Institut National Polytechnique de Lorraine

Centre de Recherche en Informatique de Nancy
C.R.I.N.

SPÉCIFICATION ALGÉBRIQUE À L'AIDE DE TYPES ABSTRAITS DU SYSTÈME EXPRIM
(EXpert Pour la Recherche d'IMages)

THESE

soutenue publiquement le **3 mars 1984**

À l'Institut National Polytechnique de Lorraine

pour l'obtention du grade de
DOCTEUR DE 3ème CYCLE EN INFORMATIQUE
par
Mekki BOUKAKIOU
devant la Commission d'Examen :

Daniel COULON
Marion CREHANGE
Pierre LESCANNE
Roger MOHR
Jean-Luc REMY
Martin WIRSING
Invité : *Henry HUDRISIER*

Institut National Polytechnique de Lorraine

Centre de Recherche en Informatique de Nancy
C.R.I.N.

SPÉCIFICATION ALGÉBRIQUE À L'AIDE DE TYPES ABSTRAITS DU SYSTÈME EXPRIM
(EXPERT POUR LA RECHERCHE D'IMAGES)

THESE

soutenue publiquement le **3 mars 1984**

À l'Institut National Polytechnique de Lorraine

pour l'obtention du grade de
DOCTEUR DE 3ème CYCLE EN INFORMATIQUE
par
Mekki BOUKAKIOU
devant la Commission d'Examen :

Président : . . . . . *Daniel COULON*
Examinateurs : . . . . . *Marion CREHANGE*
Pierre LESCANNE
Roger MOHR
Jean-Luc REMY
Martin WIRSING
Invité : . . . . . *Henry HUDRISIER*

A mes parents,

Je remercie Marion Créhange d'avoir accepté de diriger ma recherche ; je lui suis reconnaissant pour son dévouement à mon égard, son aide, ses conseils et ses encouragements.

Je remercie Martin Wirsing pour avoir bien voulu m'accueillir quelques jours à l'Université de PASSAU et pour les conseils qu'il m'a donnés durant ce séjour.

Je remercie Roger Mohr et Pierre Lescanne d'avoir eu la gentillesse d'accepter de faire partie du jury et pour les critiques qu'ils m'ont faites au sujet de ce travail.

Je remercie Daniel Coulon pour l'honneur qu'il me fait en présidant ce jury.

Je tiens à remercier vivement Jean-Luc Rémy qui m'a fait bénéficier de ses connaissances sur les types abstraits. Tout au long des nombreuses discussions que j'ai pu avoir avec lui, il n'a cessé de m'encourager et surtout de m'éclairer sur de nombreux points.

Je remercie Nadine Gautier pour la frappe qu'elle a effectuée.

Tous mes remerciements à mes collègues de bureau, à toutes les personnes qui m'ont aidé et soutenu tout au long de ce travail, ainsi qu'à la DGRST pour son soutien financier.

Je remercie également l'Agence SYGMA, et en particulier Henri Hudrisier, qui ont été à l'origine du sujet, de m'avoir gentiment accueilli et conseillé et de m'avoir mieux fait percevoir le fonctionnement d'une agence de presse.

Hantao Zhang m'a permis de tester la correction de certains types à l'aide du système REVE 4. Je le remercie vivement pour sa collaboration.

S O M M A I R E

INTRODUCTION

INTRODUCTION

Toute personne voulant illustrer un texte ou appuyer une idée peut avoir besoin de choisir une ou des images dans un important fonds d'images.

Un scientifique qui rédige un article utilisera une image d'illustration, non pas seulement pour argumenter son article, mais aussi pour faire rêver un peu le lecteur, le distraire et rendre la lecture plus attrayante.

C'est dire combien le documentaliste audiovisuel, chargé de répondre aux besoins de cet auteur scientifique, devra pouvoir à la fois se situer dans le champ du discours scientifique, stricto sensu, donc donner des images très précises répondant à des concepts scientifiques souvent très fins, et aussi illustrer moins "techniquement" le discours.

Suivant sa rigueur ou celle de son journal par rapport aux images, le journaliste cherchera une image précise et signifiante d'un évènement ou une simple évocation d'une ambiance. En résumé les iconothèques (bibliothèques d'images) oscillent entre deux extrêmes :

- pouvoir proposer des images répondant à quelques mots-clés précis ; dans ce cas on peut prévoir qu'au plus quelques images répondront à ces critères ;

- pouvoir proposer des images répondant à des mots-clés extrêmement larges ou répondant de façon très large à des mots-clés : par exemple proposer des photos du Fuji Yama pour illustrer la fabrication d'un micro-processeur au Japon.

Devant la formidable inflation d'images que connaît notre époque il apparaît de plus en plus nécessaire d'automatiser le travail du documentaliste de l'image. Cette automatisation devra prendre en compte la saisie et le stockage d'images, ainsi que leur recherche et leur visualisation rapide.

Le but de cette thèse est la spécification, à l'aide de types abstraits algèbriques, d'un système permettant la recherche d'images par le biais d'une base contenant leurs descriptions, système bati autour d'un système expert simulant l'expertise d'un documentaliste de l'image.

Le chapitre A est consacré à la présentation du projet EXPRIM (EXPert pour la Recherche d'IMages), à la mise en évidence de la spécificité de l'information image, à la description informelle du fonctionnement du système, et à la présentation de l'architecture du système avec ses composantes logicielles et matérielles. En particulier nous présentons le poste de manipulation d'images (poste de travail) et ses diverses fonctions.

Au chapitre B nous préciserons les outils utilisés pour la spécification du système (les types abstraits algébriques) et nous tenterons également de justifier ce choix.

Le chapitre C présente une spécification algébrique de la base descriptive, des types manipulés dans cette base et des requêtes adressées à la base, ainsi que de l'adéquation entre descriptions et requêtes, et aussi (brièvement) de l'échantillonnage.

Le chapitre D présente une approche formelle des Systèmes Experts. On y décrit de façon algébrique différents modes de fonctionnement ainsi que les types d'informations manipulées par les systèmes experts.

Au chapitre E nous donnons une brève description du langage CLU permettant d'implanter des types abstraits.

Nous essaierons également de dégager, dans la conclusion, en quoi les spécifications algébriques que nous présentons tout au long de cette thèse seront utiles pour la réalisation du système.

CHAPITRE A

LE PROJET EXPRIM

CHAPITRE A :

LE PROJET EXPRIM

A-I) LE PROJET :<AIT 82>, <AIT 83>

A l'origine du projet EXPRIM se trouvent les problèmes que connaît une agence de presse devant gérer un important fonds d'images et désirant automatiser ses activités.

EXPRIM sera un système intéractif permettant l'interrogation alternée avec renforcement mutuel d'une base d'images (sans recherche directe sur les images elles-mêmes) et d'une base de textes (base descriptive) contenant des éléments descriptifs des images.

Le système que nous concevons pourra non seulement être utilisé par un iconographe (documentaliste images) mais également par un usager non spécialiste et désirant cheminer dans un fonds d'images dont il ne connaît pas la teneur à priori.

L'information image présente certaines spécificités qui conditionnent très fortement le système.

A-II) SPECIFICITES DE L'INFORMATION IMAGE : <HUD 82>

Tant que la documentation a pour mission de traiter une information textuelle, le problème est, sinon simple, du moins largement décrit et fait l'objet d'une tradition : le savoir faire des bibliothèques. Le contenu d'un livre est généralement représenté par un texte de même que son titre et que son résumé éventuel: il n'y a pas de saut qualitatif grave entre le texte et son analyse.

Dans le traitement documentaire classique il y a compression de l'information mais les modes de perception de cette information restent les mêmes.
Pour l'image, au contraire, le saut qualitatif est certain : l'image et le texte ne sont pas de même nature et il s'agit de les faire cohabiter dans un même système documentaire.

La description d'une image passe par une phase d'analyse et il est d'usage de situer l'image à deux niveaux de signification : le "denoté" et le "connoté". Le premier niveau, le "denoté", a traditionnellement pour valeur de décrire les objets présents sur une image de manière apparemment réaliste : un chat sera toujours un chat ... Le "connoté" c'est tout le reste :

- thème qu'illustre l'image

- esthétique (images de contre-jour, couleurs dominantes dans l'image....)

- registre (image à sensation, pédagogique, romantique ....)

- chronologie

.
.
.

(Le chat peut incarner alors le thème de la douceur, de la paresse, etc...).

A-II-a) Aspects multi-dimensionnels de l'information image

Une image peut être perçue comme une information multi-dimensionnelle et chacun des aspects ci-dessus (thème, esthétique, registre, chronologie ....) peut être considéré comme une dimension de cette information.

A-II-b) Richesse de l'information image

L'information recélée par une image peut être d'une richesse très grande et non toujours connue a priori : imaginez une foule de skieurs de fond s'élancant, nul ne peut dire que tel ou tel deviendra prix Nobel ou ministre etc....).De plus la densité de cette richesse peut être très inégale d'un endroit de l'image à l'autre et l'information peut aussi être très inégalement mise en valeur (gros plan, plan éloigné ...)

A-II-c) "Bruit" et "silences" ; vision rapide des images:

La très grande richesse de l'information contenue dans une image par rapport à celle contenue dans son indexation conduira très souvent à l'introduction de "bruit" et de "silences" dans le résultat d'une recherche et il convient de faire les remarques suivantes :

- en ce qui concerne les bruits :

a) Il est admis que le "bruit" est moins important dans une iconothèque automatisée (notammemt du fait de la présentation rapide et intéractive de l'image elle-même) que dans une bibliothèque. La perception globale d'une image fixe est très rapide comparativement à la lecture d'un texte et ceci conduit à une élimination rapide du bruit.

b) Dans le cas d'une agence de presse, dont le but évident est de fournir rapidement un nombre important de références, le bruit importe peu, et il serait même dans une certaine mesure souhaitable,pouvant suggérer des idées nouvelles ou même,pourquoi pas, répondre à un autre aspect de la recherche non exprimé par le documentaliste.

- en ce qui concerne les "silences" :

Lors de la description de l'image certaines informations peuvent être omises et ceci peut conduire à des silences lors d'une phase de recherche concernant ces informations.
Cependant il n'est pas exclu de pouvoir retrouver,par une recherche indirecte, des images non décrites par l'information passée sous "silence" mais contenant cette information.Celle-ci revêt une très grande importance dans notre système Nous en verrons un exemple au paragraphe IV.

A-II-d) Vision globale d'un ensemble d'images - Inférence

Contrairement aux textes, la vision globale d'un ensemble d'images est possible et elle peut être très "suggestive". Cette vision peut être à l'origine d'idées nouvelles de recherche et on procède ainsi par inférence visuelle.

Nous pouvons également remarquer que :

. Les informations textuelles devant représenter une image doivent être facilement modifiables pour les raisons suivantes :

- La très grande richesse de l'information recelée par une image ne pourra pas être complètement reproduite.

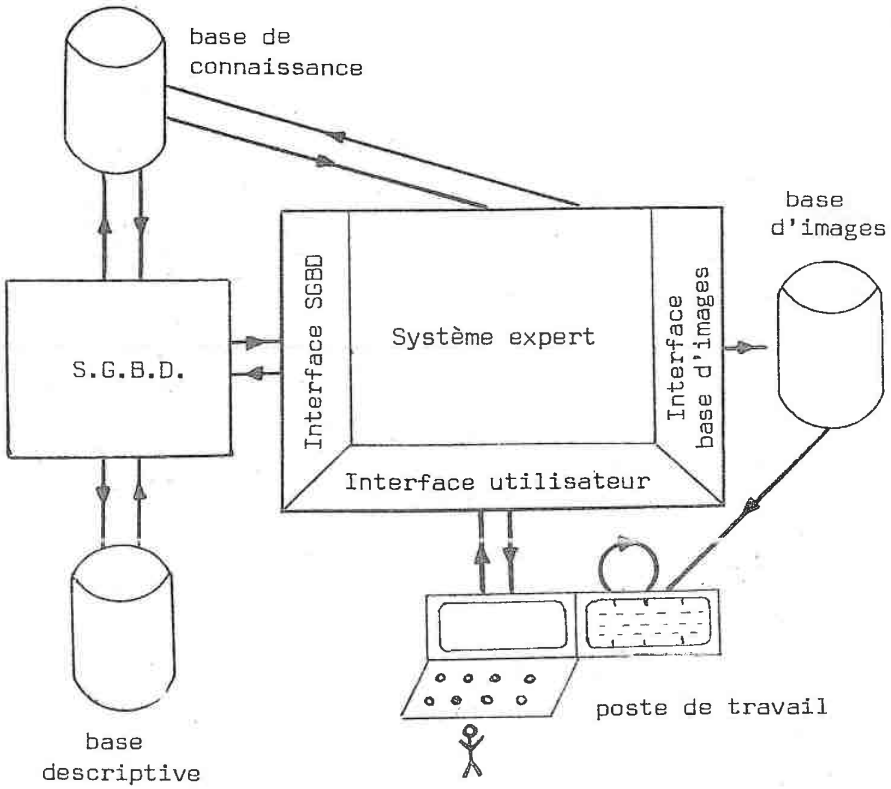
- L'information "intéressante" peut varier dans le temps.

- La description d'une image sera subjective.

. Plus la construction documentaire avec des mots-clés devient sophistiquée, plus elle rétrécit le champ de réponse de la collection et surtout, plus elle devient longue à construire.
Il faudra s'appliquer à trouver un équilibre entre le langage textuel et le langage image dans une banque de données iconographiques automatisée.

A-III) ARCHITECTURE DU SYSTEME

L'architecture du système EXPRIM sera la suivante :



Les parties a) et b) existent déjà. Les images sont stockées sur vidéodisque. Une description plus détaillée de ces différentes composantes du système est donnée ci-dessous :

a) Le poste de travail :

Le poste de travail est composé :

- d'un clavier
- d'un écran alphanumérique
- d'un écran image avec deux modes d'utilisation :
  - mode normal : L'image est visualisée plein écran.
  - mode damier : L'écran est divisé en n * m fenêtres. Après sélection d'un certain nombre d'images, l'utilisateur pourra fabriquer des "tas" images et avoir ainsi un accès comparatif aux images. Le mode de visualisation offre de nombreuses possibilités :
    - de choisir, au début, parmi des images éloignées les unes des autres.
    - de choisir ensuite plus facilement entre des images proches les unes des autres.
    - d'avoir une vue globale d'un ensemble d'images.
    - etc ...

b) La base d'images:

Les images pourront être stockées sur différentes mémoires :

- disque numérique
- disque optique numérique
- vidéodisque

.
.
.

Dans tous les cas chaque image sera identifiée par son numéro de référence et un interface d'accès aux images par leur référence sera nécessaire.

Les parties a) et b) sont constituées dans la version actuelle:

-d'un vidéodisque contenant 54 000 images relié à un moniteur T.V.

-d'un clavier

-d'un micro-ordinateur pilotant l'ensemble et sur lequel est implanté le logiciel de manipulation et de sélection d'images.

Les requêtes qui peuvent être actuellement adressées au système sont formées d'une suite de mots-clés, separés par des ET.Ces requêtes sont adressées à une base textuelle implantée sur un Mini 6.

Les images sélectionnées par une requête peuvent être visualisées suivant différents modes:

-défilement sur l'écran damier 16 par 16 par remplissage horizontal ou vertical.Ce défilement des images peut s'effectuer soit dans l'ordre où elles se trouvent physiquement sur le disque soit avec un pas de "n".

-possibilités de constituer des "tas" d'images.Ces "tas" ont une structure de pile sur laquelle on peut mettre une image "couvercle";on peut constituer des sous-tas,changer l'ordre des "tas", ou supprimer des images et des "tas";on peut également consulter localement un tas sur une seule zone de damier en le faisant "tourner".

c) La base descriptive :

Contiendra les informations alphanumériques décrivant les images

d) e) f) SGBD :

La base descriptive sera gérée par un SGBD qui pourra être ;

- Un SGBD classique, par exemple relationnel
- Une machine base de données
- Un système documentaire.

g) h) Le système expert et la base de connaissance

Le système expert est l'élémemt principal du système EXPRIM. Il utilisera une base de connaissance contenant :

- des règles de production pouvant aider le système à générer des requêtes ou à faire évoluer une requête initiale, à guider ou exploiter les phases de manipulation visuelle des images.

- des informations sur le vocabulaire utilisé pour la description des images.

- des informations sur le domaine qu'illustrent les images (sport, géographie, actualités ....)

A-IV) FONCTIONNEMENT DU SYSTEME

Plusieurs modes de fonctionnement sont prévus dans EXPRIM. L'utilisateur peut adresser des requêtes et le système cherchera les références des images satisfaisant les requêtes.
Cependant les requêtes peuvent être amenées à évoluer en subissant de multiples remodelages. Le remodelage peut aller dans deux sens :

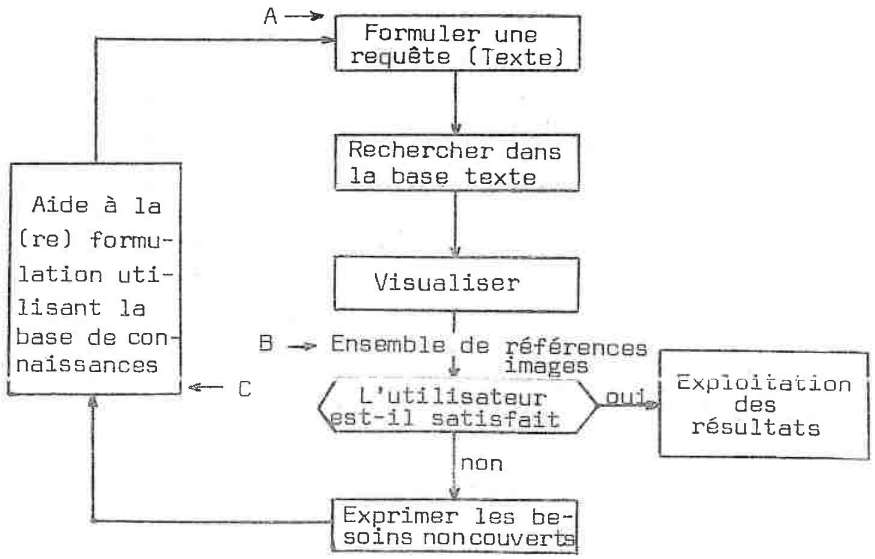
- Si le nombre d'images selectionnées est trop important, le temps de visualisation serait trop long . Le système, pourra avec l'aide de l'utilisateur, restreindre la requête ou fabriquer un échantillonnage des images sélectionnées et n'en visualiser que certaines.

- Si le nombre d'images selectionnées est peu important, il pourra au contraire élargir la requête et ainsi proposer plus de choix à l'utilisateur. Il faut cependant prendre garde de ne pas trop "trahir" l'objectif premier de l'utilisateur.

La requête peut également évoluer après la visualisation des images si l'utilisateur n'est pas satisfait. Le système devra faciliter l'expression des besoins non couverts par la requête initiale et que seule la visualisation aura permis à l'utilisateur de mettre en évidence.

On peut même envisager, dans certains cas, d'inférer une requête par des images tirées absolument au hasard dans le fonds. Ceci facilite au maximum la démarche "associative" du documentaliste. En effet, les questions posées en termes linguistiques sont parfois fort imprécises ou éloignées de la véritable idée visuelle du documemtaliste. Lui fournir une photographie non pertinente ou non demandée est une façon utile de recentrer la question vers une direction de recherche plus intéressante.

Le schéma ci-dessous nous indique les différents modes de fonctionnement du système.



Notre but final est de faire d'EXPRIM un véritable système expert pour documentaliste-image dans lequel interviendront deux niveaux d'expertise :

- l'expertise propre au documemtaliste-image.
- l'expertise liée au domaine que recouvrent les images.

EXEMPLE DE SCENARIO D'UTILISATION DE EXPRIM

Un journaliste désire illustrer le fait suivant :

Monsieur X, inconnu du public, devient ministre.

Etapes :

I Requête : "Photos de Monsieur X ?"

II . Je ne connais pas Monsieur X (n'est pas dans la base)
. Donne-moi quelques détails sur lui.

Monsieur X est adjoint au maire de Lorient.

. Je trouve 105 photos d'évènements officiels à Lorient.
. Veux-tu que je les visualise - toutes ?
- un échantillon ?

Toutes

PHASE DE VISUALISATION au cours de laquelle l'utilisateur indique les photos qu'il choisit (les 31 ou figure Mr X)

III . Je remarque que, pour plusieurs des images choisies, la description contient le nom du navigateur Alain Bombard.
. Veux-tu que je cherche des images d'Alain Bombard à Lorient ?

Oui
. Je trouve 63 photos
. je visualise celles qui sont diffèrentes des précédentes.

PHASE DE VISUALISATION au cours de laquelle l'utilisateur indique les photos qu'il choisit (les 10 ou figure Mr X)

IV . Je n'ai plus d'inférence simple à faire, en as-tu ?

J'observe qu'une des images montre un évènement concernant l'association écologique GRIGRI (peut-être qu'en recherchant d'autres images de GRIGRI j'en trouverai montrant Mr X)
Requête : "GRIGRI'

etc...

V . Veux-tu aussi illustrer le "fondement" de la recherche : la soudaine promotion d'un homme ?
Oui

etc...

PHASE DE VISUALISATION (un départ de course, un décollage d'avion, ...) au cours de laquelle
choisit le décollage d'avion

etc...

VI Revisualisation de toutes les images choisies et sélection finale.
Mise à jour du lexique et des descriptions des images ou Monsieur X apparaît.

CHAPITRE B

TYPES ABSTRAITS

CHAPITRE B

TYPES ABSTRAITS

B-I) JUSTIFICATION:

B-I-1) Présentation:

Depuis une douzaine d'années, devant la difficulté de comprendre et de concevoir les programmes, on a cherché à développer des outils de spécification statique des problèmes.

Il apparaît de plus en plus que la première étape de la programmation consiste à spécifier l'"univers" des objets que l'on utilise dans un programme. Cet univers est composé d'objets de natures différentes : d'où l'idée de distinguer des types dans cet univers. Chaque type possède ses règles propres d'utilisation et de comportement : on parlera des fonctions ou des opérations attachées à un type d'objets.

Exemples :

1) Un type bien connu est celui des entiers : ses fonctions sont toutes les opérations de l'arithmétique : +, -, *, etc ....

2) Pour écrire un programme d'édition de texte on aura les types : ligne, caractère,l'opération:supprimer une ligne, etc ....

La notion de types abstraits s'avère comme un outil intéressant de spécification, de structuration, de lisibilité de construction et de vérification de programme.

Parallèlement, des méthodes de spécification des types abstraits ont été développées : ce sont essentiellement la méthode de Hoare <HOA 72> que nous appellerons "axiomatique", qui est utilisée systématiquement en ALPHARD, et la méthode "algébrique" <GOG ; THAT ; WAU 76> <GUT, HOR, MUS 78>, <REM 82> ; <REM 82> ....

