce type de théorèmes en s'appuyant de plus sur la classification des problèmes déjà utilisée à plusieurs reprises. En effet, lorsqu'on a reconnu un schéma classique de construction, les théorèmes à appliquer et les façons de les appliquer sont connus à l'avance et en petit nombre. A chaque schéma de construction est ainsi associé un schéma de démonstration qui permet de vérifier si le programme est injectif.

Prouver que le programme n'est pas injectif

Si E_c est inclus dans E_d et si la multiplicité algorithmique m est supérieure à card(E_d), alors le programme n'est pas injectif. C'est le cas du programme de la figure 4 `au_moins_un_a_version_2` , qui engendre deux fois le mot "aa". On peut prouver que toute configuration X construite contient bien au moins une occurrence de a et que la multiplicité algorithmique est 6 alors que le cardinal est 5.

5. CONCLUSION

Nous avons introduit un cadre conceptuel permettant d'étudier les problèmes de dénombrement. Dans ce cadre, nous avons défini une méthode de résolution, la "méthode constructive de dénombrement". Elle consiste à élaborer une définition de l'ensemble à dénombrer qui puisse être interprétée comme un algorithme de génération des éléments de cet ensemble, puis à raisonner sur cet algorithme au niveau méta, sans l'exécuter.
Nous l'avons mise en œuvre dans un système capable de résoudre des exercices du niveau de la classe de terminale scientifique et pouvant également recevoir des propositions de solutions et étudier leur validité. Actuellement, le système s'adresse surtout à un utilisateur averti, en raison de son langage technique spécialisé. Nous travaillons maintenant à le rendre accessible à un élève, tout en conservant une base mathématique solide et un contrat

didactique clair et précis [Allen et al 92]. Pour cela, il est nécessaire de présenter les concepts sous une forme moins formalisée et au travers d'une interface simple à utiliser. Nous avons déjà commencé à explorer cette voie en réalisant des maquettes d'interface correspondant à différents styles d'expression [Duma et al 95].

L'intérêt pédagogique de la méthode tient d'abord à ce qu'elle fournit des outils permettant aux élèves de dépasser les difficultés qu'ils ressentent habituellement face aux problèmes de dénombrement. Ensuite, elle illustre la possibilité de raisonner au niveau méta. Et enfin, elle constitue un moyen d'accéder par le biais des dénombrements à d'autres domaines : logique, structures de données, algorithmique.

6. REFERENCES BIBLIOGRAPHIQUES

[Allen et al 90] Allen R., Nicolas P., Trilling L., Figure Correctness in an Expert System for Teaching Geometry. *Proc. of the eigth biennal conference of the canadian society for computational studies of intelligence*, Ottawa, mai 1990, pp. 154-160.

[Allen et al 92] Allen R., Desmoulins C., Trilling L., Tuteurs intelligents et intelligence artificielle : problèmes posés en construction de figures géométriques. *Proc. of the second international conference ITS '92*, Montréal, Springer Verlag, 1992, p. 325-334.

[Antibi 88] Antibi A., *Étude sur l'enseignement de méthodes de démonstration. Enseignement de la notion de limite : réflexions, propositions*. Thèse d'état, Toulouse, juin 1988.

[Batanero et al 95] Batanero C., Godino J.D., Navarro-Pelayo V., The use of implicative and correspondence analysis for assessing pupils' combinatorial reasoning. *R. Gras (ed): Actes du Colloque Méthodes d'analyses statistiques multidimensionnelles en Didactique des Mathématiques* IRMAR Rennes et IRESTE Nantes 1995, p. 245-256.

[Desmoulins 94] Desmoulins C. *Etude et réalisation d'un système tuteur pour la construction de figures géométriques*, Thèse de l'Université Joseph Fourier, Grenoble 1994.

[Dubois 84] Dubois J.-G., Une systématique des configurations combinatoires simples. *Educational studies in Mathematics 15* (1984), p. 37-57.

[Duma et al 95] Duma J., Giroire H., Tisseau G., Le Calvez F., Urtasun M., Mise en évidence de styles de résolution. Évolution de l'interface dans le projet Combien?. *Guin D., Nicaud J.-F., Py D. (eds) Environnements interactifs d'apprentissage avec ordinateur, tome 2.* Paris, Eyrolles 1995, p. 245-256.

[Fischbein & Gazit 88] Fischbein E., Gazit A., The combinatorial solving capacity in children and adolescents. *Zentralblatt für Didaktik der Mathematik, 5*, 1988, p. 193-198.

[Green 81] Green D.R., *Probability concepts in high school pupils aged 11-16 years.* Ph. D. Thesis. Loughborough University, 1981.

[Guin et al 95] Guin N., Giroire H., Tisseau G., Le classement de problèmes : une méthode de résolution pour le module expert d'un EIAO. Application aux problèmes de dénombrement. *Guin D., Nicaud J.-F., Py D. (eds) Environnements interactifs d'apprentissage avec ordinateur, tome 2.* Paris, Eyrolles 1995, p. 113-124.

[Kowalski 74] Kowalski R., Predicate logic as a programming language, *Proc. IFIP Congress 1974*, Stockholm, North Holland, 1974.

[Nicolas 89] Nicolas P., *Construction et vérification de figures géométriques dans le système MENTONIEZH*, Thèse de l'Université de Rennes, 1989.

[Pitrat 90] Pitrat J., *Métaconnaissance. Futur de l'intelligence artificielle*. Paris, Hermès, 1990.

[Schreck 93] Schreck P. *Automatisation des constructions géométriques à la règle et au compas,* Thèse de l'Université Louis Pasteur, Strasbourg, 1993.

[Tisseau et al 93] Tisseau G., Duma J., Giroire H., Le Calvez F., Urtasun M., *Représentation des problèmes de dénombrement dans le projet Combien?*, rapport interne Laforia 93/39, novembre 1993.

[Tisseau et al 94] Tisseau G., Duma J., Giroire H., Le Calvez F., Urtasun M., Un langage de descriptions fonctionnelles, relationnelles et ensemblistes pour des problèmes de dénombrement. *Actes du 9ème congrès Reconnaissance des Formes et Intelligence Artificielle (Paris 11-14 janvier 1994), volume 2*, Paris, AFCET 1994 (ISBN 2-903677129-X), p. 739-744.