L'une et l'autre de ces deux méthodes permettent de spécifier les types abstraits et leurs opérations, puis, étant donné un module de représentation, de faire la preuve que ce module satisfait les spécifications. La méthode axiomatique est directement utilisable dans un langage algorithmique, car les spécifications sont faites en terme d'assertions inductives (pré et post conditions des opérations) et d'invariants ; le point faible est qu'on ne peut définir le domaine abstrait (auquel appartiennent les objets du type) que par restriction de domaines standard prédéfinis.

La méthode n'est donc pas constructive. A l'inverse, la mèthode algébrique donne précisement la nature du domaine abstrait ; mais le langage algébrique ne possède pas les concepts de "variables", "affectation", etc... ce qui le rend a priori impropre à la programmation algorithmique traditionnelle.

Dans la suite de ce travail nous nous plaçons dans un cadre de spécification à l'aide de types abstraits algébriques (T.A.A). Nous dégagerons les principaux avantages que présentent pour nous les types abstraits et nous justifierons le choix d'une présentation algébrique.

B-I-2) Pourquoi utiliser des types abstraits (T.A) :

Avantages que présentent pour nous les T.A. :

* Caractère constructif

Les T.A. possèdent des propriétés de dérivation qui permettent de construire progressivement un type à partir d'un autre déjà défini en lui ajoutant :

- soit des opérations et on parle alors d'enrichissement de types <REM 82>, <BUR et GOG 77>
- soit de nouvelles sortes, de nouvelles opérations et de nouveaux axiomes. On parle alors d'extension de type.

La suite de ce travail nous montrera que cet aspect constructif nous servira beaucoup.

* Caractère structurant

L'approche T.A. amène à distinguer dans une application les principaux types d'objets qui y interviennent et à chercher à spécifier le comportement de chacun de ces types d'objets en soi et vis-a-vis des autres. Cette démarche conduit donc à structurer l'environnement de l'application, à mettre en évidence et localiser ses principaux centres d'intéret, à isoler les opérations les plus intéressantes ou délicates.
Ainsi nous pouvons souligner que l'utilisation des T.A. permet une certaine rigueur lors de la conception d'un projet.

* Pouvoir d'abstraction

L'abstraction consiste à penser à un objet en termes des actions que l'on peut faire sur lui et non en termes de représentation.

Le fait de pouvoir spécifier objets et opérations indépendamment de leur représentation est une aide importante à la conception du projet EXPRIM et donc à sa réalisation.

* Généricité

Une propriété des T.A. qui nous semble également très intéressante est la possibilité de définir des schémas des T.A. Cette notion permet une grande économie de pensée dans la mesure ou une même spécification peut être utilisée dans différents contextes. (La construction d'une pile d'entiers sera la même que celle d'une pile de caractères). On a ainsi la possibilité de généraliser les constructeurs <DER et FIN 79>, <GOG et PAR 81>

Nous verrons également que cette propriété des T.A. nous sera également très utile.

B-I-3) Pourquoi une présentation algébrique:

- Une présentation algébrique permet d'obtenir une représentation concrète des objets qui offre la possibilité d'effectuer des calculs.

- On représente les éléments d'un type comme des expressions composées à l'aide des constructeurs, donc en fait des arbres. Seule la spécification algébrique permet de définir un objet quelconque d'un type à partir d'une suite d'opérations en partant d'un objet initial.

- On définit les opérations par des équations qui sont orientées comme des règles de réécriture. Les règles permettent de calculer la valeur du résultat d'une opération sur des éléments par transformation d'arbres.

- Plusieurs études ont été réalisées pour développer des méthodes de preuves de la complétude suffisante et de la consistance d'une spécification algébrique. Bien que ces propriétés soient en général indécidables, on obtient des conditions vérifiables algorithmiquement en se restreignant à des classes de spécifications particulières. Nous reviendrons plus tard sur les problèmes de consistance et de complétude d'une spécification.

B-II) CADRES ALGEBRIQUES D'UNE SPECIFICATION: <PAI 80> <BER 79>

Avant de présenter de façon plus approfondie les T.A.A., nous allons faire les rappels d'algèbre nécessaires à leur compréhension.

a) Signature:

Une signature est formée :

- de sortes S1....Sn en nombre fini et interprétées comme des ensembles ; l'ensemble des sortes est noté : S

- de symboles d'opérations chacun muni d'une fonctionnalité, ou profil.

$$Si1 \times \ldots \times Sip \longrightarrow Sj \quad (p > 0)$$

L'ensemble des symboles d'opérations est noté Σ .

b) Algèbre associé à une signature :

Soit une signature (S, Σ), une $(S, \Sigma 0)$ - algèbre est la donnée :

$(\forall\, d \in S)$ d'un ensemble Ad
$(\forall\, f \in \Sigma)$ tel que f: d1 x d2 x ... x dn --> dn+1

d'une application fA : Ad1 x Ad2 x ... x An1 --> Adn+1

Exemple :

Soit la signature NAT = < {NAT}, {0, S} > (S ={nat} ; Σ = {0, S})

Les opérations 0 et S ont les profils suivants :

--> nat : 0

nat --> nat : S

Les algèbres suivantes sont des NAT - algèbres. (nous utilisons des notations du λ - calcul).

(A1) $A_{nat} = N$ $\quad O_{A1} = 0$ $\quad S_{A1} = \lambda x.x+1$

(2) $A_{nat} = N$ $\quad O_{A2} = 0$ $\quad S_{A2} = \lambda x.2*x$

(3) $A_{nat} = N$ $\quad O_{A3} = 0$ $\quad S_{A3} = \lambda x.x$

(4) $A_{nat} = \{|k : k > 0\}$ $\quad O_{A4} = |$ $\quad S_{A4} = \lambda x.|x$

Avec une telle définition, on voit qu'il existe une infinité d'algèbres pour une signature donnée (S,Σ). Ces Σ-algèbres sont d'ailleurs comparables à l'aide de Σ-homomorphismes qui étendent la notion d'homomorphisme classique pour tenir compte des domaines.

Définition :

Etant donné une signature (S,Σ), un Σ-homomorphisme h entre deux (S,Σ) - algèbres A et B est une famille d'applications $\{h_d \mid d \in S\}$ telle que :

$(\forall f : d1 \times d2 \times \ldots \times dn \dashrightarrow dn+1 \in \Sigma)$

$h_{dn+1}(f_A(a1,\ldots, an)) = f_B(h_{d1}(a1), \ldots, h_{(dn)}(a1))$

La question que l'on se pose est de savoir s'il existe une algèbre. Considérons en effet l'algèbre suivante construite sur l'exemple de NAT : TNAT. Le domaine est l'ensemble des chaînes sur "0", "S", "(", ")", ","
avec : $O_{TNAT} = 0$ $\quad S_{TNAT} = x$ "s" "(" x ")"

Ex : les éléments du domaine sont : 0, S(0), S(S(0)), ...

Une algèbre construite de cette manière à partir d'une signature (S,Σ) s'appelle l'algèbre des termes des termes clos ou sans variables. On la note :

$T_{S,\Sigma}$ ou T_{Σ}.

Théorème :

Il existe un homomorphisme unique entre l'algèbre des termes $T_{S,\Sigma}$ et toute (S,Σ) - algèbre.

Applications :

Etant donné un terme $t \in T_{NAT}$, on peut calculer sa "valeur" (qui est unique) dans une NAT-algèbre.

Ex :

Soit le terme t = S(S(S(S(S)0)))))
L'homorphisme h_{A1} entre T_{NAT} et A_1 est défini par

$$h_{A1}(0) = 0$$

$$\forall x \in T_{NAT} \quad h_{A1}(S(x)) = h_{A1}(x) + 1$$

D'où : $h_{A1}(t) = 5$

De même

$$h_{A2}(t) = \underbrace{S(\underbrace{S(\underbrace{S(\underbrace{S(\underbrace{S(0)}_{2})}_{4})}_{8})}_{16})}_{32} = 32$$

$$h_A{}^3(0) = 0$$

$$h_{A3}(t) = 0$$

$$h_{A4}(0) = |$$

$$h_{A4}(t) = ||||||$$

B-III) TYPES ABSTRAITS ALGEBRIQUES

B-III-1) Définitions, exemples et notations:

Il semble maintenant généralement admis <GAU 80>, <GUT et HOR 78>, <GOG, BUR 77> <JUL 83> de considérer un type abstrait comme une théorie algébrique typée formée par la donnée d'un triplet <S, Σ, E> où :

. S est une liste de noms de types (ou sortes) où l'on distingue un type d'intérêt, qui est le type que l'on veut définir.

.Σ une liste de noms d'opérations sur ces types ; à chacun de ces noms est associé un profil composé d'un domaine et d'un co-domaine constitués de types de S ,selon la notation suivante:

∀ fi ∈ Σ , nous notons :

- << (t0, t1, ..., tk) --> tj : fi >> pour dire que l'opération fi a pour profil (t0, t1, ..., tk) --> tj où t0, ..., tk, tj ∈ S et tel que k ∈ <1,n> et j ∈ <0,n>. On dit que t0, t1, ... tk est le domaine de fi et que tj est son co-domaine.

- << ( ) --> t0 : fi >> pour dire que fi est une constante de profil

( ) --> t0.

Les symboles d'opérations permettent de construire des termes qui sont des schémas fonctionnels du type abstrait.
t0 est dit type d'"interêt", c'est-à-dire celui que l'on est en train de spécifier.

.E est une liste d'axiomes (ou lois) qui expriment les relations entre les opérations. Ces axiomes sont des équations entre termes composés de noms d'opérations et de variables. Celles-ci sont universellement quantifiées de façon implicite.
Nous verrons que E définit les classes d'équivalence des termes de l'algèbre.

Remarque :

Pour construire l'axiomatisation, en général, nous partitionnerons l'ensemble d'<<opérations>> en trois sous-ensembles :

- C : Les constructeurs à co-domaine dans le type d'intéret.

En général C contient :

. un opérateur qui est une constante (<< créer>>) de création d'un objet de type "type d'intérêt" tel que ( ) --> t0 : créer

. un opérateur de construction d'un objet de type tO à partir d'un objet de type tO et d'un autre objet tel que :

(tO, tk) -->tO: ajouter avec k $\in$ <1,n>

- Ex : !es extensions dont le co-domaine est le type d'intérêt et qui peuvent se réécrire à partir des constructeurs et d'autres extensions. En général,cet ensemble est formé du seul opérateur d'obtention d'un objet de type tO à partir d'un autre tel que :

(tO, tk) --> tO : enlever

- O : !es opérateurs externes à co-domaine diffèrent du type d'intéret, en général :

. un opérateur permettant d'accéder à une donnée mémorisée par un objet de type type d'intérêt tel que :

(tO) --> tk : valeur avec k $\in$ <1,n>

. un ensemble d'opérateurs <<annexes>> utiles à la définition de la sémantique tels que : (tO) --> tl : opl

A partir de ces trois ensembles, on construit l'ensemble d'axiomes E qui permet d'exprimer la sémantique du type. Pour qu'elle soit bien exprimée, il faudra que le système vérifie les propriétées de consistance et de complétude sur lesquels nous reviendrons plus en détail.

Exemple : spécification des entiers naturels

type Entier, Booléen

Opérations :

( ) --> Entier : zéro

(Entier) --> Entier : succ (fonction successeur)

(Entier) --> Entier : pred (fonction prédécesseur)

(Entier) --> Bool : nul? (fonction qui teste si un entier est nul)

(Entier, Entier) --> Entier : plus

Dans ce cas :

C = {zéro, succ}

O = {nul?}

EX : {pred, plus}

Le type d'intérêt est le type Entier
Booléen est un type (ou sorte) prédéfini qui sert dans la spécification du type Entier.

Axiomes:

Soit x,y ∈ Entier
---

(1) pred (zéro) = zéro

(2) pred (succ(x)) = x

(3) nul? (zéro) = VRAI

(4) nul? (succ(X)) = FAUX

(5) plus (zéro,y) = y

(6) plus (succ(x),y) = succ(plus(x,y))

On voit que la partie Opérations de la spécification définit un langage, celui des "termes du type Entier" et que la partie Axiomes définit une ou des relations de congruence sur les termes de ce langage. S'il n'y a pas d'axiomes, tous les termes du type abstrait sont différents. Dans le cas général, où il y a des axiomes, ceux-ci définissent des classes d'équivalence dans le langage des termes.

B-III-2) Propriétés d'une spécification

On voit que la présentation d'un type abstrait algébrique est d'une syntaxe relativement simple. Le langage de base comporte trois primitives :

. La composition de fonctions (avec parenthèses pour ajouter à la lisibilité).

. Le signe =

. Des variables libres

Si la syntaxe de telles spécifications est élémentaire, leur sémantique est moins immédiate et demande à être étudiée soigneusement.
Etant donné une spécification (S, Σ, E) on peut se poser la question de savoir si elle est conforme à ce qu'on attend d'elle, si elle ne conduit pas à des contradictions, et si elle définit complètement l'ensemble d'objets et d'opérations du type présenté. Si le premier problème n'est pas aisé à appréhender les deux derniers peuvent être désignés comme ceux de la consistance et de la complétude d'une spécification.

a) Consistance d'une spécification

La consistance d'une spécification exprime le fait que les axiomes ne conduisent pas à des contradictions vis-à-vis des types définis et utilisés dans la spécification.

Si l'on retire de la spécification précédente l'axiome pred(zéro) = zéro et que l'on ajoute l'axiome (7) succ(pred(x)) = x on obtient une spécification inconsistante car pour :

t = succ(pred(zéro)) on a :

nul? (succ(pred(zéro)) = FAUX (axiome 4)

axiome (7) axiome (3)

et nul? (succ(pred(zéro)) = nul? (zèro) = VRAI

b) Complétude d'une spécification

Savoir qu'une spécification algébrique d'un type abstrait est consistante est en général insuffisant. En effet on doit être capable de prouver que l'égalité entre deux termes est vraie ou fausse.

Il existe deux principale approches pour définir la sémantique d'un type abstrait:

-l'approche algèbre initiale

-l'approche algèbre terminale

Dans l'approche algèbre initiale on considère que deux objets sont équivalents si et seulement si,en utilisant l'ensemble d'axiomes E,on peut prouver qu'ils le sont.L'algébre initiale est la Σ-algébre quotient de $T_{S,\Sigma}$ par $\equiv_E$

On travaille sur les classes d'équivalence de termes au lieu de travailler sur les termes).
Si on essaie de se dégager des détails techniques,cette définition revient à dire qu'on satisfait les équations de E et rien de plus.

Dans l'approche algèbre terminale on considère que deux objets sont différents si en leur appliquant une opération de O (opérateur externes à co-domaine différent du type d'intérêt,qu'on appelle aussi observateurs), ils renvoient des valeurs différentes.En d'autres termes deux objets sont égaux si un utilisateur peut les observer comme étant deux "boîtes noires" ayant le même commportement vis à vis de l'"extérieur".On définit ainsi une équivalence de comportement, vis à vis de n'importe quels observateurs,entre les objets du type abstrait. L'approche algèbre initiale est décrite dans <GOG 78> et <REM 82>, l'approche algèbre terminale est décrite dans <LES 83>.

c) T.A.A. et systèmes de réecriture

Une spécification algébrique d'un type abstrait est donnée par une signature (ensemble de sortes et opérations avec leurs profils) et un ensemble d'équations. Ces équations décrivent des relations entre les opérations et définissent, en particulier, une algèbre qui vérifie exactement ces relations.

Quand on raisonne avec des équations il y a un certain nombre de propriétés que l'on veut vérifier, telles que la consistance et la complétude.

Il n'existe pas de méthode complètement générale pour décider de ces propriétés. Toutefois de nombreux résultats pratiques ont été obtenus lorsque les spécifications peuvent être regardées comme des systèmes de réécriture.

Si on fait certaines restrictions sur la manière dont on écrit les équations d'un T.A.A. on peut employer les résultats de la théorie des systèmes de réécriture pour prouver la consistance et la complétude d'une spécification. Cette approche se trouve dans les travaux de Pierre Lescanne <LES 79> et Jean-Luc Rémy <REM 82>.

Un système de réécriture est d'une façon assez simple, un outil de raisonnement avec des équations. Il permet de faire des preuves par induction (recurrence) dans des théories définies par un ensemble d'équations, de vérifier la complétude d'une théorie ou sa consistance.

Nous décrirons de manière plus détaillée ce qu'est un système de réécriture ainsi que les critères syntaxiques qui permettent d'écrire des spécifications permettant d'utiliser les résultats de la théorie des systèmes pour faire des preuves de propriétés sur les T.A.A. <KIR 82> <REM 82>.

d) Le traitement des erreurs:

Les restrictions sur la définition des opérations sont que celles-ci soient complètement définies alors que dans les cas réels les types ont souvent des opérations partielles (Expl : sommet (pile) n'est pas défini sur pile = pilevide).

Il existe plusieurs approches pour résoudre le problème de totalisation d'une spécification ayant des opérations partiellement définies.<GOG 78>,<REM 83> <BRO 82>. La plupart des solutions ont un point commun:c'est la définition d'une (ou plusieurs)constantes d'erreurs par sortes.
On complète la spécification en ajoutant,pour les points où les opérations partielles n'ont pas été définies, des règles avec la constante erreur.

Exemple:

Si nous reprenons la spécification des entiers naturels et que nous introduisons l'opération définie moins, de profil :

(Entier,Entier) ------> Entier

avec les axiomes suivants:

Soit x,y ∈ Entier

moins(x,zero)=x

moins(succ(x),succ(y))=moins(x,y)

L'opération moins n'a pas été définie dans le cas où le premier paramètre est zéro.
Afin de compléter la spécification on ajoute l'axiome:

moins(zero,succ(x))=ERR

ERR est la constante erreur du type entier.

Il faut donc l'ajouter dans C et son profil est:

( ) -----> (Entier) : ERR

Le problème pour compléter la spécification est de trouver les axiomes manquants et de s'assurer qu'après les avoir introduits la spécification est à la fois complète et consistante.

La propagation des erreurs:

On a introduit dans les sortes de nouveaux termes contenant la constante erreur, il faut définir le comportement des opérations du type vis à vis de ces termes. Il faut ajouter à la spécification des axiomes qui,pour chaque opération,propagent les erreurs.

Ainsi,dans la spécification du type Entier on ajoute la propagation des erreurs du constructeur:

succ(ERR) = ERR

et la propagation des erreurs pour les opérations définies:

moins(ERR,x)=ERR

moins(y,ERR)=ERR

B-IV) SYSTEMES DE REECRITURE

B-IV-1) Quelques définitions

Un système de réécriture est un ensemble de règles notées G --> D, où G et D sont des termes composés de symboles d'opérations et de symboles de variables, et tels que l'ensemble de variables de D, noté V(D) est inclus dans l'ensemble V(G) de variables de G.

On appelle alors substitution un ensemble σ de couple (xi, ti) ou les xi sont des variables et les ti des termes.

Appliquer une substitution σ à un terme t consiste à remplacer chaque occurrence des xi dans t par le terme ti correspondant, le terme obtenu est noté σ t.

On dira alors qu'un terme t se réécrit en un terme t' si il existe une substitution σ, un sous-terme s de t et une règle g --> d tel que σ g = s et t' est le terme obtenu par remplacement du sous-terme s par σ d.

A un système de réécriture R, on associe une relation binaire $\xrightarrow{R}$ appelée relation de réduction.

Un terme t est dit en forme normale ou irréductibe si et seulement si il n'existe pas de terme t' tel que

$$t \xrightarrow{R} t'$$

c'est à dire si t ne peut plus se réécrire dans R.

B-IV-2) Terminaison d'un système de réécriture

Définition :

Un système de réécriture R est noethérien si et seulement si pour tout terme t il n'existe pas de dérivation infinie

t = t0 --> t1 --> .... --> tn ....

On dit aussi dans ce cas que la relation --> a la propriété de terminaison finie.

Quand un système est noethérienm tout terme a au moins une forme normale. On dit aussi qu'il est à terminaison finie.

La terminaison finie est en général indécidable, mais il est possible d'établir des critères suffisants qui la garantissent,par exemple en proposant un ordre sur les termes et de s'assurer que pour chaque règle le membre droit est "plus petit" que le membre gauche.

Plusieurs auteurs ont proposé divers ordres pour prouver la terminaison finie des systèmes de réécriture <DER 79>,<LES 81>,<LES 84>.

B-IV-3) Confluence d'un système de réécriture

Notation : $\xrightarrow{*R}$ désigne la fermeture réflexive transitive de $\xrightarrow{R}$

Définition :

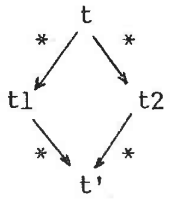
Un système de réécriture est confluent si et seulement si pour tous termes t, t1, t2 tels que

$$t \xrightarrow{*} t1 \quad \text{et} \quad t \xrightarrow{*} t2$$

il existe un terme t' tel que

$$t1 \xrightarrow{*} t' \quad \text{et} \quad t2 \xrightarrow{*} t'.$$

On l'exprime par le diagrame :



Remarque :

Quand un système de réécriture est confluent, tout terme a au plus une forme normale(c' est à dire une forme irréductible qui lui est équivalente par R).

La confluence d'un système de réécriture est en général indécidable. Mais ce n'est pas le cas pour des systèmes noethériens et l'idée de base est de localiser le test de confluence.

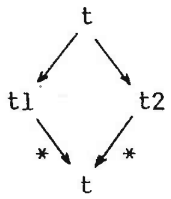
Confluence locale :

Définition :

Un système de réécriture est localement confluent si et seulement si pour tous termes t, t1, t2 tels que t ---> t1 et t ---> t2 il existe un terme t' tel que :

$$t1 \xrightarrow{*} t' \quad \text{et} \quad t2 \xrightarrow{*} t'.$$

Cette propriété dite du losange se visualise par le diagramme :



Lemme du losange

Un système de réécriture noethérien est confluent si et seulement si il est localement confluent. Ce lemme, du à M. Newmann, doit son importance au fait qu'il existe un moyen simple de tester la confluence locale d'un système de réécriture. Le test est du à D. Knuth et P. Bendix.

Définition :

Un unificateur de deux termes t1 et t2 est une substitution σ telle que $\sigma(t1) = \sigma(t2)$. Si σ existe, t1 et t2 sont dits unifiables et σ est une solution de l'équation (t1 = t2).

Proposition :

Si deux termes t1 et t2 sont unifiables, ils admettent un unificateur minimum.

Paires critiques et théorèmes de Knuth-Bendix :

Nous supposerons dans toute la suite que deux règles G --> D et G' --> D' de R vérifient $V(G) \cap V(G') = \emptyset$, condition que l'on peut toujours réaliser par un renommage des variables.

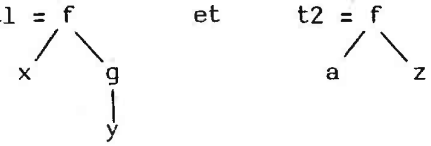
Définition :

Soient (G,D) et (G',D') deux couples de termes. Supposons qu'il existe un sous-terme non variable de G unifiable avec G'. Soient m l'occurrence de ce sous-terme et σ l'unificateur minimum de $G_{|m}$ et de G'. Alors la paire

$\{\sigma(G\langle m\langle\text{--- } D'\rangle), \sigma(D)\}$ est appelée paire critique, obtenue par superposition de (G',D') sur (G,D).

Exemple :

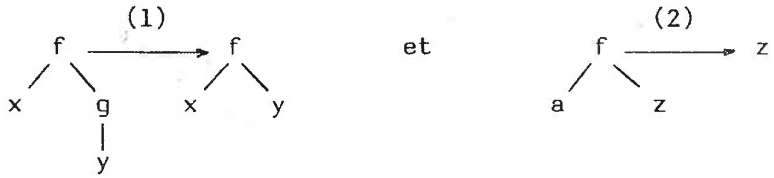
Soient les deux termes t1 = f(x, g(y)) et t2 = f(a, z)



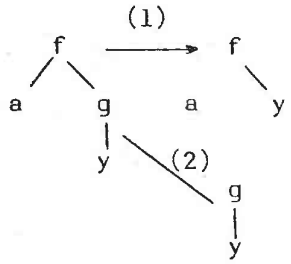
Un unificateur pour les deux termes est la substitution :

$$\sigma = \{(x / a), (z / g(y))\}$$

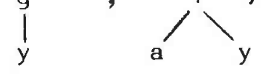
Si de plus on a les règles :



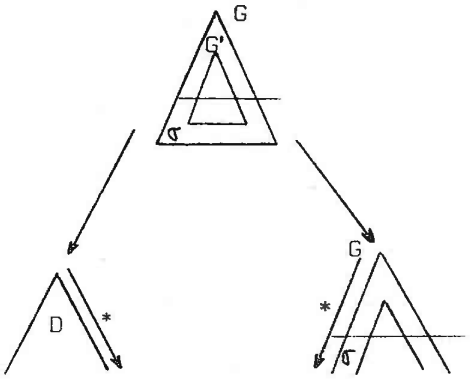
En supposant la règle (1) sur le règle (2) on a :



On obtient la paire critique $(g(y), f(a, y))$



Ceci peut se résumer par le schéma suivant :



L'intérêt des paires critiques vient du théorème suivant :

Théorème de Knuth-Bendix :

Un système de réécriture de termes R est localement confluent si et seulement si pour toute paire critique (t,t') de R, il existe t" tel que

$$t \xrightarrow{*} t'' \quad \text{et} \quad t' \xrightarrow{*} t''$$

Dans le cas où le système R est fini et noethérien, le résultat donne une procédure de décision pour la confluence de R : comme il n'existe qu'un nombre fini de paires critiques, il suffit pour chacune d'elles de calculer les formes normales des deux membres et de tester l'égalité.

B-V) SYSTEMES DE REECRITURE ET SPECIFICATIONS STRUCTUREES DE TYPES ABSTRAITS

<REM 82>

B-V-1) Définitions :

Une spécification (S,Σ,R) est dite structurée si elle peut être associée à un système de réécriture confluent et à terminaison finie et qui vérifie quelques propriétés supplémentaires sur ses formes normales.

En particulier,l'ensemble des opérations et l'ensemble des règles peuvent être partionnées en deux de telle sorte que la spécification consiste en une définition de la deuxième partie par rapport à la première.

Une spécification (S,Σ,R) est structurée si (Σ,R) peuvent être partitionnés en :

$\Sigma = \Sigma 0 \cup \Sigma 1$, $R = R0 \cup R1$ de telle façon que :

1) (S,Σ,R) soit un système de réécriture confluent

2) Les membres gauches de R1 contiennent au moins un symbole de $\Sigma 1$

3) R est à terminaison finie

4) Les formes normales des termes clos sont des termes primitifs.

Les opérations de $\Sigma 0$, $\Sigma 1$ sont appelées constructeurs et opérations définies.

Les règles de E0 et E1 sont appelées relations entre constructeurs et définitions.

Remarques :

La première condition entraîne que les règles de R0 opèrent sur les $\Sigma 0$-termes (termes primitifs). De plus R0 est nécessairement à terminaison finie et, donc, tout $\Sigma 0$-terme admet une forme normale.

La deuxième condition entraîne que les $\Sigma 0$-termes sont R1-irréductibles, et donc que leurs formes normales $\bar{t}_{R0}$ et $\bar{t}_{R}$ pour R0 et R coincident.

La dernière condition entraîne en particulier que pour toute opération f de $\Sigma 1$, tous $\Sigma 0$-termes t1 .... tn, il existe un $\Sigma 0$-terme t0 tel que

$$f t1 \ldots tn \xrightarrow{*} t0$$

et tel que t0 soit une forme normale. Cette condition implique que (S,Σ,R) est complète vis-à-vis de $(S,\Sigma,R0)$.

Exemple :
---

type Entier relatif

Opérations

Constructeurs

( ) --> Entier : zéro
(Entier) --> Entier : succ
(Entier) --> Entier : pred

} Σ 0

Opérations définies :

(Entier) --> Entier : opp
(Entier, Entier) --> Entier : plus
} Σ 1

Axiomes :

Relations entre constructeurs
---

pred(succ(x)) --> x
succ(pred(x)) --> x
} R0

Définitions :
---

opp(zéro) --> zéro
opp(succ(x)) --> pred(opp(x))
opp(pred(x)) --> succ(opp(x))
plus(zéro,y) --> y
plus(succ(x),y) --> succ(plus(x,y))
plus(pred(x),y) --> pred(plus(x,y)0
} R1

Consistance d'une spécification (S,Σ,E) vis-à-vis de (S0,Σ0,E0):

Une extension (S,Σ,E) est consistante vis-à-vis de (S0,Σ0,E0) si et seulement si la restriction de la congruence $\equiv_E$ aux Σ0-termes coincide avec la congruence $\equiv_{E0}$, c'est-à-dire si :

$$t0 \equiv_E t'0 \Rightarrow t0 \equiv_{E0} t'0$$

Complétude, complétude suffisante :

Une extension (S,Σ,E) est complète vis-à-vis de (S0,Σ0,E0) si et seulement si pour toute sorte s0 de S0, pour tout t de $T_{\Sigma,S0}$ il existe t0 de $T_{\Sigma0,S0}$ tel que : $t \equiv_E t0$

On retrouve une notion voisine de complétude suffisante de GUTTAG <GUT 78> qui s'énonce comme suit :

Définitions :

Soit $< S + SO , \Sigma + \Sigma O, E + EO >$ une présentation d'un type abstrait S avec :

ΣO : le sous-ensemble des opérations dont le co-domaine est le type d'intérêt S

Σ : le complémentaire de ΣO (c'est-à-dire les opérations à valeurs dans un type externe avec l'hypothèse $\Sigma O = \emptyset$)

S est suffisamment complet si, pour tout terme de la forme f(x1, ...,xn) où $f \in \Sigma$, il existe un théorème démontrable à partir des axiomes du type S de la forme f(x1, ..., xn) = u ou u est un terme ne contenant aucune opération de S (u est un terme externe).

Le concept de complétude suffisante apparaît comme une condition minimum pour qu'un type abstrait algébrique soit intéressant.

Afin d'"extraire de l'information" on demande à une spécification algébrique de contenir au moins une fonction dont le domaine n'est pas le type d'intérêt (telle que la fonction nul? dans le type Entier).

Condition de consistance d'une spécification structurée :

Proposition :

Une spécification structurée (S,Σ,R) telle que R soit confluent est consistante.

Condition de complétude d'une spécification structurée ;

Proposition :

Soit R = R0 ∪ R1 un système de réécriture à terminaison finie. Si pour tout f de $T_{\Sigma 1}$ tous t1...tn de $T_{\Sigma O}$, il existe une règle G --> D de R et une substitution σ telle que f(t1 ... tn) = G σ alors R constitue un système complet de définitions.

B-V-2) Preuve de la consistance d'une spécification structurée

Pour prouver la consistance d'une spécification structurée nous utilisons l'algorithme de complétion du à Knuth et Bendix et dont le principe est le suivant :

- orienter les équations pour constituer un système de réécriture à terminaison finie.

- vérifier la propriété de confluence en montrant que les formes normales de certaines paires critiques de termes coincident.

- si ce n'est pas le cas, ajouter de nouvelles règles en orientant les couples de formes normales obtenus pour préserver la terminaison finie ; poursuivre alors la preuve de confluence jusqu'à un (éventuel) succès complet.

Le résultat de l'algorithme, s'il existe, est un système de réécriture convergent, c'est-à-dire confluent et à terminaison finie, définissant les formes normales recherchées.

Pour prouver la consistance d'une spécification vis-à-vis d'une spécification primitive, on applique l'algorithme de complétion ou système de réécriture de la spécification primitive. On ajoute les définitions de nouvelles opérations et on applique à nouveau l'algorithme de complétion en s'imposant de ne pas introduire des règles qui réduisent les termes primitifs.

De cette façon on montre que les systèmes convergents obtenus définissent les mêmes formes normales sur les termes primitifs. Cette application de l'algorithme de Knuth-Bendix est décrite dans <KIR 83>.

B-V-3) REVE <LES 83>

Pour prouver la terminaison finie et la complétude de quelques uns de nos types nous avons utilisé le système REVE développé par Pierre Lescanne <LES 83>.

REVE est un générateur de système de réécriture ; REVE lit des équations et produit des règles de réécriture vérifiant les propriétés de terminaison finie et de confluence.

Pour prouver la propriété de confluence (ou propriété de Church-Rasser) il utilise l'algorithme de Knuth-Bendix <KNU et BEN 70>. Cet algorithme transforme un ensemble d'équations en un système de réécriture convergent équivalent (équivalent au sens que le système de réécriture prouve les mêmes théorèmes que l'ensemble d'équations).

L'algorithme de Knuth-Bendix génère de nouvelles équations qui seront orientées afin de donner lieu à de nouvelles règles de réécriture.

Exemple :

Soit l'ensemble d'équations suivant :

x * e = x

x * i(x) = e

(x * y) * z = x * (y * z)

x / y = x * i(y)

Dans un premier temps l'algorithme oriente les équations de gauche à droite et engendre les règles de réécriture suivantes :

(1) x * e ---> x

(2) x * i(x) ---> e

(3) (x * y) * z ---> x * (y * z)

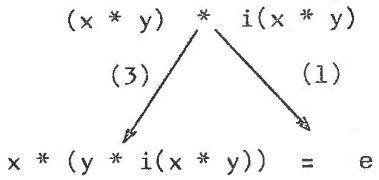
(4) x / y ---> x * i(y)

L'algorithme calcule les paires critiques entre les règles obtenues.

Exemple :

Soit t = (x * y) * i(x * y)

La superposition de la règle (1) sur la règle (3) donne la paire critique :



L'orientation de cette opération est ici triviale et l'algorithme de K.B engendre la règle :

(5) x * (y * i(x * y)) --> e

Il engendre ainsi un certain nombre de règles et peut également en supprimer. L'algorithme engendrera, par exemple, les règles :

i(x * y) ---> i(x) * i(y)

y * (i(y) * n) ---> x

et la combinaison de ces règles permet de simplifier la règle (5).

L'orientation d'une équation est faite par un algorithme qui teste que la règle préserve la terminaison finie de tout le système.

En définitive, l'algorithme engendre un système de réécriture convergent composé d'un certain nombre de règles, équivalent au système d'équations initial.

Dans REVE la terminaison finie du système de réécriture est prouvée au fur et à mesure que des équations sont transformées en règle;l'orientation de l'équation est faite de telle sorte que la terminaison est conservée.Dans REVE la méthode qui assure cette terminaison est basée sur l'ordre de décomposition avec statut. <JOU,LES,REI 84>,<LES 84>.

CHAPITRE C

SPECIFICATION A L'AIDE DE T.A.A. DE LA BASE DESCRIPTIVE ET DE SON INTERROGATION

CHAPITRE C

SPECIFICATION A L'AIDE DE T.A.A. DE LA BASE DESCRIPTIVE ET DE SON INTERROGATION:

Nous allons, dans un premier temps, présenter une description algébrique assez élémentaire du fonctionnement de l'ensemble du système. Cette description aura l'avantage de mettre en valeur quelques uns des principaux types utilisés par notre système. Nous essaierons, autant que possible, d'adopter une méthode de spécification "descendante", c'est-à-dire partant du fonctionnement général du système; nous donnerons, plus loin, une spécification plus complète des types manipulés par EXPRIM.

Nous utiliserons, au maximum, toutes les propriétés de construction de types citées chapitre B.

- extension de T.A.

- paramétrisation de T.A.

C-I) DESCRIPTION ALGEBRIQUE ELEMENTAIRE DU FONCTIONNEMENT DE L'ENSEMBLE DU SYSTEME

Les images sont représentées, dans la base descriptive (Bdi), par leur description et une référence. Elles seront sélectionnées par le biais de requêtes et visualisées sur un écran suivant différents modes : (plein écran, damier, ...). Les opérations que l'on peut effectuer sur la base descriptive sont les suivantes :

- Ajouter une description d'image et sa référence. (référence €N)

- Supprimer une description d'image et sa référence.

- Rechercher les images en adéquation avec sa requête.

- Les visualiser sur l'écran par le biais de leurs références.

Spécification du type Base Descriptive d'Image: (Bdi)

Les sortes apparaissant dans cette spécification sont les suivantes:

- Descripimage : description d'image
- Requête : requête pouvant être adressées à la base descriptive
- Reférence : numéro de référence d'une image sur le vidéodisque.
- Ensderéférence: Ensemble de références.
- Ecran

type Bdi, Descripimage, Requête, Reference, Ecran,Ensdereference

Opérations :

( ) ----> Bdi : bdvide

(Bdi,(Descripimage,Reference)) ---> Bdi : ajdescription

} constructeurs

(Bdi,(descripimage,Reference)) ----> Bdi : supdescription

(Bdi,Requête) ----> Ensdereference : Rech

} opérations définies

Axiomes

Soit b ∈ Bdi ; i, i' ∈ Descripimage ; r ∈ Requête ; n, n' ∈ Reference

Relation entre constructeurs ;

ajdescription(ajdescription(b,(i,n)),(i',n')) = si eg<(i,n),(i',n')> alors
ajdescription(b,(i,n))
sinon
ajdescription(ajdescription(b,(i',n')),(i,n))
fsi

Définitions :

supdescription(bdvide(i,n)) = bdvide

supdescription(ajdescription(b,(i,n)),(i',n')) = si eg<(i,n),(i',n')> alors
sinon
  supdescription(b,(i',n'))
sinon
  ajdescription(supdescription(b,i',n'),(i,n))
fsi

Rech(bdvide,r) = ∅

Rech(ajdescription(b,(i,n)),r) = si adequation(i,r) alors ajouteref(Rech(b,r),n)
sinon Rech(b,r)
fsi

Les images se trouvent sur un vidéodisque et l'accès à une image se fait par le biais de sa référence.
On définit sur le type Vidéodisque une opération de visualisation de profil :

(Vidéodisque,référence) -----> Ecran

Le type Bdi fait apparaître les types Descripimage et Requête dont nous donnerons une spécification complète au paragraphe suivant. Il met également en évidence une opération importante : "adéquation" de profil.

(descripimage, requête) ----> Bool

et qui fera également l'objet d'une étude approfondie.

L'opération ajouteref est définie dans le type Ensemble de références que nous spécifierons dans C-II).

C-II) SPECIFICATION DES TYPES DESCRIPIMAGE ET REQUETE:

Nous présentons, dans un premier temps, une spécification du type Ensemble d'éléments qui nous sera utile dans toute la suite de notre travail.

Type Ensemble,élément,Bool

OPERATIONS:

Constructeurs:

( )----> Ensemble :ensvide

(Ensemble,Ensemble) ----> :ajelem

Opérations définies:

(Ensemble,Elément) ----> Ensemble :supelem

(Ensemble,Ensemble,Elément) ----> Ensemble:modifie

(Ensemble,Elément) ----> Bool:présent

AXIOMES:

Soit ens ∈ Ensemble ; e,e' ∈ Elément

Relations entre constructeurs:

(1) ajelem(ajelem(ens,e),e') = si eg(e,e') alors ajelem(ens,e)
sinon ajelem(ajelem(ens,e'),e)
fsi

Définitions:

(2) présent(ensvide,e)=FAUX

(3) présent(ajelem(ens,e,e')= si eg(e,e') alors VRAI
sinon présent(ens,e')
fsi

(4) supelem(ensvide,e)= ensvide

(5) supelem(ajelem(ens,e),e')= si eg(e,e') alors supelem(ens,e')
sinon ajelem(supelem(ens,e'),e)
fsi

(6) modifie(ens,e,e') = si présent(ens,e) alors ajelem(supelem(ens,e),e')
sinon ens
fsi

Remarque:

On voit apparaître dans la présentation du type Ensemble des équations conditionnelles.Cela revient à ajouter à tout type T une opération :si-alors-sinon

vérifiant les axiomes suivants:

si-alors-sinon(vrai,t,t')=t

si-alors-sinon(faux,t,t')=t'

Nous pouvons paramétrer ce type par référence (i.e. élément=référence).Les opérations et les axiomes restant inchangés. (L'opération ajouteref est ainsi définie).

Nous présentons une spécification progressive du type descripimage en procédant par extention de type et en utilisant la propriété de généricité des types abstraits.
Nous donnerons trois spécifications possibles tout en sachant qu'il peut y en avoir d'autres.

C-II-a) Premier mode de description des images : type $Descripimage_{(a)}$

La façon la plus simple pour nous de décrire une image est d'énumérer les "objets pertinents" concernant celle-ci.Elle sera ainsi représentée par un ensemble de descripteurs,chacun étant un mot-clé ou un mot-clé suivi d'une nuance.

Il nous suffit de paramétrer le type Ensemble(élément) par : descripteur (i.e : élément=descripteur).Nous appellerons le type résultant :

type $descripimage_{(a)}$

Type $Descripimage_{(a)}$,Descripteur,Requete,Bool

Constructeurs:

( ) ----> Descripimage: imvide

(Descripimage,Descripteur) ----> Descripimage: ajdes

Opérations définies:

(Descripimage,Descripteur) ----> Descripimage: supdes

(Descripimage,Descripteur,Descripteur) ----> Descripimage :modifides

(Descripimage,Descripteur) ----> Bool: présentdes

(descripimage,Requête) ----> Bool :adéquation

AXIOMES:

Ce sont les mêmesque ceux du type Ensemble(élément). Nous ne donnerons pas ici d'axiome pour l'opération adéquation, celle-ci faisant l'objet d'une étude approfondie C-II-d).

Remarque:

Cette spécification,assez simple,du type Descripimage$_{(a)}$ définit bien les objets que l'on manipule ainsi que les opérations que l'on désire leur appliquer.L'objet image est perçu comme un ensemble de descripteurs que l'opération ajdes nous permet de construire en partant de l'objet initial imvide.

Cet objet peut évoluer grace aux opérations supdes et modifides et présendes
------ --------- ---------

nous permet de voir si un objet est présent ou non.
De plus,toutes les opérations sont complétement définies,par récurrence sur les constructeurs,pour chaque objet de taille quelconque.

Dans notre définition du type Descripimage$_{(a)}$ nous ne voulons pas qu'un descripteur soit répété. C'est pour cette raison que l'axiome 5 est défini comme suit:

Soit i$\in$ Descripimage$_{(a)}$; d_1, $d_2 \in$ Descripteur
----

supdes(ajdes(i,d_1),d_2)= si eg(d_1,d_2) alors supdes(i,d_2)
-- -----
sinon ajdes(supdes(i,d_2,), d_1)
-----
fsi

Ceci peut intriguer le lecteur peu familier avec les types abstraits algébriques.Afin de clarifier ce point considérons le terme suivant:

supdes(ajdes(ajdes(ajdes(imvide,d_1), d_2), d_1), d_1)

Ce terme se réécrit par l'axiome 5 en:

supdes(ajdes(ajdes(imvide,d_1),d_2),d_1)

5
= ajdes(supdes(ajdes(imvide,d_1),d_2),d_1)

5
= ajdes(supdes(imvide,d_1),d_2)

4
= ajdes(imvide,d_2)

qui est bien le résultat qu'intuitivement on désirait.

Si l'on remplace l'axiome 5 par le suivant

(5') : Supdes(ajdes(i,d_1),d_2) = si eg(d_1, d_2) alors i
sinon ajdes(supdes(i,d_2), d_1)
fsi

On obtient un type différent qui correspond à un ensemble où les descripteurs peuvent avoir plusieurs occurrences (multiensemble) et ce n'est pas ce que nous souhaitons.

Le terme ci-dessus se réduirait par l'axiome (5') en :

supdes(ajdes(ajdes(ajdes(imvide,d1),d2),d1),d1)

(5')
= ajdes(ajdes(imvide,d1),d2)

et l'on tombe sur une forme normale différente de ajdes(invide,d2)

Etude des relations entre constructeurs :

Expliquons maintenant l'utilité de l'axiome 1 qui est une relation entre constructeurs.

soit C = {imvide,ajdes} (constructeurs primitifs)

DES = {descripteurs}

Considérons DES comme étant l'algèbre des descripteurs munie uniquement de la fonction d'égalité:

eg (d1,d1) = VRAI
eg (d1,d2) = FAUX (pour d1 ≠ d2) } Eq(DES)

T_c(DES) : algèbre des termes construits avec les constructeurs primitifs sur l'algèbre des descripteurs.

$\forall i \in T_c(DES)$ et $d \in$ DES l'opération supdes est définie de la façon suivante:

supdes (i,d) = FN <supdes(i,d)>
R ∪ Eq(DES)

R : ensemble d'axiome du type Descripimage

Eq(DES) : équations sur l'algèbre des descripteurs

FN : forme normale

Considérons les termes i = ajdes(ajdes(imvide,a),c),b) et

i' = ajdes(ajdes(ajdes(imvide,b),c),a)

(a,b,c $\in$ DES)

et appliquons leurs l'opération supdes (,b)

supdes(i,b) = FN <supdes(ajdes(ajdes(ajdes(imvide,a),c),b),b)>

axiome(5)
= FN <supdes(ajdes(ajdes(ajdes(imvide,a),c),b)>

(5)
= FN <ajdes(supdes(ajdes(imvide,a),b),c)>

(5)
= FN <ajdes(ajdes(supdes(imvide,b),a),c)>

(4)
= FN <ajdes(ajdes(imvide,a),c)>

= ajdes(ajdes(imvide,a),c)

Si nous appliquons l'opération supdes (,b) à i' on obtient :

supdes(i',b) = FN <supdes(ajdes(ajdes(ajdes(imvide,b),c),a),b)>

(5)
= FN <ajdes(supdes(ajdes(ajdes(imvide,b),c),b),a)>

(5)
= FN <ajdes(ajdes(supdes(ajdes(imvide,b),b),c),a)>

(5)
= FN <ajdes(ajdes(supdes(imvide,b),c),a)>

(4)
= FN <ajdes(ajdes(imvide,c),a)>

= ajdes(ajdes(imvide,c),a)

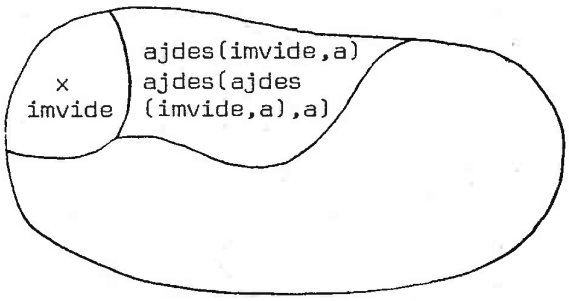
On tombe sur deux formes normales différentes mais qui pour nous sont équivalentes du fait que l'ordre d'apparition des descripteurs représentant l'image n'a pas d'importance.

C'est ce qu'exprime l'axiome (1) :

ajdes(ajdes(imvide,a),b) = si eg(a,b) alors ajdes(imvide,a)
-- -----
sinon ajdes(ajdes(imvide,b),a)
-----
fsi
---

pour (a,b) $\in$ DES.

Les objets étant multi-représentés, il nous faut définir des classes d'équivalence entre "description d'image".



supdes(,b) appliquée à deux éléments d'une même classe d'équivalence retourne 2 éléments appartenant à la même classe d'équivalence.

Ainsi, lorsque l'on passe au quotient :

$\forall$ op $\in$ {présentdes, supdes, modifiedes} op doit vérifier la condition suivante :

$$\forall (i1, i2) \in T_c(des),\ \forall\ d \in DES$$

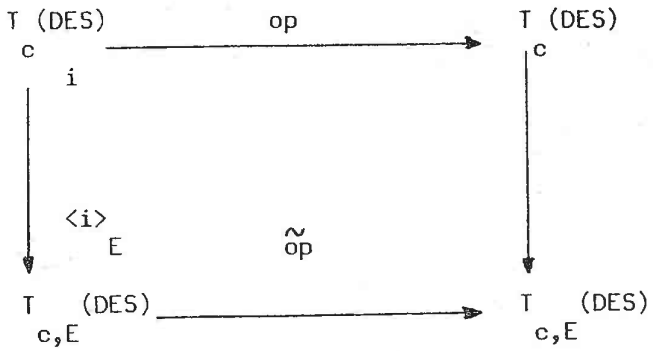
$$i1 \underset{E}{\equiv} i2 \Rightarrow op\ (i_1,d) \underset{E}{\equiv} op\ (i_2,d)$$

Ceci est une condition de consistance et on pose alors :

$$op\ (\langle I \rangle_E, d) = \langle \tilde{op}(i,d) \rangle_E$$

$\tilde{op}$: s'appliquant aux classes d'équivalence.

Nous pouvons résumer ceci par le diagramme suivant :



On associe, ainsi, aux termes de chaque classe d'équivalence une forme normale unique.

On a ainsi défini une sémantique de notre T.A en lui associant une algèbre. A chaque objet du type est associée une classe d'équivalence et les opérations sur ces objets sont définies comme précédemment.
L'algèbre ainsi définie est dite initiale de la classe de toutes les algèbres
vérifiant dès lors que l'on ne peut trouver leur égalité. (Ils seront alors dans des classes d'équivalence distinctes).

Ç-II-b) Deuxième mode de description : type Descripimage(b)

Une description plus précise des images peut être donnée en introduisant des relations entre descripteurs. (Nous nous limiterons, pour l'instant, aux relations binaires).

Pour ceci, nous procédons par extension du type Descripimage (a) en ajoutant à son ensemble de domaines la sorte : op-de-rel (opérateur de relation), ainsi que les opérations suivantes : ajrel, suprel, presentrel dont les profils sont:

(Descripimage,op-de-rel,Descripteur,Descripteur) -> Descripimage : ajrel

(Descripimage,op-de-rel,Descripteur,Descripteur) -> Descripimage : suprel

(Descripimage,op-de-rel,Descripteur,Descripteur) -> Bool : presentrel

Nouveaux axiomes :

Soit i ∈ Descripimage ; r1, r2 ∈ op-de-rel ; d1,d2,d3,d4 ∈ Descripteur

Relations entre constructeurs :

ajrel(ajrel(i,r1,d1,d2),r2,d3,d4) = si eg<(r1,d1,d2),(r2,d3,d4)> alors
ajrel(i,r1,d1,d2)
sinon ajrel(ajrel(i,r2,d3,d4),r1,d1,d2)
fsi

ajrel(ajdes(i,d1)r1,d2,d3) = ajdes(ajrel(i,r1,d2,d3),d1)

Définitions :

supdes(ajrel(i,rl,dl,d2),d3) = si eg(dl,d3) ou eg(d2,d3) alors supdes(i,d3)
sinon ajrel(supdes(i,d3),rl,dl,d2)
fsi

suprel(invide,rl,dl,d2) = imvide

suprel(ajrel(i,r2,dl,d2),r2,d3,d4) = si eg <(rl,dl,d2),(r2,d3,d4)> alors
ajrel(i,r2,d3,d4)
sinon ajrel(suprel(i,r2,d3,d4),rl,dl,d2)
fsi

presentrel(ajdes(i,dl),rl,d2,d3) = présentrel(i,rl,d2,d3)

presentdes(ajrel(i,rl,dl,d2),d3) = si eg(dl,d3) ou eg(d2,d3) alors VRAI
sinon presentdes(i,d3)
fsi

On doit cependant s'assurer que les axiomes introduits ne modifient pas l'ensemble des classes d'équivalence. <ADJ 78>
c'est-à-dire que la spécification enrichie ou étendue est suffisamment complète et consistante vis-à-vis de la spécification initiale <REM 82>.

Les propriétés que doit vérifier une spécification (S,Σ,E), extension d'une spécification (S_0,Σ_0,E_0) pour conserver la consistance et la complétude ont été données au chapitre B.

NB : Le profil de l'opération adéquation reste inchangé.

Les images peuvent être considérées comme des informations multi-dimensionnelles. Chacune des dimensions sera constituée :

- d'un nom de dimension
- d'un ensemble de descripteurs
- d'un ensemble de relations entre les descripteurs.

Remarque :

Un descripteur peut appartenir à plusieurs dimensions. Il sera alors répété. Dans ce mode de description, nous nous limiterons à des relations intérieures aux dimensions.

Ceci est un choix, mais on pourrait également, dans un 4ème mode de description, ne pas exclure les relations entre descripteurs appartenant à des dimensions différentes.

Le type $\text{Descripimage}_{(c)}$ sera une extension du type $\text{Descripimage}_{(b)}$.

Nouvelle sorte : Dimension

Nouvelles opérations :

- ajdim (constructeurs)
- supdim

C-II-c) Type $Descripimage_{(c)}$

Type $Descripimage_c$, Dimension, Descripteur, Requete, Op-de-rel, Bool

( ) ------> Descripimage : imvide

(Descripimage,dimension) ----> Descripimage : ajdim (ajoute un nom de dimension)

(Descripimage,dimension) ----> Descripimage : supdim (supprime le nom de la dimension et son contenu)

(Descripimage,dimension) ----> Bool : presentdim

(Descripimage,dimension,descripteur) ----> Descripimage : ajdes, supdes

(Descripimage,dimension,descripteur,descripteur) ----> Descripimage: modifiedes

(Descripimage,dimension,descripteur) ----> bool : Presentdes

(Descripimage,dimension,op-de-rel,descripteur,descripteur) ----> Descripimage: ajrel,suprel

(Descripimage,dimension,op-de-rel,descripteur,descripteur) ---> bool: presentrel

(Descripimage,requete) -----> bool : adéquation

AXIOMES :

Soit i $\in$ $Descripimage_{(c)}$; $dim_1, dim_2 \in$ Dimension ;

$r_1, r_2 \in$ op-de-rel; $d_1, d_2, d_3, d_4 \in$ Descripteur

presentdes(imvide,dim_1,d_1) = FAUX

presentdes(ajdim(i,dim_1),dim_2,d_2) = presentdes(i,dim_2,d_2)

presentdes(ajdes(i,dim_1,d_1),dim_2,d_2) =

si eg((dim_1,d_1),(dim_2,d_2)) alors VRAI
sinon presentdes(i,dim_2,d_2)
fsi

presentdes(ajrel(i,dim_1,r_1,d_1,d_2),dim_2,d_3) =

si eg((dim_1,d_1),dim_2,d_3)) ou
eg((dim_1,d_2),(dim_2,d_3)) alors VRAI
sinon presentdes(i,dim_2,d_2) fsi

$supdes(imvide,dim_1,d_1) = imvide$

$supdes(ajdim(i,dim_1),dim_2,d_1) =$

$ajdim(supdes(i,dim_2,d_1),dim_1)$

$supdes(ajdes(i,dim_1,d_1),dim_2,d_2) =$

si $eg((dim_1,d_1),(dim_2,d_2))$ alors

$supdes(i,dim_2,d_2)$

sinon $ajdes(supdes(i,dim_2,d_2),dim_1,d_1)$

fsi

$supdes(ajrel(i,dim_1,r_1,d_1,d_2),dim_2,d_3) =$

si $eg((dim_1,d_1),dim_2,d_3))$ ou

$eg((dim_1,d_2),(dim_2,d_3))$

alors $supdes(i,dim_2,d_3)$

sinon $ajrel(supdes(i,dim_2,d_3),dim_1,r_1,d_1,d_2)$

fsi

$presentdim(imvide,dim_1) = FAUX$

$presentdim(ajdim(i,dim_1),dim_2) =$ si $eg(dim_1,dim_2)$ alors VRAI

sinon $presentdim(i,dim_2)$

fsi

$presentdim(ajdes(i,dim_1,d_1,)dim_2) =$ si $eg(dim_1, dim_2)$ alors VRAI

sinon $presentdim(i,dim_2)$

fsi

$supdim(imvide,dim_1) = imvide$

$supdim(ajdim(i,dim_1),dim_2) =$ si $eg(dim_1,dim_2)$ alors

$supdim(i,dim_2)$

sinon $ajdim(supdim(i,dim_2),dim_1)$

fsi

supdes(ajdes(i,dim_1,d_1),dim_2,d_2) =

si eg((dim$_1$,d$_1$),(dim$_2$,d$_2$)) alors
supdes(i,dim$_2$,d$_2$)
sinon ajdes(supdes(i,dim$_2$,d$_2$),dim$_1$,d$_1$)
fsi

supdes(ajrel(i,dim$_1$,r$_1$,d$_1$,d$_2$),dim$_2$,d$_3$) =

si eg((dim$_1$,d$_1$),dim$_2$,d$_3$)) ou
eg((dim$_1$,d$_2$),(dim$_2$,d$_3$))
alors supdes(i,dim$_2$,d$_3$)
sinon ajrel(supdes(i,dim$_2$,d$_3$),dim$_1$,r$_1$,d$_1$,d$_2$)
fsi

presentdim(imvide,dim$_1$) = FAUX

presentdim(ajdim(i,dim$_1$),dim$_2$) = si eg(dim$_1$,dim$_2$) alors VRAI
sinon presentdim(i,dim$_2$)
fsi

presentdim(ajdes(i,dim$_1$,d$_1$,)dim$_2$) = si eg (dim$_1$, dim$_2$) alors VRAI
sinon presentdim(i,dim$_2$)
fsi

supdim(imvide,dim$_1$) = imvide

supdim(ajdim(i,dim$_1$),dim$_2$) = si eg(dim$_1$,dim$_2$) alors
supdim(i,dim$_2$)
sinon ajdim(supdim(i,dim$_2$),dim$_1$)
fsi

supdim(ajdes(i,dim_1,d_1) ,dim_2) = si eg(dim_1,dim_2) alors
supdim(i,dim_2)
sinon
ajdes(supdim(i,dim_2),dim_1,d_1)
fsi

supdim(ajrel(i,dim_1,r_1,d_1,d_2),dim_2) =
si eg(dim_1,dim_2) alors supdim(i,dim_2)
sinon ajrel(supdim(i,dim_2),dim_1,r_1,d_1,d_2)

presentrel(imvide,dim_1,r_1,d_1,d_2) = FAUX

presentrel(ajdim(i,dim_1),dim_2,r_1,d_1,d_2) =
presentrel(i,dim_2,r_1,d_1,d_2)

presentrel(ajdes(i,dim_1,d_1),dim_2,r_1,d_2,d_3) =
presentrel(i,dim_2,r_1,d_2,d_3)

presentrel(ajrel(i,dim_1,r_1,d_1,d_2),dim_2,r_2,d_3,d_4) =
si eg((dim_1,r_1,d_1,d_2),(dim_2,r_2,d_3,d_4)) alors VRAI
sinon presentrel(i,dim_2,r_2,d_3,d_4)
fsi

suprel(imvide,dim_1,r_1,d_1,d_2) = imvide

suprel(ajdim(i,dim_1),dim_2,r_2,d_1,d_2) =
ajdim(i,dim_2,r_2,d_1,d_2),dim_1)

suprel(ajdes(i,dim_1,d_1),dim_2,r_1,d_2,d_3)=
ajdes(suprel(i,dim_2,r_1,d_2,d_3),dim_1,d_1)

suprel(ajrel(i,dim_1,r_1,d_1,d_2),dim_2,r_2,d_3,d_4) =

si eg((dim$_1$,r_1,d_1,d_2),dim_2,r_2,d_3,d_4)) alors
suprel(i,dim_2,r_2,d_3,d_4)

sinon
ajrel(suprel(i,dim_2,r_2,d_3,d_4),dim_1,r_1,d_1,d_2)

fsi

modifiedes(i,dim_1,d_1,d_2) = si presentdes(i,dim_1,d_1) alors
ajdes(supdes(i,dim_1,d_1),dim_1,d_2)

sinon i

fsi

+ relations entre constructeurs.

Nous avons testé, à l'aide du système REVE la consistance et la terminaison finie des types Descripimage$_a$ et Descripimage$_b$ et genéré la forme normale de certains termes (voir annexe).

C-II-d) ADEQUATION D'UNE IMAGE ET D'UNE REQUETE

C-II-d.1) Type paramétré "Expression booléenne"

L'interrogation de la base se fait par le biais de requêtes qui ont la forme d'expressions booléennes dont les symboles atomiques dépendent du mode de description des images choisi.

image	requête
type Descripimage $_{(a)}$	expression booléenne dont les symboles atomiques sont des descripteurs
type Descripimage $_{(b)}$	expression booléenne dont les symboles atomiques sont des descripteurs et des relations entre les descripteurs
type Descripimage $_{(c)}$	expression booléenne dont les symboles atomiques sont des requêtes de la même forme que celles du type Descripimage$_b$

Nous allons tout d'abord donner la grammaire des requêtes et la coder ensuite sous la forme d'une spécification algébrique et définir son évaluation récursivement.

expr ::= facteur | facteur $\cup$ expr
facteur ::= primaire| primaire $\cap$ facteur
primaire ::= T $\cap$primaire | (expr)
T ::= descripteur | op-de-rel

Dans la spécification les non-terminaux seront codés par des sortes et nous définirons trois opérations d'inclusion:o,o',o''

La possibilité de définir des schémas de T.A. nous amène tout naturellement à spécifier un type paramétré permettant de construire une expression booléenne sur un ensemble arbitraire de symboles atomiques et de définir la pertinence d'une image pour une telle expression booléenne.

Type Expr(T), Facteur, Primaire, T, Image, Bool

où T a une opération CONV dont le profil est le suivant :
(image,T) : ----> bool : CONV (cette notation est inspirée du langage CLU)

OPERATIONS:

(facteur,expr) -----> expr : $\underline{\cup}$

(facteur) -----> expr : o

(primaire,facteur) -----> facteur : $\underline{\cap}$

(primaire) -----> facteur : o'

(primaire) -----> primaire : $\underline{\neg}$

(expr) ----> primaire : $\{\ \}$

(T) ----> primaire : o"

(Descripimage,expr) ----> bool :
(Descripimage,facteur) ----> bool :
(Descripimage,primaire) ----> bool
} A

AXIOMES :

Soit i $\in$ $\text{Descripimage}_{(c)}$; facteur $\in$ Facteur ; primaire $\in$ Primaire

$A(i,\underline{\cup}(primaire,facteur)) = A(i,primaire)\ AND\ A(i,facteur)$

$A(i,\underline{\neg}\,primaire) = NOT\ A(i,primaire)$

$A(i,\{expr\}) = A(i,expr)$

$A(i,o(facteur)) = A(i,facteur)$

$A(i,o'(primaire)) = A(i,primaire)$

$A(i,o"(T)) = CONV(i,T)$

La construction du type expr(T) nécessite les opérateurs booléens OR,AND,NOT
Il nous faut spécifier le type booléen dont une présentation possible est la suivante :

Type Bool :

OPERATIONS :

( ) ----> Bool : VRAI , FAUX

(Bool) ---> Bool : NOT

(Bool,Bool) ----> Bool : AND , OR

AXIOMES

Soit b , b' $\in$ Bool

NOT VRAI = FAUX

NOT NOTb = b

AND(VRAI,b) = b

AND(FAUX,b) = FAUX

AND(b,b') = AND(b',b)

OR(b,b') = NOT AND(NOTb,NOTb')

C-II-d.2) Les différentes interprétations de l'opération d'adéquation :

Pour toutes les interprétations de T, A(i, b) où b est une expression booléenne signifie que l'image i est pertinente pour b. Sa définition récursive nous permet de décomposer l'expression jusqu'aux atomes ; on lui substitue alors l'opération CONV. Interprétons effectivement cela pour les 3 types Descripimage définis précédemment.

a) type $\text{Descripimage}_{(a)}$
T = $\text{descripteur}_{(a)}$

A = adéquation_{D}

CONV = convient

Les opérations ci-dessus, (adéquation_{D} et convient), vérifient les équations décrites dans la partie axiomes du type expr.
L'opération convient à la signification suivante :

convient (i, descripteur) = VRAI ssi presentdes(i,descripteur)

Remarque :

Cette opération convient est insuffisante. Il serait intéressant de pouvoir effectuer des recherches sur des voisinages de descripteurs et élargir ainsi, comme dans les systèmes documentaires, la notion d'égalité entre éléments de requêtes et éléments de documents. Nous serons amenés à chiffrer l'éloignement entre les descripteurs et à introduire ainsi une notion de distance.
Une définition plus satisfaisante de l'opération convient pourrait être la suivante :

Soit des,des' ∈ Descripteur ; i ∈ $\text{Descripimage}_{(a)}$

convient_{S} (i,des, seuil) = VRAI ssi presentdes(i,des') et
distance (des,des') ⟨ seuil

L'opération A aurait alors plutôt le profil suivant :

(Descripimage, expr, seuil) --> bool

Nous reviendrons ultérieurement de manière plus approfondie sur cet important problème.

b) type $\text{Descripimage}_{(b)}$

Une requête, pour ce mode de description, est formée d'une expression entre descripteurs.

T = des | op-de-rel (des, des)

A = $\text{adéquation}_{(D,R)}$

∀ t ∈ T : conv(i,t) = si t = descripteur alors convient(i,t)
sinon presentrel(i, op-de-rel (des, des))
fsi

(les équations décrites dans la partie axiomes du type expr sont vérifiées par les opérations adéquation$_{(D, R)}$, convient et vérifie).

Nous pouvons formuler à propos de vérifie la même remarque que pour "convient". La définition suivante de vérifie est plus satisfaisante :

Soit d_1, d_2, d'_1, $d'_2 \in$ Descripteur ; seuil $\in \langle 0,1 \rangle$;
i $\in$ Descripimage$_{(b)}$

vérifie (i, op-de-rel (d_1, d_2), seuil) =

VRAI ssi $(d'_j)_{j = 1,2}$ dans i tq
convient$_S$ (i, d_j, seuil)$_{j = 1,2}$ = VRAI et
presentrel (i, op-de-rel, d'_1, d'_2) = VRAI

On peut entrevoir l'élargissement de la notion "d'égalité" entre deux relations

c) type Descripimage $_{(c)}$

Une requête sera, dans ce cas, une expression booléenne à deux niveaux :

1) dimensions

2) descripteur et relations entre les descripteurs à l'intérieur d'une même dimension (à ce niveau, l'expression booléenne est de la même forme que celle du type Descripimage $_{(b)}$.

Cependant nous allons distribuer le nom de la dimension pour faciliter la spécification et se ramener ainsi à une expression booléenne à un seul niveau.

- T = (dim, des) | (dim, op-de-rel (des, des))
- A = adéquation$_{(D, R)}$
- CONV(i,T)= si T = (dim, des) alors convient $_{(dim)}$(i,dim,des)
  sinon vérifie$_{(dim)}$ (i,dim,op-de-rel (des,des))

où $convient_{(dim)}$ et $vérifie_{(dim)}$ sont définies de la façon suivante:

$convient_{(dim)}$ (i, dim, des) = VRAI si presentdes(i,dim,des)

$vérifie_{(dim,)}$ (i, dim, op-de-rel (des, des')) =

VRAI ssi presentrel(i,dim,op-de-rel(des,des'))

(dim1: (d1 ∧ d2) ∧ ¬d3 ∧ dim2: r1(d4,d5) ∨ r2(d6,d7)) ∨ dim3: d8

Cette requête sera transformée en :

((dim1,d1) ∧ (dim1,d2) ∧ ¬(dim1,d3)) ∧ (dim2,r1(d4,d5)) .....

L'opération A nous permet de la décomposer récursivement jusqu'aux symboles atomiques. Suivant la valeur de ce symbole on fait appel à : $convient_{(dim)}$ ou $verifie_{(dim)}$

Différentes autres formes d'élargissement de requêtes peuvent être envisagées. Nous avons pris pour hypothèse (qui peut fort bien être remise en cause) qu'un descripteur d'image est composé d'un mot-clé et d'une nuance ou d'un mot clé tout seul. Le type descripteur est composé de l'union de deux types :

type des = type desimple ∪ type descomposé

où: desimple : mot-clé
descomposé : mot-clé + nuance

Un premier mode de déformation de requêtes est de ne pas tenir compte des nuances exprimées dans les descripteurs.

Soit d1 ∈ Desimple ; d2 ∈ Descomposé

eg(d1,d2) = VRAI si d1 et d2 ont même mot-clé

Un autre mode d'élargissement de requêtes est la transformation des opérateurs booléens AND en OR.

La suppression d'éléments de requête peut également contribuer à l'élargissement de celle-ci.(Il serait préférable de supprimer en premier lieu les relations entre descripteurs).

On peut également envisager le retrécissement de requêtes bien que ceci semble-plus difficile à mettre en oeuvre.

L'élargissement d'une requête ne doit cependant pas "trop" trahir l'objectif premier de l'utilisateur.L'introduction d'un contexte pour une requête donnée devrait permettre au système de ne pas trop s'écarter de la requête initiale. La solution la plus simple serait d'interdire pour certains élément de requête, toute déformation.

Il existe des systèmes documentaires tel que SPIRIT <SPI 84>(systéme syntaxique et probabiliste d'indexation et de recherche d'informations textuelles) permettant d'élargir une requête.Le critère de sélection de documents répondant à une requête est le suivant:

pour chaque document qui a un nombre suffisant de mots en communs avec une requête on calcule une pondération qui est d'autant plus forte qu'il y a plus de mots en communs.Les documents sont classés par ordre décroissant des pondérations et listés dans cet ordre avec les mot-clés commun à la question. La pondération de chaque mot est une fonction de poids sémantique qui mesure son pouvoir informationnel.
Cette forme d'élargissement correspond à la suppression d'élément de requête et à l'attribution d'une pondération à chaque mot-clé.(Cette pondération n'a pas été prise en compte dans notre spécification).

C-III) ECHANTILLONNAGE:

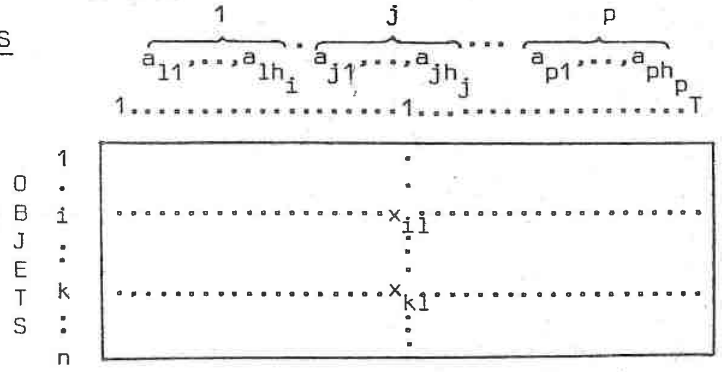
Lorsque le nombre de références d'images sélectionnées par une requête est trop grand,le temps de visualisation de toutes les photos serait trop long. La solution la plus simple pour parer à ceci serait de ne visualiser qu'une image sur n.Une solution plus adéquate serait d'appliquer des techniques de classification automatique.On décompose ainsi une population d'images,décrites par un ensemble de caractéristiques,en un certain nombre de sous-groupes homogènes et on visualise un représentant de chaque sous-groupe.
Nous allons décrire une méthode de classification automatique et l'appliquée à notre problème.

I) Description de la méthode:<CHA 81>

On se place dans le cas où n objets sont décrits par p variables nominales. Chaque variable nominale j,souvent appelée attribut,présente un ensemble de modalités: $A_j = \{a_1, \ldots\ldots, a_n\}$ .Le nombre de modalités peut varier d'un attribut à l'autre.
Un codage des objets est une application de l'ensemble des objets dans le produit cartésien des attributs.On décide par convention de coder un objet par une suite de variables logiques a_{ijh} qui peut prendre la valeur 1 si l'objet i possède la modalité h du jième attribut et la valeur 0 sinon.Le nombre total de valeurs d'attributs est donné par:

$$T = \sum_{j=1}^{p} |A_j|$$

La matrice de données se présente sous la forme d'un tableau à n lignes et T colonnes où la case d'indice (i,l) contient un 1 si l'objet i possède la modalité l et la valeur 0 sinon.



Matrice logique ou tableau de codage disjonctif

La matrice des données logiques est appelée tableau de codage disjonctif ou tableau de présences-abscences.
Ce dernier définit quatre conditions générales auxquelles tout bon indice de similarité se devrait de satisfaire.

1) La contribution d'un attribut à la mesure de similarité entre deux objets doit être la même pour tous les attributs. Cette condition résulte d'un parti pris raisonnable. Le chercheur doit introduire tous les attributs qu'il estime pertinents pour sa classification,mais toute pondération à priori ne pourrait que résulter d'une idée préconçue de la classification à obtenir.

2) La contribution d'une association négative,mesurée par co-absence de deux attributs,doit être au plus égale à la contribution positive,mesurée par la co-présence de deux attributs.
Cette condition postule que la co-absence d'attributs est moins informative que la co-présence d'attributs. Il est clair que si la description des objets inclut beaucoup d'attributs non possédés par la majorité des objets,la co-absence gonflera la similarité entre les objets et diluera du même coup les différences existant au niveau des co-présences.

3) La similarité entre deux objets doit être une fonction croissante du nombre de co-présences d'attributs et une fonction décroissante du nombre de non-coincidences d'attributs.
Ceci est la définition même du concept de similarité: plus les objets ont des attributs en communs et plus ils sont similaires.Plus nombreux sont les désaccords et moindre est la ressemblance.

4) La similarité entre deux objets doit être une fonction symétrique du nombre d'attributs exclusifs de chacun des deux objets.
On ne peut concevoir un indice de similarité où les attributs exclusifs de l'objet x se verraient attribuer un role différent de celui des attributs exclusifs de l'objet y ,car la similarité entre x et y doit être identique à la similarité entre y et x.

Il existe plusieurs indices permettant de mesurer la similarité (proximité) entre deux objets.Il convient cependant de noter que les deux objets peuvent avoir ou non le même nombre d'attributs.Il est prouvé que les différents indices donneront des résultats d'autant plus divergents que la variance du nombre d'attributs sera grande.

Quelques indices de proximité définis sur les matrices logiques:

Nous désignerons par P le nombre de co-présences,par A le nombre de non-coincidences et par T le nombre total d'attributs.
Nous appellerons X l'ensemble des attributs possédés par l'objet x et XE l'ensemble des attributs exclusifs de l'objet x.

Exemple:
---

x 0 0 1 0 0 1 1

y 1 0 0 0 0 1 1

T=7 , XE=1 , YE=1 , N=XE+YE

Indice de RUSSEL et RAO proximité(x,y)= P/T

" OCHIAI " = $P/\sqrt{X \cdot Y}$

" JACCARD " = P/P+N

" YULE " = (P)(A)-(XE)(YE)/(P)(A)+(XE)(YE)

II) Constitution des groupes:

La connaissance des proximités entre paires d'objets n'est pas "lisible",pas plus que ne l'est la matrice des données.
Il s'agit de remplacer cette information par une autre plus synthétique et plus compréhensible en faisant apparaitre un petit nombre de groupes d'objets.L'objectif de toute classification est de simplifier l'information contenue dans le tableau des proximités entre paires d'objets par des proximités entre groupes d'objets.
Il existe de très nombreuses méthodes de classification décrites dans <CHA 81>.

III) Adaptation de la méthode à notre problème:

Notre but est de partitionner en un certain nombre de sous-groupes un ensemble de descriptions d'images sélectionnées par une requête.La classification devra se faire le plus souvent sur des descripteurs non exprimés dans la requête.

Le type $descripimage_{(c)}$ se prête bien à la méthode décrite ci-dessus en prenant comme attributs les dimensions de l'image et comme modalités les valeurs que peuvent prendre les descripteurs dans ces dimensions. Chaque ligne de la matrice sera une description d'image avec la valeur 1 si la modalité est présente dans la dimension et 0 sinon.

Le type $descripimage_{(c)}$ se prête bien à la méthode décrite ci-dessus en pre-
trice sera une description d'image avec la valeur 1 si la modalité est présente dans la dimension et 0 sinon.

Nous pouvons définir un type Matlog (matrice logique) avec les opérations suivantes: assigne,accès,distance,co-présence

Type Matlog,Index,Valeur,Descripimage,Dimension,Descripteur, ℕ, ℝ

Opérations:

( ) -----> Matlog : matvide

(Matlog,Index,Index,Valeur) ----> Matlog : assigne

(Matlog,Index,Index) -----> Valeur : accès

(Matlog,Index,Index,Index) ----> ℝ : distance

(Matlog,Index,Index,Index) ----> ℕ : co-présence

Axiomes:

Soit m ∈ Matlog; i,j,i',j' ∈ Index ; v ∈ Valeur ;dim ∈ Dimension;des ∈ Descripteur
d ∈ Descripimage

assigne(m,i,j,v)= si $présent_{dim}$(d,dim,des) alors v=1 et m(i,j)=v
fsi SINON m(i,j)=0

accès(matvide,i,j)=undef

accès(assigne(m,i,j,v),i',j')= si eg((i,j),(i',j')) alors v
sinon accès(m,i',j')
fsi

distance(m,i,j,k)= div(co-presence(m,i,j,k),T)

co-presence(m,i,j,k) = si eg(accès(m,i,k),accès(m,j,k)) alors P=P+1 et
co-presence(m,i,j,succ(k))

sinon co-présence(m,i,j,succ(k))

fsi

Ceci pour pour l'indice de RUSSEL et RAO.

L'opération présentdim est définie dans le type Descripimage$_{(c)}$, la dimension

L'opération de constitution des groupes aura pour paramètres la matrice des distances et le critère de classification.
Cette petite étude sur l'échantillonnage n'est pas assez approfondie (l'objet de cette thèse n'étant pas la classification automatique) mais elle permet d'entrevoir la possibilité d'adapter à notre problème les techniques utilisées en analyse typologique.

CHAPITRE D

SPECIFICATION D'UN SYSTEME EXPERT A L'AIDE DE TYPES ABSTRAITS ALGEBRIQUES

CHAPITRE D

SPECIFICATION D'UN SYSTEME EXPERT A L'AIDE DE TYPES ABSTRAITS ALGEBRIQUES

Après s'être interessés à la description de mécanismes généraux modélisant le raisonnement humain, les chercheurs en Intelligence Artificielle se sont intéressés à la construction de systèmes utilisant une grande quantité de connaissances dans des domaines spécifiques. Ceci a conduit au développement de systèmes experts (S.E.) simulant la démarche des experts humains (E.H.) confrontés aux problèmes posés dans ces domaines.

D.I.1) MOTIVATIONS POUR DEVELOPPER DES S.E <Bonnet 80>

Lorsque des informaticiens et des experts d'un sujet se réunissent pour construire un S.E., ils n'ont pas pour but de remplacer à plus ou moins long terme les E.H. Les motivations principales pour construire des S.E. sont :

i) Aider des non-experts d'un domaine (les utilisateurs) à résoudre des problèmes soit nécessitant beaucoup de connaissances très spécifiques, soit présentant un caractère fastidieux.

ii) Essayer de comprendre les mécanismes de raisonnement humain. Un E.H. a généralement des difficultés à exposer la façon dont il raisonne, soit parce qu'il y a un niveau <<boîte noire>>, soit parce qu'il y a une réticence à livrer son savoir et surtout son savoir faire. L'exercice consistant à transférer son expertise vers un programme permet, par un phénomène de <<retour arrière>> d'améliorer, voire de rendre plus rigoureux ses processus de raisonnement.

iii) D'un point de vue plus étroitement informatique, le développement de S.E. permet de chercher des moyens de plus en plus adéquats pour représenter les connaissances en I.A. et de mieux formaliser les mécanismes de raisonnement afin de les rendre (autant que possible) utilisables dans divers domaines.

iiii) Par rapport aux experts humains irremplaçables puisqu'ils sont la source de la connaissance introduite dans les S.E., ceux-ci ont l'avantage d'être duplicables, de pouvoir fonctionner jour et nuit et d'être toujours identiques à eux-mêmes.

D.I.2) CARACTERISTIQUES DES S.E.

Les S.E. possèdent généralement une base de connaissances spécifiques du domaine à laquelle est associée une structure dite de controle interprétant la base de connaissances et l'appliquant aux données des problèmes en vue de leur résolution. Lors de cette résolution il faut que le système soit compris de l'utilisateur (spécialiste ou non-spécialiste) et qu'au besoin il explique et justifie son raisonnement.
La base de connaissances est la clé d'un S.E. Celui-ci ne produira de résultats intéressants que si elle contient pour un domaine donné des connaissances aussi précises,aussi justes, aussi complètes et détaillées que celles que possède un expert du domaine.

Il faut également que le spécialiste (expert) puisse aisément transmettre ses connaissances et qu'au besoin plusieurs experts puissent enrichir chacun le système de leurs connaissances propres dans la spécialité. Ceci impliéque que cette connaissance puisse être entrée sans préjuger de l'utilisation qui en sera faite, donc en vrac et de façon purement déclarative.

D.I.3) L'OPPOSITION DECLARATIF/PROCEDURAL <LAU 82 - parties 1 et 2>

Cette controverse est vraiment née avec les premiers succès des S.E. utilisant une base de connaissances mise sous forme de règles de production du type :

S ---> C

Situation conclusion

Dans cette approche des S.E., dite approche <<déclarative>>, l'accès à l'information est alors obtenu par unification ou unification restreinte (dite <<pattern matching>> ou <<filtrage>>).

Dans l'approche procédurale, au contraire, il faut prévoir les questions a priori et écrire une procédure pour chacune d'entre elles.

Bien qu'ils présentent certains inconvénients sur lesquels nous reviendrons,nous préférons, pour notre part, les S.E. avec règles de production pour les avantages suivants qu'ils présentent.

. Modularité :

Chaque unité d'information peut, ici, par construction, être ôtée, changée, ajoutée sans qu'il soit besoin de modifier les autres.
Les règles doivent être indépendantes les unes par rapport aux autres. Une règle ne pouvant faire référence à une autre, la base est ainsi constituée d'un ensemble granulaire de connaissances.

. Modifiabilité :

Si l'on ajoute ou si l'on modifie une règle, tout ce qui avait été fait reste valide tant que la situation présente ne concerne pas la nouvelle règle. Tout changement est additif et local. Le système tout en sachant plus sait différemment. Dans les programmes procéduraux en revanche, les intéractions étant très étroites, toute modification est difficile et peut se répercuter de façon incontrolée.

. Faculté d'auto-explication:

Le mode de raisonnement sous-jacent des systèmes qui utilisent comme base de connaissances des règles de production est le modus ponens (de p et p => q on conclut q). Le système remonte facilement la chaîne des modus ponens qu'il a utilisée pour raisonner.

Cependant il est à noter que les systèmes de production présentent les difficultés suivantes :

- Difficulté de concevoir une règle de production correspondant à un élément de connaissance. Il faut que le domaine concerné ait déjà été suffisamment étudié pour que des bonnes primitives aient été dégagées et que le niveau de détail ne soit pas trop fin. Sans ce présupposé, il faudrait en effet une règle pour chaque situation et on voit mal comment l'homme pourrait transmettre ce savoir informe.

Puisque, par construction, toute information utile doit figurer dans la partie S (situation) des règles, il faut, afin de ne pas alourdir démesurément les membres gauches, imposer une autre condition : que les éléments intervenant dans un même membre gauche ne soient pas fortement liés.Ceci signifie qu'ils doivent posséder des comportements assez indépendants les uns des autres.

- Difficulté d'écrire une règle : Le format unique <<si S alors C>> entraîne une certaine lourdeur dans les expressions des membres gauches et une certaine répétition des mêmes prémisses pour des situations semblables.

D.II) DESCRIPTION D'UN SYSTEME DE PRODUCTION

Une règle de production est une expression de la forme :

si P_1 λ_1 P_2 ... λ_j P_i alors A_1 λ_1 A_2 ... λ_2 A_i

. Les P_i sont appelés prémisses ou conditions de la règle.

. λ_i : opérateurs booléens (dans la plupart des S.E. λ_i= ET)

. Les A_i sont les conclusions à tirer si la situation S est détectée, chaque A_i pouvant être soit l'affirmation d'un fait, soit éventuellement le déclenchement d'une procédure.

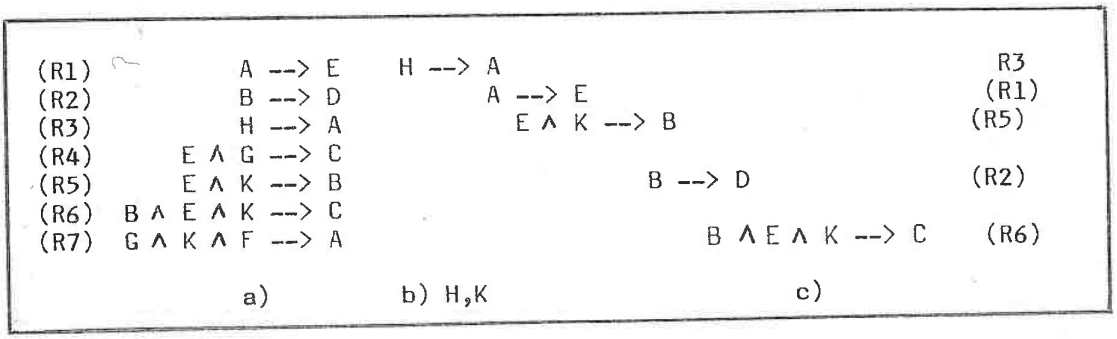
Ce sont les règles de production qui sont chargées de supporter toute la connaissance. Cette connaissance est ainsi donnée de façon modulaire et aisément modifiable puisqu'elle n'est pas mélée au corps des programmes qui l'utilisent.

Le système global, outre sa banque de connaissance qui est l'ensemble des règles de production, se compose d'un espace de travail (ou base des faits) et d'un interprète.L'espace de travail contient l'ensemble des faits que le programme a pu déduire à un instant donné. Cet espace de travail joue le rôle d'une <<mémoire à court terme>> qui contient les opérateurs des transformations légales sous forme de règles. Ces règles peuvent contenir des variables dont les valeurs sont <<instanciées>> lors de chaque exécution pour tenter de valider un fait donné par <<unification>> (ou filtrage).

L'unification (ou le filtrage) est ainsi le mécanisme fondamental qui permet aux systèmes de production de fonctionner. Dans toute sa généralité il permet de dire s'il existe un jeu de substitutions des variables qui permet de rendre deux formules logiques identiques.

L'"interprète" est le coeur du S.E. Alimenté par une base de connaissances,il construit dynamiquement une solution,décidant quelles règles de production déclencher et dans quel ordre. Il utilise pour cela sa mémoire de travail, ou base des faits,dans laquelle les informations décrivant la situation initiale (le problème posé) et celles qui sont déduites au cours du raisonnement sont conservées.

Exemple simple de raisonnement dirigé par les données au moyen de règles de production : <LAURIERE 82>



a) Règles
b) Base des faits initiale
c) Chaîne de dérivation

Le raisonnement se fait en utilisant les règles à partir des faits connus (au départ H et K). Ainsi de H on peut déduire A par la règle (R3), E par la règle (R1), B par (R5) etc....
Lorsqu'une règle est employée,les nouveaux faits sont ajoutés à la base des faits.Ce type de raisonnement est dit "chaînage avant".
Nous reviendrons plus en détail sur ceci.

Cette structure des systèmes experts,séparant les connaissances des mécanismes de contrôle,s'oppose à l'approche la plus classique en programmation. Lorsqu'un spécialiste conçoit un logiciel capable de résoudre un certain type de problèmes,ses connaissances sont inscrites dans les instructions de son programme,et il est obligé de définir par avance l'enchaînement de leur utilisation au cours de l'exécution de son programme. De ce fait les programmes "classiques" ne sont capable de résoudre que des problémes toujours posés de la même façon.De plus la prise en compte d'une nouvelle connaissance passe par la modification du programme,dans le meilleur des cas, et souvent par sa réécriture. Au contraire,dans les systèmes experts utilisant de la connaissance mise sous forme de règles de production, la solution du problème ne passe pas par l'exécution d'une séquence d'instructions dans un ordre prévu à l'avance. Si le moteur d'inférence est bien un programme dans le sens habituel , l'ordre dans lequel il utilise les connaissances de la base n'a été déterminé par aucun programmeur.

D.III) ROLE DE L'INTERPRETE DANS UN SYSTEME EXPERT

L'interprète est la partie clé du système qui contrôle complètement l'ordre des déductions successives. Cependant il faut distinguer deux stratégies générales de recherche possibles:

- Le chaînage avant : On part des faits connus donnés par l'expert ; on déclenche toute règle dont les prémisses sont satisfaites ; on ajoute les faits conséquents, on recommence jusqu'a <<saturation>> du système (plus de nouvelles déductions).

- Le chaînage arrière : En supposant que l'on cherche une réponse à une question de l'utilisateur et que le nombre de réponses possibles soit fini. On considère la première d'entre elles : c'est le but. On regarde toutes les règles dont le conséquent exprime une affirmation contenant le but. S'il n'y en a pas on passe au but suivant. Sinon chacune de ces règles est considérée tour à tour si toutes ses prémisses sont satisfaites dans la base des faits, le but est atteint, sinon on enregistre les prémisses inconnues comme autant de nouveaux buts et on recommence le cycle. On remonte ainsi de la question aux faits initiaux. Le système aura en outre le droit de poser des questions à l'expert à chaque fois qu'il aura besoin, en cours de raisonnement, d'un fait non donné et non déductible par règle.

Il existe également une stratégie de recherche combinant le chaînage avant et le chaînage arrière et que l'on appelle chaînage mixte.

Cependant il faut noter que les systèmes de production diffèrent entre eux non seulement en ce qui concerne la stratégie de recherche adoptée mais également sur les points suivants et bien d'autres.

- La mise à jour de la base des faits.
- La présence de variables dans les règles.
- Le recours ou non aux procédures. Si un S.E. possède en membre droit de certaines règles une procédure, il est dit "ouvert" (sinon il est dit "fermé"). Cependant, pour conserver tout l'intéret du S.E., il est alors nécessaire que de telles procédures ne s'appellent jamais mutuellement.
- La structure de contrôle qui concerne <<la résolution de conflit>> entre toutes les règles candidates au déclenchement. Nous reviendrons sur ce point dans la suite de notre travail.

Tenant compte de ces différences qui ne sont pas les seules,nous n'hésiterons cependant pas dans certains cas à faire des choix, lorsque cela permettra d'illustrer de façon déterminante l'intéret d'une spécification algébrique sur une classe restreinte de S.E. plutot que de rester à un trop grand niveau de généralités.(Afin de mieux comprendre toutes ces différences nous travaillons actuellement à la typologie des S.E.)

D.IV) SPECIFICATION A L'AIDE DE T.A.A. D'UN SYSTEME A REGLES DE PRODUCTION

Nous nous intéressons tout d'abord aux S.E. utilisant pour stratégie de recherche le chaînage arrière.

Cette stratégie de résolution est celle utilisée dans le langage PROLOG, conçu par A. COLMERAUER <ROU 75> <CHA 82> <ROB 65>. Certains systèmes experts sont d'ailleurs réalisés en PROLOG.
Il faut cependant remarquer que, à la différence de PROLOG, lors de notre résolution nous pouvons faire appel à l'utilisateur. La résolution d'un problème peut donc se faire de manière interactive en établissant un dialogue homme-machine.

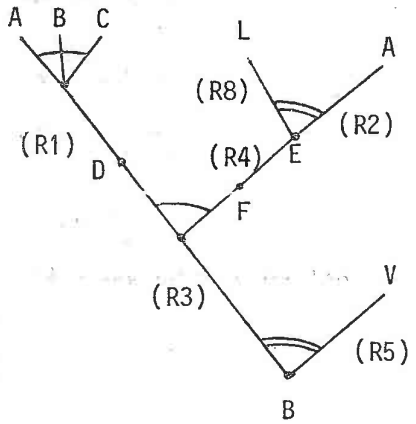
Dans la spécification que nous donnons nous ne nous intéressons qu'aux interprètes (moteurs) d'ordre zéro, c'est-à-dire n'utilisant pas de variables.

En chaînage arrière, on suppose que l'on cherche une réponse à une question de l'utilisateur. La recherche du but peut être représentée à l'aide d'un arbre ET/OU.

Exemple :

(R1) A et B et C --> D
(R2) A --> E
(R3) D et F --> B
(R4) E --> F
(R5) V --> B
(R6) I --> Z
(R7) K et M --> Y
(R8) L --> E

ensembles de règles de production



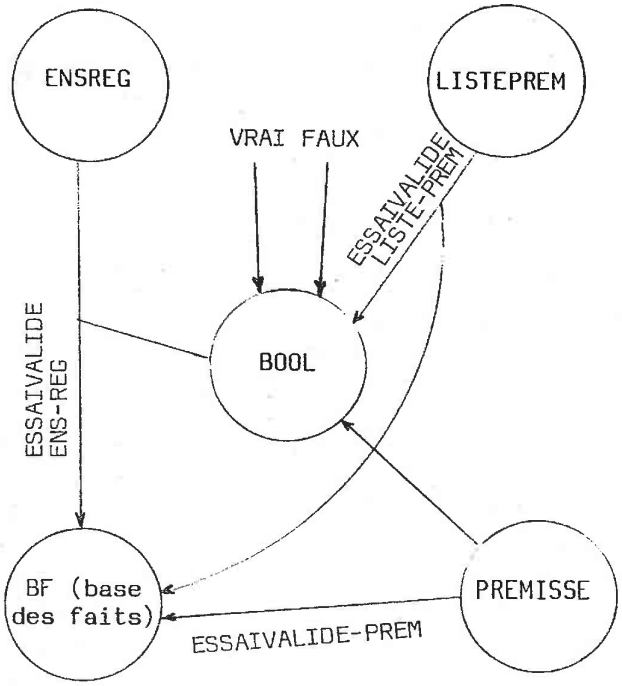
ET
OU

Construction de l'arbre ET/OU si l'on désire atteindre le but B avec la base de connaissance ci-dessus.

D.IV.1) Types et opérations intervenant dans la spécification

Nous allons donner une description formelle à l'aide de T.A. d'un S.E. utilisant pour principe de résolution le chainage arrière.Dans cette spécification nous faisons un parcours profondeur d'abord,de plus on s'arrête dès qu'une règle ayant le but à atteindre en partie conclusion a été validée. Cette façon de procéder n'est pas la seule et nous reviendrons sur ceci.

Le schéma ci-dessous représente les principaux types et les opérations les plus importantes.



Ce schéma fait apparaître trois opérations :

- essaivalide-ens-reg
- essaivalide-liste-prem
- essaivalide-prem

qui renvoient chacune un résultat de type produit cartésien à deux éléments.

- un de type booléen : Résul
- un de type base des faits (Bf).

Au cours de la résolution le système peut effectuer de nouvelles déductions qui viendront enrichir la base des faits. C'est pour tenir compte de l'évolution de la base des faits que les 3 opérations possèdent en résultat le type BF.
Le booléen nous indiquera si la recherche du but à atteindre s'est soldée par un échec ou par un succès.

D.IV.2) Roles des opérations essaivalide-ens-reg ; essaivalide-liste-prem ; essaivalide-prem :

- essaivalide-ens-reg :

Cette opération agit sur l'ensemble des règles qui ont le but à atteindre dans leur partie conséquent et essaie de valider une de ces règles. Cet ensemble de règles constitue en quelque sorte l'ensemble des régles candidates.
Cette règle est définie dans le type Ensreg (ensemble de règles) et a pour profil :

(Ensreg,But,Bf) --> (Resul,Bf)

- essaivalide-liste-prem :

Cette opération agit sur l'ensemble des prémisses d'une règle et essaie de valider ses prémisses. Elle est définie dans le type listeprem (liste de prémisses). et son profil est

(Listeprem,Ensreg,Bf) --> (Resul,Bf)

- essaivalide-prem :

Cette opération essaie de valider une prémisse ; son profil est :

(Prémisse,Ensreg,Bf,Utilisateur) --> (resul,Bf)

D.IV.3) Spécification algébrique des types : Ensreg ; Listeprem ; Prémisse ; Bf

Remarque :

Une règle sera représentée par le produit cartésien :

(Listeprem, Conséquent)

S C

a) Type Ensreg (ensemble de règles)

type Ensreg,Listeprem,Conséquent,But,Bool,Bf

Opérations :

( ) --> Ensreg : ensregvide

(Ensreg,(Listeprem,Conséquent)) --> Ensreg : ajrègle

(Ensreg,(Listeprem,Conséquent)) --> Ensreg : suprègle

(Ensreg,(Listeprem,Conséquent),(Listeprem,Conséquent) --> Ensreg : modifregle

(Ensreg,But,Bf) --> (Resul,Bf) : essaivalide-ens-reg

Axiomes :

Nous ne donnons pas la définition des axiomes sur suprègle, modifrègle qui sont les axiomes classiques du type Ensemble.

Définition de l'opération : essaivalide-ens-reg :

Soit e ∈ Ensreg ; lisp ∈ Listeprem ; conseq ∈ Conséquent ; b ∈ But ; bf, bf' ∈ Bf.

essaivalide-ens-reg(ensregvide,b,bf) = (FAUX,bf)

essaivalide-ens-reg(ajregle(e,(lisp,conseq)),b,bf) =

```
si ¬(eg(b,conseq)) alors essaivalide-ens-reg(e,b,bf)
--                    -----
   sinon  si essaivalide-liste-prem(Lisp,e,bf) = (FAUX,bf')
   -----  --
          alors  essaivalide-ens-reg(e,b,bf)
          ------
              sinon (VRAI,ajfait(bf',b))
              ------
          fsi
fsi       ---
---
```

Remarques :

- L'opération ajfait de profil : (Bf,Fait) --> Bf est définie dans le type Bf (Base des faits) que nous verrons plus loin.

- Les prémisses et les faits doivent être de même type pour que l'unification soit possible.

- Pour valider une règle il faut que toutes ses prémisses soient vérifiées. Dans le cas contraire RESUL aura la valeur FAUX et le but ne peut être atteint par la règle courante. L'opération qui tente de valider les prémisses d'une règle est essaivalide-liste-prem et elle est définie dans le type Listeprem dont la spécification est la suivante :

b) Type Listeprem :

type Listeprem,Bf,Ensreg,Prémisse,Bool

Opérations :

( ) --> Listeprem : listepremvide

(Listeprem,Prémisse) --> Listeprem : ajprèmisse

(Listeprem,Ensreg,Bf) --> (Resul,Bf) : essaivalide-liste-prem

Axiomes :

Soit lisp ∈ Listeprem ; bf,bf' ∈ Bf ; e ∈ Ensreg ; p ∈ Prémisse

essaivalide-liste-prem(listepremvide,e,bf) = (FAUX,bf)

essaivalide-liste-prem(ajprémisse(lisp,p),e,bf) =

si essaivalide-prem(p,e,bf) = (VRAI,bf') alors (VRAI,bf') et
essaivalide-liste-prem(lisp,bf',e)

sinon (FAUX,bf')
fsi

Une prémisse peut être validée de trois façons :

- par examen direct de la base des faits

- par déduction récursive à l'aide de l'opération essaivalide-ens-reg

- par question directe à l'utilisateur chaque fois que le système aura besoin, en cours de raisonnement d'un fait non donné et non déductible par règle. Cependant nous n'autoriserons pas cela pour tous les faits et il faudra que celui-ci appartienne à l'ensemble des propositions demandables.

Dans le type Prémisse, il nous est utile de définir un prédicat permettant de tester si un fait appartient ou non à l'ensemble des propositions demandables ainsi qu'un prédicat donnant la réponse de l'utilisateur.

c) Type Prémisse :

Type Prémisse,Ensreg,Bf,Utilisateur,Bool

Opérations :

(Prémisse) --> Bool : demandable

(Prémisse,Utilisateur) --> Bool : valeur-rep-u

(Prémisse,Ensreg,Bf,Utilisateur) --> (Resul,Bf) : essaivalide-prem

Axiomes :

Soit p ∈ Prémisse ; bf,bf' ∈ Bf ; e ∈ Ensreg : u ∈ Utilisateur

```
essaivalide-prem(p,e,bf,u) = si présent(bf,p) alors (VRAI,bf)
                             --                -----
                               sinon  si présent(bf,  p) alors (FAUX,bf)
                               -----  --                -----
                                     sinon si essaivalide-ens-reg(e,p,bf') =
                                     ----- --
                                              (VRAI,bf') alors (VRAI,bf')
                                                         -----
                                                sinon  si (demandable(p) et valeur-
                                                -----  --                 --
                                                           rep-u)

                                                       alors (VRAI,ajfait(bf',p))
                                                       -----

                                                       fsi
                                                       ---
                                             fsi
                                             ---

                                  fsi
                                  ---
```

Remarque :

La base des faits (Bf) contiendra la liste des affirmations que le système aura déduites au cours de sa recherche. Au départ la Bf est initialisée par :

- l'ensemble des faits toujours VRAI (mémoire à "long terme")

- l'ensemble des faits connus et donnés par l'utilisateur et qui seront locaux à la session en cours (mémoire à "court terme").

d) Type Bf
---

type Bf, Fait, Bool

Opérations :

( ) --> Bf : bfvide

(Bf,Fait) --> Bf : ajfait

(Bf,Fait) --> Bf : supfait

(Bf,Fait) --> Bool : présent

Les axiomes sont ceux du type Ensemble

L'initialisation de la base des faits se fera par le biais d'une opération d'union entre les faits toujours VRAI et ceux qui ne sont vérifiés que lors de la session en cours.

La gestion du but initial et des éventuels sous-buts intermédiaires à atteindre se fera au moyen d'une pile.

Nous ne donnons pas la spécification bien connue d'une pile. Nous dirons seulement que l'opération dépiler s'appliquera à tous les buts validés ou quand il sera prouvé qu'ils ne seront jamais atteints. Dans ce dernier cas, cela signifie que la règle courante que l'on traite ne permet pas d'atteindre le but.

Au départ on empilera le but initial et si celui-ci est atteint on se retrouvera avec pilevide? = VRAI.

Remarques :

- Le système est capable, par le biais de la pile des buts de garder une trace de son raisonnement et d'en faire l'historique à l'utilisateur si celui-ci le désire.

- Souvent pour éviter les risques de bouclages, nous n'empilerons pas un but déjà présent dans la pile.

Nous avons donné une spécification algébrique d'un moteur d'inférence utilisant un mode de recherche en chaînage arrière. Il serait intéressant de pouvoir adapter une stratégie combinant le chaînage avant et arrière (chaînage mixte)

D-IV-4) MOTEUR FONCTIONNANT EN CHAINAGE MIXTE:

Rappel :

En chaînage avant, on part des faits connus donnés par l'utilisateur ; on déclenche la règle choisie parmi l'ensemble des règles candidates : on ajoute les faits conséquents : on recommence jusqu'à ce que le but soit atteint ou jusqu'à "saturation" du système (plus de nouvelles déductions).

Le cycle de base d'un interprète fonctionnant en chaînage avant consiste à :

a) Déterminer l'ensemble des règles candidates.

b) Choisir la règle à déclencher.

c) Exécuter la partie conclusion de la règle choisie et mettre à jour la base des faits.

Nous verrons à quel moment nous pouvons faire appel au chaînage arrière lorsque l'on adopte au départ une stratégie en chaînage avant.

La première étape est donc la détermination de l'ensemble des règles candidates. Nous dirons qu'une règle est candidate si :

- toutes ses prémisses sont validées,

- un sous-ensemble de ses prémisses est validé et aucune prémisse n'est contredite par la base des faits.

La sélection des règles candidates nécéssite la confrontation de chacune d'entre elles avec l'ensemble des faits de la base . Cette opération peut devenir rapidement très coûteuse (elle est répétée à chaque cycle). L'utilisation de stratégies peut réduire sensiblement l'explosion combinatoire. Ce point est étudié au paragraphe D-VI.

Lorsqu'une règle candidate n'aura qu'un sous-ensemble de ses prémisses non vérifié, le système pourra tenter de valider les autres en faisant appel au chaînage arrière. Il suffira pour ceci de faire appel à l'opération essaivalide-ens-reg avec les paramètres suivants :

(Ensreg,p,bf) où p est une prémisse non validée de la règle candidate.

La détermination de l'ensemble des règles candidates se fera au moyen de l'opération "candidate" dont le profil est le suivant :

(Ensreg,bf) --> Enscouple(Règle,Indicateur)

(Le co-domaine de l'opération "candidate" est de type Ensemble paramétré par (Règle,Indicateur))

Remarque:

Si l'on ne désire que déduire de l'information sans vouloir atteindre un but précis, le profil de l'opération "canditate" sera:

(Ensrègle,Bf) -----> Enscouple(Règle,Indicateur)

Le critère d'arrêt est obtenu lorsque aucune nouvelle déduction ne peut être inférée.

On enrichit donc le type Ensrègle par l'opération "candidate" qui nous délivrera un ensemble de règles chacune étant munie d'un indicateur pouvant prendre les valeurs suivantes :

- SUR : si toutes les prémisses de la règle candidate sont validées.
- PEUTETRE : si un sous-ensemble propre des prémisses est validé.

Définition de l'opération "candidate".

Soit e ∈ Ensreg ; lisp ∈ Listeprem ; conseq ∈ Conséquent ; bf ∈ Bf

```
Candidate(ensvide,bf) = ∅

Candidate(ajregle(e,(lisp,conseq)),bf) =

                si eg(valide(lisp,bf),TOUTPRESENT) alors
                --                                 -----
                   ajcouple(candidate(e,bf),((lisp,conseq),SUR))

                     sinon  si eg(valide(lisp,br) , UNFAUX) alors
                     -----  --                              -----
                            candidate(e,bf)

                            sinon ajcouple(candidate(e,bf),((lisp,conseq),
                            -----          PEUTETRE))

                            fsi
                            ---
                fsi
                ---
```

L'opération "valide" nous indique si toutes les prémisses d'une règle sont validées dans la base des faits. Si oui, on ajoute le couple <(lisp,conseq),SUR> à l'ensemble des règles candidates. Si seul un sous-ensemble des prémisses est validé on ajoute le couple <(lisp,conseq),PEUTETRE> à l'ensemble des règles candidates. Si une des prémisses est prouvée FAUX on élimine la règle. (Une règle peut être candidate sans qu'aucune de ses prémisses ne soit validée, pourvu qu'aucune ne soit prouvée FAUX. Dans ce cas la valeur de son indicateur sera PEUTETRE).

On enrichit le type Listeprem des opérations "valide" et "validepeutetre" dont le profil est le suivant :

(Listeprem,Bf) --> {TOUTPRESENT,UNFAUX,PEUTETRE} : valide

(Listeprem,Bf) --> {UNFAUX,PEUTETRE} : validepeutetre

Definition de l'opération "valide":

Soit lisp ∈ Listeprem : bf ∈ Bf ; p ∈ Prémisse
---

valide(listepremvide,bf) = TOUTPRESENT

validepeutetre(listepremvide,bf) = PEUTETRE

valide(ajprémisse(lisp,p),bf) = si présent(bf,p) alors valide(lisp,bf)
--
sinon si présent(bf,¬p) alors UNFAUX
sinon validepeutetre(lisp,bf)
fsi
fsi

validepeutetre(ajprémisse(lisp,p),bf) = si présent(bf,¬p) alors UNFAUX
--
sinon validepeutetre(lisp,bf)
fsi

Lorsque l'ensemble des règles candidates est déterminé il faut choisir la règle à déclencher. Nous définissons, pour ceci, une opération choix sur le type Enscouple((Listeprem,Conséquent),Indicateur) de profil :

(Enscouple,critère) --> ((Listeprem,Conséquent),Indicateur)

Les critères de choix de la règle à déclencher varient d'un S.E. à l'autre. Citons quelques critères pour la "résolution du conflit" entre les règles candidates.

- Ordre par rapport à une priorité des règles évaluée dynamiquement en fonction de l'intérêt de leur partie conclusion par rapport au but.

- Ordre par rapport aux faits : priorité à la règle qui s'unifie avec le fait jugé le plus important.

- Choix en fonction d'un graphe, lu en donnée, traduisant un ordre partiel sur les règles.

- Choix de la règle la plus récemment utilisée.

- Contrôle par méta-règles qui sont elles-mêmes des règles et qui sont là pour dire ce qu'il faut essayer en priorité dans chaque situation.

-"Sureté" des prémisses dans le cas du raisonnement approximatif.Nous reviendrons sur ce mode de raisonnement.

etc .....

L'opération choix définie en fait une famille de fonctions.

Définition:

Soit ecp ∈ Enscouple; c ∈ Critère; indic ∈ Indicateur

```
choix(ecp,c)= cas c=stratégiel alors choixl(ecp)
                                -----
              cas c=stratégie2 alors choix2(ecp)
               .                -----
               .
               fcas
               ----
```

Lorsque la règle à déclencher est choisie, on met à jour la base des faits grâce aux nouvelles déductions que celle-ci aura permis de faire.

L'opération choix peut délivrer une règle dont les prémisses ne sont pas encore toutes validées. Le système pourra alors tenter de les valider en faisant du chaînage arrière en appelant l'opération essaivalide-liste-prem.

Définition de l'opération "déclenche" :

Profil :

((Listeprem,Conséquent),Indicateur),Bf) --> Bf

Soit lisp ∈ Listeprem : conseq ∈ Conséquent : indic ∈ Indicateur : e ∈ Ensreg ;
---
bf ∈ Bf

```
déclenche (((lisp,conseq),indic)bf) = si indic = SUR alors ajfait(bf,conseq)
                                      --              -----

                                        sinon essaivalide-list-prem(lisp,e,bf)
                                        -----
                                      fsi
                                      ---
```

essaivalide-list-prem fera appel à essaivalide-ens-reg pour les prémisses de lisp non validées.

Si l'opération essaivalide-ens-reg échoue on pourra choisir une nouvelle règle à déclencher.
Le but est atteint lorsque celui-ci sera ajouté à la base des faits.

Remarque :

Si l'on ne désire que déduire de l'information sans vouloir atteindre un but précis, nous ferons appel à l'opération cand de profil:

(Ensregle, bf) --> Enscouple (règle, Indicateur)

Le critère d'arrêt est obtenu lorsque aucune nouvelle déduction ne peut être inférée.

D-V) STRATEGIES:

Pour être véritablement efficace un S.E a besoin de stratégies. Ces stratégies peuvent intervenir tout au long de la tentative de résolution du problème.

Dès le départ, la première question qui se pose est : quel type de raisonnement adopter ? (chaînage arrière, chaînage avant, chaînage mixte).

Un expert en tableaux, par exemple, analysant une oeuvre et n'ayant pas de but précis sinon celui de collecter un maximum d'informations sur le tableau examiné aura une approche de type déductif (chaînage avant). Si au contraire ayant un certain nombre d'informations de départ sur son tableau il désire savoir si l'oeuvre est de Picasso il procèdera de préférence au moyen du raisonnement régressif (chaînage arrière et Picasso est le but à atteindre).

Les deux types de raisonnement peuvent être adoptés au cours d'une même session (chaînage mixte).

Nous pouvons, dans une spécification, tenir compte de cette possibilité de choix de la stratégie en définissant deux opérations : selec et prouver

1) selec décidera de la stratégie à adopter.
2) prouver fera appel au mode chaînage que selec aura choisi.

Si l'utilisateur ne désire atteindre aucun but prédéfini, le chaînage avant s'impose. Si à l'inverse il désire atteindre un but, le système a le choix entre les différents modes de chaînage.
Ce choix peut dépendre de plusieurs facteurs : le but à atteindre, de la taille de la base des faits .....

Le profil de l'opération selec est le suivant :

(stratégie, But, Bf) --> { chainarrière, chainavant, chainmixte } : selec

Le module qui implante selec doit permettre d'analyser le contenu de l'environnement.

Profil de l'opération prouver :

(Ensregle, Stratégie, But, Bf) --> (RESUL, Bf) : prouver

Si selec retourne chainarrière le système fera appel à l'opération essaivalide-ens-reg,si elle retourne chainavant il appellera cand sinon candidate.

Axiome :

Soit e ∈ Ensregle ; s ∈ Stratégie ; b ∈ But ; bf ∈ Bf

```
prouver (e,s,b,bf) = si selec (s,b,bf) = chainarrière alors
                     --                              -----
                     essaivalide-ens-reg (e,b,bf)

                     sinon  si selec(s,b,bf)=chainavant alors
                     -----  --                          -----
                               cand(e,bf)

                            sinon candidate(e,b,bf)
                            -----
                            fsi
                            ---
                      fsi
                      ---
```

La validation d'un but pouvant nécessiter celle de sous buts intermédiaires, les problèmes de stratégies ne se posent pas uniquement au départ d'une session mais tout au long de celle-ci.

Nous sommes également confrontés à ces problèmes de stratégies lors de l'exploration de l'arbre ET/OU. Doit-on effectuer un parcours en profondeur d'abord ou en largeur d'abord ? Dans notre spécification nous avons choisi la première solution. Ce problème de choix de parcours de l'arbre est évidemment moins crucial lorsque la solution du problème nécessite une exploration de tous les chemins possibles menant au but. C'est le cas par exemple de certains systèmes de diagnostic médical où toutes les possibilités sont examinées avant de ne retenir que les plus probables.

Dans notre spécification, en chaînage arrière, nous avons adopté la solution suivante : on parcourt séquentiellement toutes les règles ayant en partie "conséquent" le but à atteindre et on tente de valider l'une d'entre elles.

Une autre stratégie serait d'essayer de valider en premier la règle ayant le moins de premisses mais rien ne garantit que ceci est une solution optimale. En effet, si une des prémisses n'est pas vérifiée dans la base des faits, elle devient un sous but. Sa validation peut entraîner l'empilement de nouveaux sous-buts à vérifier et ainsi de suite.

Une solution adéquate aux problèmes de stratégies semble l'introduction de méta-règles permettant au système de piloter sa recherche.

Les méta-règles sont des règles stratégiques qui évitent l'énumération exhaustive en indiquant la meilleure approche pour un but donné.

Les méta-règles ont la même forme que les règles de production et leurs conclusions indiquent les actions à envisager dans une certaine situation donnée.

Cependant, l'introduction de méta-règles n'est possible que si l'on est capable de dégager les heuristiques de guidage de la recherche à exprimer dans ces règles.

En résumé, pour être véritablement efficace, un S.E devrait être muni de métarègles permettant de dire : étant donné une certaine situation , quel type de raisonnement utiliser, quelles règles utiliser en priorité.

D.VI) LE RAISONNEMENT APPROXIMATIF

Les règles de production qui traduisent l'expérience de l'expert ne reflètent pas des implications logiques mais plutôt les convictions de l'expert. Dans certains systèmes, ce dernier peut traduire la plus ou moins grande certitude (ou confiance) qu'il accorde à chacune des règles en lui associant un coefficient de vraisemblance (c.v).

Si nous prenons l'exemple de MYCIN <SHO 76> chaque partie action d'une règle est affectée d'un coefficient d'atténuation compris entre 0 et 1, et qui exprime donc la confiance qu'a l'expert vis-à-vis de la conclusion annoncée dans le cas où les prémisses sont certaines.

Une règle est entrée dans la base de connaissance sous la forme suivante :

$$R1 : \underline{si}\ (A_1\ \underline{et}\ A_2\ \underline{et}\ \ldots\ \underline{et}\ A_n)\ \underline{alors}\ (B,\ c_b)$$

Au début d'une session,l'utilisateur exprime ses hypothèses de départ avec un certain coefficient de vraisemblance. Ainsi les faits de la partie situation d'une règle sont eux-mêmes valués. Dans MYCIN le coefficient de vraisemblance d'une prémisse est compris entre -1 et 1.

Ainsi, si l'on désire déclencher la règle R1 elle se présente sous la forme suivante :

$$R1 : \underline{si}\ ((A_1,c_1)\ \underline{et}\ \ldots\ \underline{et}\ (A_n,c_n))\ \underline{alors}\ (B,c_b)$$

et le c.v de B déduit par cette règle est le suivant :

$$cv1(B) = c_b \times \min(c_i),\ i \in [1,n]$$

De plus l'existence de plusieurs règles concluant à un même fait conduit à un renforcement du coefficient de ce dernier.

Ainsi si l'on avait une deuxième règle R2 telle que :

$$R2 : \underline{si}\ (H_1\ \underline{et}\ H_2\ \underline{et}\ \ldots\ \underline{et}\ H_p)\ \underline{alors}\ (B,\ c'_b)$$

L'ajout de B à la base des faits se ferait avec le c.v suivant :

$$cv(B) = cv1(B) + cv2(B) - (cv1(B) \times cv2(B))$$

avec $cv2(B) = c'_b \times \min(c'_j),\ j \in [1,p]$

La combinaison des coefficients de vraisemblance est le principal problème posé par leur utilisation dans les S.E.

Le mode de calcul décrit ci-dessus est celui adopté dans MYCIN mais l'on peut en imaginer d'autres.

Il faut cependant remarquer que la non-indépendance des termes conditions des règles rend l'utilisation des concepts probabilistes simples peu adaptée.

La cohérence d'un tel ensemble de coefficients, surtout si plusieurs experts collaborent à la constitution d'une base de connaissances importante est très difficile à garantir.
La détermination d'un mode de combinaison des coefficients qui garantisse une non-dégradation de ceux-ci, quel que soit le nombre d'enchaînements de règles effectués, reste donc à trouver. Le problème du raisonnement approximatif est encore bien mal résolu et reste l'un des grands sujets de recherche actuel.

Spécification

Si l'on désire tenir compte du raisonnement approché dans notre spécification, le profil de l'opération ajrègle du type Ensrègle devient le suivant :

(Ensreg, (Listeprem,(consequent,coefattenu))) --> Ensreg : ajrègle

avec coefattenu $\in [n, n']$ $(n,n' \in \mathbb{N})$.

Chaque système aura ses bornes propres.

L'opération "ajfait" du type Bf aura le profil suivant :

(Bf, (fait, cv)) --> Bf : ajfait avec cv $\in [n,n']$

Il nous faut également définir une opération qui évalue la certitude avec laquelle on ajoute un fait à la base. Deux cas peuvent se présenter :

1) Une seule règle a en partie conclusion le fait à ajouter ; dans ce cas il nous faut définir une opération eval_{ET} de profil ;

(Listeprem, Consequent) --> (Consequent, cv) : eval_{ET}

avec : Listeprem = $((A_1,c_1), (A_2,c_2) \ldots (A_n,c_n))$

Consequent = (B,c_b)

cv $\in \mathbb{R}$

2) Si plusieurs règles concluant à un même fait ont été validées, le calcul du cv est plus complexe.

Soit eval_{OU} l'opération délivrant le cv, son profil est le suivant :

(Ensreg,Conséquent) --> (Consequent, cv) : eval_{OU}

(En fait eval_{OU} est une famille de fonctions)

Remarque :

Ce mode de calcul des coefficients de vraisemblance se rattache bien à la structure de l'arbre ET/OU.

En chaînage arrière, l'opération $eval_{ET}$ serait liée à essaivalide-liste-prem et $eval_{OU}$ à essaivalide-ens-reg. Cependant il faudrait modifier le choix que nous avons pris et qui était de s'arrêter à la première règle validant le but à établir. Le but ne serait ajouté à la base des faits qu'après avoir tenté de valider toutes les règles concluant au but. Si une seule règle a pu être validée, le but est ajouté à la base avec un cv calculé au moyen de $eval_{ET}$.

Lorsque l'on adopte une stratégie de recherche mixte, l'opération "valide", lorsqu'elle délivre TOUTPRESENT, devra également renvoyer un cv indiquant la certitude que l'on peut accorder au(x) fait(s) auquel(s) conclut la règle. Pour ceci on fait appel à l'opération $eval_{ET}$. Si plusieurs règles concluant au(x) même(s) fait(s) ont la valeur TOUTPRESENT, $eval_{OU}$ evaluera le CV.

L'opération choix pourra retourner non pas seulement une seule règle à déclencher mais un ensemble de règles à déclencher.

Dans les S.E utilisant le raisonnement approximatif on n'ajoute un fait à la base que si son cv est supérieur à un certain seuil. Si ce cv est inférieur au seuil, on attribuera à la règle (ou aux règles) en question la valeur UNFAUX afin de les éliminer de l'ensemble des règles candidates.

CONCLUSION

Les T.A ont été pour nous un cadre formel qui nous a permis de mieux comprendre la notion de système expert ; de dégager les objets et les opérations à effectuer sur ces objets ; de bien situer à quel niveau devraient s'effectuer les choix.
Le cycle de base de tous les moteurs d'inférences est le suivant :

a) déterminer l'ensemble des règles candidates

b) choisir la règle à déclencher

c) exécuter la partie conclusion de la règle et mettre à jour la base des faits.

On se rend compte qu'il existe des problèmes de choix, liés surtout à la stratégie, aux niveaux a) et b).

CHAPITRE E

IMPLEMENTATION DE TYPES ABSTRAITS

CHAPITRE E

IMPLEMENTATION DE TYPES ABSTRAITS

Il existe des langages bien adaptés à l'implémentation de spécifications typées et fonctionnelles.Ces langages sont dits "fortement typés" et nous pouvons en citer ALPHARD,SIMULA,ATM,CLU,etc...
Nous allons nous interesser plus particulièrement au langage CLU developpé au MIT par l'équipe de Barbara Liskov.

E-I) CLU <GAU 82> <LIS 81>

CLU fournit un certain nombre de types de base prédéfinis et un mécanisme puissant de définition de nouveaux types:les clusters qui définissent chacun un type abstrait, c'est-à-dire un ensemble d'objets et un ensemble d'opérations sur ces objets.

E-1) Les types de base:

On trouve les types habituels,munis des opérations usuelles:int,real,bool,char, string .

On a deux autres types de base plus inhabituels qui sont null et any.

- null est un type avec un seul objet nil,et sans opération.Il est utilisé pour initialiser certaines constructions de types récursifs.

- any est l'union de tous les types possibles.Il lui est associé une procédure (ou plus précisément un générateur de procédures) force(t) qui permet des conversions de any dans t.

E-2) Les Clusters:

CLU a été conçu pour encourager l'usage des types abstraits en programmation. Un Cluster permet de définir un type abstrait et son corps n'est pas accessible de l'extérieur.

En effet, à l'extérieur du cluster,on ne peut utiliser sur le type t que les opérations lui correspondant.

Un Cluster est composé de deux parties:

-un en-tête qui décrit les identificateurs définis par le Cluster et qui pourront etre utilisés à l'extérieur du Cluster.

-un corps formé de déclarations.

Les déclarations d'identificateurs n'apparaissant pas dans l'en-tête sont cachées pour l'extérieur du Cluster.

Exemple:

```
pile= cluster is  create,is-empty,push,pop,top
      ------- --
```

E-3) Clusters paramétrés:

Tout au long de notre travail nous avons utilisé les types paramétrés.En CLU les Clusters peuvent être pramétrées et on peut indiquer des conditions (syntaxiques) sur les paramètres.Par exemple,on a besoin pour spécifier un type Ensemble

de valeurs d' une opération d'égalité sur les valeurs.Ceci s'écrit en CLU dans l'en-tete du Cluster:

```
set=cluster(t:type) is empty,is-empty,insert,delete,is-in
    -------
    where t has equal:proctype(t,t) returns (bool)
    -----   ---       --------      -------  ----
```

Il s'agit d'une condition syntaxique sur t en ce sens que les seules vérifications portent sur le nom et le type de l'opération demandée,pas sur ses propriétés (equal pourrait fort bien ne pas être une relation d'équivalence).

E-4) Traitement des exceptions:

Une procédure de CLU peut se terminer soit en rendant un résultat,soit en signalant une exception.Si on reprend l'exemple de la pile on aura l'en-tête suivant pour la procédure "top".

```
top=proc(p:cvt) returns(t) signals (sommet-de-pile-vide)
    ----        -------    -------
```

Le corps de cette procédure peut être:

```
if pile$ is-empty(p) then
--                   ----
   signal(sommet-de-pile-vide)
   ------

else return (rep$top(p))
---- ------  ---
```

Conclusion:

CLU permet l'implémentation de types abstraits,il possède un mécanisme puissant de traitement des exceptions,permet une programmation très modulaire et la possibilité de compilation séparée assure la vérification des types au niveau des références entre modules.

La réalisation d'une maquette du système EXPRIM devrait être relativement aisée en CLU.Le passage d'une réalisation en CLU à une réalisation dans un langage plus répandu tel que PASCAL mériterait d'être étudié .

CONCLUSION

CONCLUSION

L'approche types abstraits algébriques nous a permis de concevoir de façon très modulaire et formelle une grande partie du système EXPRIM. Par une démarche, descendante nous sommes parvenus à un inventaire des principaux types utiles à la réalisation de notre système.

Cette approche nous a également permis de mieux comprendre les concepts utilisés dans les systèmes experts. Bien qu'il existe une littérature abondante sur ce sujet, le fonctionnement et la réalisation d'un système expert ne sont pas faciles à décrire.
Il existe en effet plusieurs "types" de système expert d'où des difficultés de choix, de comparaison et de réalisation. Notre approche formelle des systèmes experts nous a permis, en nous plaç ant à un niveau "méta-système", de mieux cerner à quels niveaux se situent les choix et donc les différences qui existent entre les différents systèmes.

Les différentes spécifications correspondant aux différents types de stratégie permettent au concepteur d'un S.E. de mieux percevoir les conséquences de ces choix et, en cas de remise en cause de certains de ceux-ci, de retrouver plus facilement les modifications à apporter à ses programmes.
Il reste cependant beaucoup de travail à faire pour améliorer et préciser notre spécification. Nous n'avons pas pris en compte les systèmes utilisant des variables dans les règles. Certains aspects tels que la fonction de choix des règles candidates et l'évaluation et la prise en compte du coefficient de vraisemblance et de diverses autres mesures d'affaiblissement n'ont pas été assez approfondis. Cependant le travail le plus délicat et fondamental restant à faire est la prise en compte de la stratégie. Cette stratégie nécessitant une certaine dynamique pose un problème sérieux pour être exprimée en termes de types abstraits car ceux-ci se prètent mieux à des définitions statiques.

Bien que le passage d'une spécification à une réalisation ne soit pas évident il existe des langages supportant le concept de types abstraits et facilitant leur implantation. En particulier, il existe des outils, les systèmes de réécriture et leurs générateurs (tel que REVE), permettant de faire des preuves sur les spécifications et de tester leur correction. Nous avons d'ailleurs pu coopérer avec l'équipe EURECA (CRIN) qui nous a permis de tester les types $\text{Descripimage}_{(a)}$ et $\text{Descripimage}_{(b)}$ à l'aide de REVE.

Il serait cependant souhaitable de pouvoir disposer dans REVE d'un type produit cartésien, et plus généralement de types paramétrés prédéfinis, et de généraliser le système pour qu'il accepte des conditions de hiérarchies plus souples dans les systèmes conditionnels.

Il reste également à reprendre les spécifications et à les réécrire en se rapprochant le plus possible de la syntaxe de CLU.

BIBLIOGRAPHIE

ANNEXE

BIBLIOGRAPHIE

<AIT 82> AIT HADDOU A., BOUKAKIOU M., CREHANGE M., DAVID J.M., FOUCAUT O., MAROLDT J. (CRIN- NANCY)
Système "intelligent" de base de données intégrant textes et images.
Séminaire Bases de Données, Toulouse (nov. 82).

<AIT 83> AIT HADDOU A., BOUKAKIOU M., CREHANGE M., DAVID J.M., FOUCAUT O., MAROLDT J. (CRIN-NANCY)
"EXPRIM" : Un système d'aide à l'interrogation progressive d'une base d'images-Premières propositions (rapport n. 83-R-013).

<AIT 83> AIT HADDOU A., BOUKAKIOU M., CREHANGE M., DAVID J.M., FOUCAUT O., MAROLDT J. (CRIN-NANCY)
"EXPRIM" : On expert system to aid in progressive retrieval from pictorial and descriptive database.
Special workshop on new applications of Databases in conjunction with ICOD-2. Septembre 2-3 1983, Cambridge (England).

<BAN 78> BANATRE M., COUVERT A., HERMAN D., RAYNAL M.
Types abstraits et pluralité de leurs représentations à l'exécution.
(I.R.I.S.A - Rennes, Publication interne n. 88 - Fev. 78).

<BER 82> BERT D.
Refinements of Generic Specifications with Algebraic Tools.
(I.M.A.G - Grenoble, RR n. 335 - Nov. 82).

<BER 79> BERT D.
Spécification algébrique des Types Abstraits et certification des programme.
(Groplan-AFCET , Cargèse 1979).

<BID 81> BIDOIT M.
Une méthode de présentation des types abstraits algébriques.
Thèse de 3e cycle, Université Paris-Sud, 1981.

<BON 81> BONNET A.
Applications de l'intelligence artificielle : les systèmes experts.
RAIRO Informatique, 1981, vol. 15, n, 4, pp. 325-341.

<BOU 83> BOUKAKIOU M., CREHANGE M. (CRIN-NANCY)
Spécification of an Expert System with Abstract Data Types.
Rapport n. 83-R-88.

<BOU 83> BOUKAKIOU M., CREHANGE M.
Un exemple de spécification à l'aide de types abstraits algébriques : le type "IMAGE" dans le système EXPRIM. Rapport CRIN 83-R-30.

<BOU 84> BOURELLY L., CHOURAQUI E.
Le système documentaire Satin 1.
CNRS, Paris, 1974.

<BRO 79> BROY M., WIRSING M., FINANCE J.P., QUERE A., REMY J.L.
Methodical solution of the problem of ascending subsequences of maximum length within a given sequence Information Processing Letters (June 79).

<BRO 80> BROY M., WIRSING M.
Programming languages as Abstract Data Types.
5ème CAAP, Lille 80.

<BRO 82> BROY M., WIRSING M. (Institut für Informatik. Munchen)
Partial Abstract Types
Acta Informatica vol. 18 nov 82

<BUC 78> BUCHANAN B., FEIGENBAW E.
Dendral and Meta-Dendral : Their applications dimension Artificial Intelligence 11 (1978) (p. 5-24).

<BUR 77) BURSTALL R.M., GOGUEN J.A.
Putting theories together to make specifications.
Proc. of 5th IJCAI, Cambridge Mass, 1977.

<CHA 82> CHABRIER J.J
Spècification et construction de systémes orientès Bases de Donnèès: Techniques et langages basès sur le concept de Type Abstrait.
Thése d'ètat, Nancy I. Octobre 82.

<CHA 82> CHABRIER J.
Présentation et utilisation du langage PROLOG.
CRIN-NANCY Rapport n. 82-R-078 (1982).

<CHA 81> CHANDON J.L., PINSON S.
Analyse typologique.
Masson 1981.

<CHA 81> CHANG N.S., FU K.S.
Picture query languages for pictorial database systems.
Computer (nov. 1981), p. 23-33.

<CHA 79> CHANG C.L., SLAGLE J.R.
Using Rewriting Rules for Connection graphs to prove theorems.
Artificial Intelligence 12 (1979), 159-180.

<COR 83> CORDIER M.O., ROUSSET M.C.
TANGO : Moteur d'inférences pour un système-expert avec variables.
Rapport de recherche n. 123, L.R.I., Orsay, 1983.

<COR 84> CORDIER M.O.
Les systèmes experts.
La Recherche n. 151, Janvier 1984.

<DAT 81> DATE. An introduction to database systems, third edition.
Addison-Wesley.

<DAV 77> DAVIS R., DUCHANAN B., SHORLIFFE E.
Production rules as a representation for a knowledge-based consultation program.
Artificial Intelligence, 15-45, 1977.

<DEL 79> DELIYANNI A., KOWALSKI R.
Logic and semantic networks.
Communications of the ACM, March 1979, volume 22.

<DEL 82> DELOBEL C., ADIBA M.
Bases de données et systèmes relationnels.
Dunod Ed. 1982.

<DER 82> DERANSART P.
Dérivation de programmes PROLOG à partir de spécifications algébriques.
INRIA - Version provisoire - Mars 1982.

<DER 79> DERNIAME J.C., FINANCE J.P.
Types abstraits de données, spécification, utilisation et réalisation.
CRIN-NANCY, Rapport 79-E-57, 1979.

<FIN 79> FINANCE J.P.
Etude de la construction des programmes. Méthodes et langages de spécification et résolution des problèmes.
Thèse d'Etat, Université de Nancy 1, 1979.

<GAR 82> GARDARIN G., BERNADAS P., TEMMERMAN N., VALDURIEZ P., VIEMONT Y.
SABRE : A relational database system for a multiprocessor machine.
2nd Int. Workshop on database machine, San Diego, Sept. 1982.

<GAU 78> GAUDEL M.C.
Spécifications incomplètes mais suffisantes de la représentation des types abstraits.
Rapport de recherche INRIA n. 320, Aout 1978.

<GAU 79> GAUDEL M.C.
Algebraic specification on abstract data types.
Rapport de recherche INRIA n. 360, Aout 1979.

<GAU 80) GAUDEL M.C.
Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation.
Thèse d'Etat, INPL Nancy, 1980.

<GAU 82> GAUDEL M.C.
Fiche d'étude du langage CLU.
Centre de Recherches de la C.G.E., 1982.

<GOG 81> GOGUEN J.A., PARSAYE - GHOMI K.
Algebraic denotational semantic using parameterized abstract modules.
Lectures Notes in Computer Science, 1981, 107, pp. 292-309.

<GOG 78> GOGUEN J.A., TATCHER J.W., WAGNER E.G.
An initial algebra approach to the specification, correctness, and implementation of abstract data types.
Acta Informatica, 1978.

<GUT 77> GUTTAG J.
Abstract Data Types and the development of Data Structures.
Communications of ACM, volume 20, number 6, 1977.

<GUT 78> GUTTAG J.V., HOROWITZ E., MUSSER D.R.
Abstract data types and software validation.
Communications ACM, vol. 21, n. 12, Décembre 1978, pp. 1048-1063.

<GUT 78> GUTTAG J.V., HORNING J.J.
The algebraic specification of abstract data types.
Acta Informatica, 10, 1978, pp. 27-52.

<GUT 80> GUTTAG J.V., HORNING J.J.
Formal specification as a design tool.
7ème POPL 80, Las Vegas, pp. 251-261.

<HUD 82> HUDRISIER H.
La banque de données d'images de presse.
Rapport Sygma, 1982.

<HUE 73> HUET G., HULLOT J.M.
Proof by induction in equational theories with constructors.
INRIA, R.R. n. 28.

<HOA 72> HOARE C.A.R.
Proof of Correctness of Data Representations.
Acta Informatica 1, 271-281 (1972).

<JAN 83> JANICE S. AIKINS
Prototypical Knowledge for expert systems.
Artificial Intelligence, vol. 20, 1983, pp. 163-210.

<JOU 82> JOUANNAUD J-P.,LESCANNE P.,REINIG F.
Recursive Decomposition Ordering.
Conf. on Formal Description of Programming Concepts,Garmisch,1982

<JUL 83> JULLIAND J. (CRIN et I.U.T de Belfort)
Spécification algébrique de la communication entre processus parallèles.
T.S.I volume 2 numéro 4 1983

<KIR 82> KIRCHNER C., KIRCHNER H.
Contribution à la résolution d'équations dans les algèbres libres et les variétés équationnelles d'algèbre.
Université Nancy, Thèse 3e cycle, 1982.

<LAU 82> LAURIERE J.L.
Représentation et utilisation des connaissances.
Les systèmes experts, vol. 1, n. 1, 1982, Techniques et Sciences Informatiques.

<LAU 82> LAURIERE J.L.
Représentation et utilisation des connaissances. Représentation des connaissances.
T.S.I., 1982, vol. 1, n. 2.

<LES 83> LESCANNE P. (CRIN-Nancy et MIT Massachussets)
Computer Experiments with the REVE term rewriting system.
Generator Proceedings of 10th Symposium of Programming Languages.
Austin, Texas, 1983.

<LES 83> LESCANNE P.
Behavioral categoricity of Abstract Data Type spécifications.
The Computer Journal, 1983.

<LES 79> LESCANNE P.
Etude algébrique et relationnelle des types abstraits et leur représentation.
Thèse d'Etat, INPL Nancy, 1979.

<LES 84> LESCANNE P. (CRIN-Nancy)
Uniform terminaison of term rewriting systems.Recursive decomposition ordering with status.
9th collo. sur les arbres en algèbre et en programmation.
Bordeaux 84.

<LES 84> LESCANNE P.
Terms rewriting systems and algebra.
7th conf. on Automated deduction (Napa Valley 84)

<LIS 74> LISKOV B.H., ZILLES S.N.
Programming with abstract data types.
Proceeding of ACM Sigplan symposium on very high level languages.
Sigplan Notices, 1974, 9, pp. 50-59.

<LIS 81> LISKOV B.,ATKINSON R.,BLOOM T., and all.
CLU reference manual
Lecture Notes on Computer Science num. 114 .Springer Verlag.

<LIV 78> LIVERCY
Théorie des programmes.
Dunod Informatique, 1978.

<MEU 83> MEUNIER B.
Un exemple de système expert utilisant des données imprécises : MYCOMATIC, rapport de D.E.A., CRIN, 1983.

<MUS 80> MUSSER R. DAVID
Abstract data type specification in the AFFIRM system.
I.E.E.E., January 1980, vol. SE-6, n. 1.

<PAI 78> PAIR C.
La programmation : de l'énoncé au programme.
CRIN-Nancy, Rapport n. 78-P-061, 1978.

<PAI 80> PAIR C.
Sur les modèles des types abstraits algébriques.
CRIN-Nancy, Rapport 80-P-052, 1980.

<PIN 81> PINSON S.
Représentation des connaissances dans les systèmes experts.
RAIRO Informatique, 1981, vol, 15, n. 4, pp, 343-367.

<PIT 82> PITRAT J.
Un langage pour décrire les connaissances de façon déclarative.
Colloque d'Intelligence Artificielle, Le Mans, 20-24 Sep. 1982.

<PRO 82> PROCH K.
ORSEC : Un outil de recherche de spécifications équivalente par comparaison d'exemples.
Thèse 3e cycle, Université Nancy 1, 1982.

<REM 82> REMY J.L.
Etude des systèmes de réécriture conditionnels et applications aux types abstraits algébriques.
Thèse d'Etat, INPL-CRIN Nancy, 1982.

<REM 80> REMY J.L.
Construction, évaluation et amélioration systématiques de structures de données.
RAIRO, 1980, p. 83-118.

<REM 83> REMY J.L.
Ecologiste a system to make complete and consistent spécifications casier.
CRIN-Nancy, Rapport n. 83-R-099, 1983.

<SYS 84> SYSTEX.
Système SPIRIT.
Document interne CISI 84.

<SAL 71> SALTON G.
The smart retrevieval system : experiments in automatic document processing.
Prentice Hall Inc. Englewood Cliffs, New-Jersey, 1971.

<TOM 80> TOMPA F.
A practical example of the specification of Abstract Data Types.
Acta Informatica 13 p. 205-224, 1980.

<WEI 80> WEINER J.L.
BLAH. a system which explains its reasoning Artificial Intelligence 15, 1980, p. 19-48.

<WUL 76> WULF W., LONDON R., SHAW M.
An introduction to the construction and verification of Alphard Programs
IEEE, vol. SE-2, Décembre 1976.

<WIR 83> WIRSING M., PEPPER P., PARTSCH H., DOSCH W., BROY M.
On hierarchies of Abstract Data Types.
TUM-18 303, Tech. Univ. Munchen, 1983.

Equations:

```
 1.  pres(imvide, x)  == false
 2.  (x = y)  :: pres(ajdes(xx, x), y)  == true
 3.  not((x = y))  :: pres(ajdes(xx, x), y)  == pres(xx, y)
 4.  supdes(imvide, x)  == imvide
 5.  (x = y)  :: supdes(ajdes(yy, x), y)  == supdes(yy, y)
 6.  not((x = y))  :: supdes(ajdes(yy, x), y)  == ajdes(supdes(yy, y), x)
 7.  pres(xx, x)  :: modifides(xx, x, y)  == ajdes(supdes(xx, x), y)
 8.  not(pres(xx, x))  :: modifides(xx, x, y)  == xx
 9.  ((x = z) | (y = z))  :: pres(ajrel(xx, xr, x, y), z)  == true
10.  (not((x = z)) & not((y = z)))  :: pres(ajrel(xx, xr, x, y), z)
          == pres(xx, z)
11.  ((x = z) | (y = z))  :: supdes(ajrel(xx, xr, x, y), z)  == supdes(xx, z)
12.  (not((x = z)) & not((y = z)))  :: supdes(ajrel(xx, xr, x, y), z) ==
                    ajrel(supdes(xx, z), xr, x, y)
13.  suprel(imvide, xr, x, y)  == imvide
14.  ((xr1 = xr2) & ((x = z) & (y = w))) ::
          suprel(ajrel(xx, xr1, x, y), xr2, z, w)
     == suprel(xx, xr2, z, w)
15.  not(((xr1 = xr2) & ((x = z) & (y = w)))) ::
          suprel(ajrel(xx, xr1, x, y), xr2, z, w) ==
                    ajrel(suprel(xx, xr2, z, w), xr1, x, y)
16.  presrel(imvide, xr, x, y)  == false
17.  ((xr1 = xr2) & ((x = z) & (y = w))) ::
          presrel(ajrel(xx, xr1, x, y), xr2, z, w)  == true
18.  not(((xr1 = xr2) & ((x = z) & (y = w)))) ::
          presrel(ajrel(xx, xr1, x, y), xr2, z, w)
     == presrel(xx, xr2, z, w)
19.  presrel(ajdes(xx, x), xr, y, z)  == presrel(xx, xr, y, z)
```

Rewrite Rules:

2.666666

Apres avoir atteint le bord de l'abime, nous avons fait un pas en avant :

-> cc

Please type in your constructors. They will be considered as standard
abstract data type constructors, rather than constructors in the sense of the
Huet & Hullot inductionless induction method.
Constructors: iimmvviided e ajajdedse s ajarjelrel

0.083333

```
Vous allez participer a des evenements prodigieux.

-> kk

Starting Knuth-Bendix...

Equation has been ordered:
    pres(imvide, x)  == false

Rule added to rewriting system:
    pres(imvide, x)  -> false

Equation has been ordered:
    (x = y)  ;; pres(ajdes(xx, x), y)  == true

Rule added to rewriting system:
    (x = y)  ;; pres(ajdes(xx, x), y)  -> true

Equation has been ordered:
    (true # (x = y))  ;; pres(ajdes(xx, x), y)  == pres(xx, y)

Rule added to rewriting system:
    (true # (x = y))  ;; pres(ajdes(xx, x), y)  -> pres(xx, y)

Equation has been ordered:
    supdes(imvide, x)  == imvide

Rule added to rewriting system:
    supdes(imvide, x)  -> imvide

Equation has been ordered:
    (x = y)  ;; supdes(ajdes(yy, x), y)  == supdes(yy, y)

Rule added to rewriting system:
    (x = y)  ;; supdes(ajdes(yy, x), y)  -> supdes(yy, y)

Equation has been ordered:
    (true # (x = y))  ;; supdes(ajdes(yy, x), y)  == ajdes(supdes(yy, y), x)

Rule added to rewriting system:
    (true # (x = y))  ;; supdes(ajdes(yy, x), y)  -> ajdes(supdes(yy, y), x)

Equation has been ordered:
    (true # pres(xx, x))  ;; modifides(xx, x, y)  == xx

Rule added to rewriting system:
    (true # pres(xx, x))  ;; modifides(xx, x, y)  -> xx

Equation has been ordered:
    ((x = z) # ((y = z) # ((x = z) & (y = z))))
        ;; pres(ajrel(xx, xr, x, y), z)  == true

Rule added to rewriting system:
    ((x = z) # ((y = z) # ((x = z) & (y = z))))
        ;; pres(ajrel(xx, xr, x, y), z)  -> true

Equation has been ordered:
    (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ;;
        pres(ajrel(xx, xr, x, y), z)  == pres(xx, z)

Rule added to rewriting system:
    (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ;;
        pres(ajrel(xx, xr, x, y), z)  -> pres(xx, z)

Equation has been ordered:
```

```
    ((x = z) # ((y = z) # ((x = z) & (y = z))))
        :: supdes(ajrel(xx, xr, x, y), z)  == supdes(xx, z)

Rule added to rewriting system:
    ((x = z) # ((y = z) # ((x = z) & (y = z))))
        :: supdes(ajrel(xx, xr, x, y), z)  -> supdes(xx, z)

Equation has been ordered:
    (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ::
            supdes(ajrel(xx, xr, x, y), z)
     == ajrel(supdes(xx, z), xr, x, y)

Rule added to rewriting system:
    (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ::
            supdes(ajrel(xx, xr, x, y), z)
     -> ajrel(supdes(xx, z), xr, x, y)

Equation has been ordered:
    suprel(imvide, xr, x, y)  == imvide

Rule added to rewriting system:
    suprel(imvide, xr, x, y)  -> imvide

Equation has been ordered:
    ((y = w) & ((x = z) & (xr1 = xr2)))
        :: suprel(ajrel(xx, xr1, x, y), xr2, z, w)  == suprel(xx, xr2, z, w)

Rule added to rewriting system:
    ((y = w) & ((x = z) & (xr1 = xr2)))
        :: suprel(ajrel(xx, xr1, x, y), xr2, z, w)  -> suprel(xx, xr2, z, w)

Equation has been ordered:
    (true # ((y = w) & ((x = z) & (xr1 = xr2)))) ::
            suprel(ajrel(xx, xr1, x, y), xr2, z, w) ==
                    ajrel(suprel(xx, xr2, z, w), xr1, x, y)

Rule added to rewriting system:
    (true # ((y = w) & ((x = z) & (xr1 = xr2)))) ::
            suprel(ajrel(xx, xr1, x, y), xr2, z, w) ->
                    ajrel(suprel(xx, xr2, z, w), xr1, x, y)

Equation has been ordered:
    presrel(imvide, xr, x, y)  == false

Rule added to rewriting system:
    presrel(imvide, xr, x, y)  -> false

Equation has been ordered:
    ((y = w) & ((x = z) & (xr1 = xr2)))
        :: presrel(ajrel(xx, xr1, x, y), xr2, z, w)  == true

Rule added to rewriting system:
    ((y = w) & ((x = z) & (xr1 = xr2)))
        :: presrel(ajrel(xx, xr1, x, y), xr2, z, w)  -> true

Equation has been ordered:
    (true # ((y = w) & ((x = z) & (xr1 = xr2)))) ::
            presrel(ajrel(xx, xr1, x, y), xr2, z, w)
     == presrel(xx, xr2, z, w)

Rule added to rewriting system:
    (true # ((y = w) & ((x = z) & (xr1 = xr2)))) ::
            presrel(ajrel(xx, xr1, x, y), xr2, z, w)
     -> presrel(xx, xr2, z, w)

Equation has been ordered:
```

```
Rule added to rewriting system:
    presrel(ajdes(xx, x), xr, y, z)  -> presrel(xx, xr, y, z)

The following rule:
    (true # (x = y))  :: pres(ajdes(xx, x), y)  -> pres(xx, y)
Resulted in the following critical pairs:
    ((true # (x = y)) & (x = y))  :: pres(xx, y)  == true
    ((x = y) & (true # (x = y)))  :: true  == pres(xx, y)

The following rule:
    (true # (x = y))  :: supdes(ajdes(yy, x), y)  -> ajdes(supdes(yy, y), x)
Resulted in the following critical pairs:
    ((true # (x = y)) & (x = y))  :: ajdes(supdes(yy, y), x)  == supdes(yy, y)
    ((x = y) & (true # (x = y)))  :: supdes(yy, y)  == ajdes(supdes(yy, y), x)

The following rule:
    (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ::
            pres(ajrel(xx, xr, x, y), z)  -> pres(xx, z)
Resulted in the following critical pairs:
    ((true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) &
        ((x = z) # ((y = z) # ((x = z) & (y = z)))))  :: pres(xx, z)  == true
    (((x = z) # ((y = z) # ((x = z) & (y = z)))) &
        (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))))  :: true ==
                    pres(xx, z)

The following rule:
    (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ::
            supdes(ajrel(xx, xr, x, y), z)
     -> ajrel(supdes(xx, z), xr, x, y)
Resulted in the following critical pairs:
    ((true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) &
        ((x = z) # ((y = z) # ((x = z) & (y = z))))) ::
            ajrel(supdes(xx, z), xr, x, y)  == supdes(xx, z)
    (((x = z) # ((y = z) # ((x = z) & (y = z)))) &
        (true # ((x = z) # ((y = z) # ((x = z) & (y = z))))))  :: supdes(xx, z) ==
                    ajrel(supdes(xx, z), xr, x, y)

The following rule:
    (true # ((y = w) & ((x = z) & (xr1 = xr2)))) ::
            presrel(ajrel(xx, xr1, x, y), xr2, z, w)
     -> presrel(xx, xr2, z, w)
Resulted in the following critical pairs:
    ((true # ((y = w) & ((x = z) & (xr1 = xr2)))) &
        ((y = w) & ((x = z) & (xr1 = xr2))))  :: presrel(xx, xr2, z, w)
         == true
    (((y = w) & ((x = z) & (xr1 = xr2))) &
        (true # ((y = w) & ((x = z) & (xr1 = xr2)))))  :: true ==
                    presrel(xx, xr2, z, w)

The following rule:
    (true # ((y = w) & ((x = z) & (xr1 = xr2)))) ::
            suprel(ajrel(xx, xr1, x, y), xr2, z, w) ->
                    ajrel(suprel(xx, xr2, z, w), xr1, x, y)
Resulted in the following critical pairs:
    ((true # ((y = w) & ((x = z) & (xr1 = xr2)))) &
        ((y = w) & ((x = z) & (xr1 = xr2)))) ::
            ajrel(suprel(xx, xr2, z, w), xr1, x, y)
     == suprel(xx, xr2, z, w)
    (((y = w) & ((x = z) & (xr1 = xr2))) &
        (true # ((y = w) & ((x = z) & (xr1 = xr2)))))  :: suprel(xx, xr2, z, w) ==
                    ajrel(suprel(xx, xr2, z, w), xr1, x, y)

The following equation currently cannot be ordered:
```

(xx, x) ;; modifides (xx, x, y) == ajdes (supdes (xx, x), y)

following precedence suggestions may allow the equation to be ordered:

1. modifides > supdes

you accept all of the above suggestions, the equation can be ordered
it stands. Do you accept all of them (y/n)? yy

ation has been ordered:
pres(xx, x) ;; modifides(xx, x, y) == ajdes(supdes(xx, x), y)

e added to rewriting system:
pres(xx, x) ;; modifides(xx, x, y) -> ajdes(supdes(xx, x), y)

following rule:
pres(xx, x) ;; modifides(xx, x, y) -> ajdes(supdes(xx, x), y)
lted in the following critical pairs:
(pres(xx, x) & (true # pres(xx, x))) ;; ajdes(supdes(xx, x), y) == xx
((true # pres(xx, x)) & pres(xx, x)) ;; xx == ajdes(supdes(xx, x), y)

tions:

ite Rules:

```
pres(imvide, x) -> false
supdes(imvide, x) -> imvide
suprel(imvide, xr, x, y) -> imvide
presrel(imvide, xr, x, y) -> false
presrel(ajdes(xx, x), xr, y, z) -> presrel(xx, xr, y, z)
(true # pres(xx, x)) ;; modifides(xx, x, y) -> xx
(x = y) ;; pres(ajdes(xx, x), y) -> true
(true # (x = y)) ;; pres(ajdes(xx, x), y) -> pres(xx, y)
(x = y) ;; supdes(ajdes(yy, x), y) -> supdes(yy, y)
((x = z) # ((y = z) # ((x = z) & (y = z))))
     ;; pres(ajrel(xx, xr, x, y), z) -> true
(true # (x = y)) ;; supdes(ajdes(yy, x), y) -> ajdes(supdes(yy, y), x)
(true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ;;
     pres(ajrel(xx, xr, x, y), z) -> pres(xx, z)
((x = z) # ((y = z) # ((x = z) & (y = z))))
     ;; supdes(ajrel(xx, xr, x, y), z) -> supdes(xx, z)
((y = w) & ((x = z) & (xr1 = xr2))) ;;
     presrel(ajrel(xx, xr1, x, y), xr2, z, w) -> true
(true # ((x = z) # ((y = z) # ((x = z) & (y = z))))) ;;
     supdes(ajrel(xx, xr, x, y), z)
-> ajrel(supdes(xx, z), xr, x, y)
((y = w) & ((x = z) & (xr1 = xr2))) ;;
     suprel(ajrel(xx, xr1, x, y), xr2, z, w)
-> suprel(xx, xr2, z, w)
(true # ((y = w) & ((x = z) & (xr1 = xr2)))) ;;
     presrel(ajrel(xx, xr1, x, y), xr2, z, w)
-> presrel(xx, xr2, z, w)
(true # ((y = w) & ((x = z) & (xr1 = xr2)))) ;;
     suprel(ajrel(xx, xr1, x, y), xr2, z, w) ->
          ajrel(suprel(xx, xr2, z, w), xr1, x, y)
pres(xx, x) ;; modifides(xx, x, y) -> ajdes(supdes(xx, x), y)
```

system is complete!

533333

d'Onyx a 15:30 aujourd'hui pour le crash hebdomadaire.

```
Please give me the term for which you would like the normal form computed:
pres((aajrjerell((imimaass,,  rrelel, , x,x ,y y)),,  xx))
The set of normal forms of your term is:
true

Would you like to see the sequences of reduction of your term ? (y/n)yy

The sequence of term reductions leading to one of the normal forms of your term is:
pres(ajrel(imas, rel, x, y), x)
((x = x) # ((y = x) # ((x = x) & (y = x))))  :: true
(true # ((y = x) # ((x = x) & (y = x))))  :: true
(true # ((y = x) # (true & (y = x))))  :: true
(true # ((y = x) # (y = x)))  :: true
(true # false)  :: true
(false # true)  :: true
true  :: true
true


.716666

    avez de la moelle


Please give me the term for which you would like the normal form computed:
pres((ajarjerell((aajrjelrel((imimasas,,  rr11,,  x,x,  yy)),,  rr22, , aa, , bb)),,  xx))
The set of normal forms of your term is:
true

Would you like to see the sequences of reduction of your term ? (y/n)

The sequence of term reductions leading to one of the normal forms of your term is:
1) pres(ajrel(ajrel(imas, r1, x, y), r2, a, b), x)
2) (a = x) # ((b = x) # ((a = x) & (b = x))))  :: true
3) pres(ajrel(ajrel(imas, r1, x, y), r2, a, b), x)
4) true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) ::
        pres(ajrel(imas, r1, x, y), x)
5) (true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) &
    ((x = x) # ((y = x) # ((x = x) & (y = x))))  :: true
6) (true & ((x = x) # ((y = x) # ((x = x) & (y = x))))) #
    (((a = x) # ((b = x) # ((a = x) & (b = x)))) &
        ((x = x) # ((y = x) # ((x = x) & (y = x))))))  :: true
7) ((x = x) # ((y = x) # ((x = x) & (y = x)))) #
    (((a = x) # ((b = x) # ((a = x) & (b = x)))) &
        ((x = x) # ((y = x) # ((x = x) & (y = x))))))  :: true
8) (true # ((y = x) # ((x = x) & (y = x)))) #
    (((a = x) # ((b = x) # ((a = x) & (b = x)))) &
        ((x = x) # ((y = x) # ((x = x) & (y = x))))))  :: true
9) (true # ((y = x) # (true & (y = x)))) #
    (((a = x) # ((b = x) # ((a = x) & (b = x)))) &
        ((x = x) # ((y = x) # ((x = x) & (y = x))))))  :: true
10) ((true # ((y = x) # (y = x))) #
     (((a = x) # ((b = x) # ((a = x) & (b = x)))) &
         ((x = x) # ((y = x) # ((x = x) & (y = x))))))  :: true
11) ((true # ((y = x) # (y = x))) #
     (((a = x) & ((x = x) # ((y = x) # ((x = x) & (y = x))))) #
         (((b = x) # ((a = x) & (b = x))) &
             ((x = x) # ((y = x) # ((x = x) & (y = x)))))))  :: true
12) ((true # ((y = x) # (y = x))) #
     ((((a = x) & (x = x)) # ((a = x) & ((y = x) # ((x = x) & (y = x))))) #
         (((b = x) # ((a = x) & (b = x))) &
             ((x = x) # ((y = x) # ((x = x) & (y = x)))))))  :: true
13) ((true # ((y = x) # (y = x))) #
```

```
                (((b = x) # ((a = x) & (b = x))) &
                   ((x = x) # ((y = x) # ((x = x) & (y = x))))))  :: true
4). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) #
             (((a = x) & (y = x)) # ((a = x) & ((x = x) & (y = x))))) #
            (((b = x) # ((a = x) & (b = x))) &
               ((x = x) # ((y = x) # ((x = x) & (y = x)))))))  :: true
5). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) #
             (((a = x) & (y = x)) # ((a = x) & (true & (y = x))))) #
            (((b = x) # ((a = x) & (b = x))) &
               ((x = x) # ((y = x) # ((x = x) & (y = x)))))))  :: true
6). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            (((b = x) # ((a = x) & (b = x))) &
               ((x = x) # ((y = x) # ((x = x) & (y = x)))))))  :: true
7). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            (((b = x) & ((x = x) # ((y = x) # ((x = x) & (y = x))))) #
               (((a = x) & (b = x)) &
                  ((x = x) # ((y = x) # ((x = x) & (y = x))))))))  :: true
8). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            ((((b = x) & (x = x)) #
                ((b = x) & ((y = x) # ((x = x) & (y = x))))) #
               (((a = x) & (b = x)) &
                  ((x = x) # ((y = x) # ((x = x) & (y = x))))))))  :: true
9). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            ((((b = x) & true) # ((b = x) & ((y = x) # ((x = x) & (y = x))))))
               #
               (((a = x) & (b = x)) &
                  ((x = x) # ((y = x) # ((x = x) & (y = x))))))))  :: true
0). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            ((((b = x) & true) #
                (((b = x) & (y = x)) # ((b = x) & ((x = x) & (y = x))))) #
               (((a = x) & (b = x)) &
                  ((x = x) # ((y = x) # ((x = x) & (y = x))))))))  :: true
1). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            ((((b = x) & true) #
                (((b = x) & (y = x)) # ((b = x) & (true & (y = x))))) #
               (((a = x) & (b = x)) &
                  ((x = x) # ((y = x) # ((x = x) & (y = x))))))))  :: true
2). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            ((((b = x) & true) # (((b = x) & (y = x)) # ((b = x) & (y = x))))
               #
               (((a = x) & (b = x)) &
                  ((x = x) # ((y = x) # ((x = x) & (y = x))))))))  :: true
3). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            ((((b = x) & true) # (((b = x) & (y = x)) # ((b = x) & (y = x))))
               #
               ((((a = x) & (b = x)) & (x = x)) #
                  (((a = x) & (b = x)) &
                     ((y = x) # ((x = x) & (y = x))))))))  :: true
4). ((true # ((y = x) # (y = x))) #
        ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
            ((((b = x) & true) # (((b = x) & (y = x)) # ((b = x) & (y = x))))
               #
               ((((a = x) & (b = x)) & true) #
                  (((a = x) & (b = x)) &
                     ((y = x) # ((x = x) & (y = x))))))))  :: true
```

```
((true # ((y = x) # (y = x))) #
     ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
         ((((b = x) & true) # (((b = x) & (y = x)) # ((b = x) & (y = x))))
             #
             ((((a = x) & (b = x)) & true) #
                 ((((a = x) & (b = x)) & (y = x)) #
                     (((a = x) & (b = x)) & ((x = x) & (y = x))))))))
:: true
((true # ((y = x) # (y = x))) #
     ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
         ((((b = x) & true) # (((b = x) & (y = x)) # ((b = x) & (y = x))))
             #
             ((((a = x) & (b = x)) & true) #
                 ((((a = x) & (b = x)) & (y = x)) #
                     (((a = x) & (b = x)) & (true & (y = x))))))))
:: true
((true # ((y = x) # (y = x))) #
     ((((a = x) & true) # (((a = x) & (y = x)) # ((a = x) & (y = x)))) #
         ((((b = x) & true) # (((b = x) & (y = x)) # ((b = x) & (y = x))))
             #
             ((((a = x) & (b = x)) & true) #
                 ((((a = x) & (b = x)) & (y = x)) #
                     (((a = x) & (b = x)) & (y = x))))))  :: true
(true #
     (((true & (a = x)) #
          ((true & (b = x)) # (true & ((a = x) & (b = x))))) #
         (false # (false # (false # false))))  :: true
(true #
     (((a = x) # ((true & (b = x)) # (true & ((a = x) & (b = x))))) #
         (false # (false # (false # false))))  :: true
(true #
     (((a = x) # ((b = x) # (true & ((a = x) & (b = x))))) #
         (false # (false # (false # false))))  :: true
(true #
     (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
         (false # (false # (false # false))))  :: true
(true #
     (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
         (false # (false # false))))  :: true
(true # (((a = x) # ((b = x) # ((a = x) & (b = x)))) # (false # false))) ::
    true
(true # (((a = x) # ((b = x) # ((a = x) & (b = x)))) # false))  :: true
((false # true) # ((a = x) # ((b = x) # ((a = x) & (b = x)))))  :: true
(true # ((a = x) # ((b = x) # ((a = x) & (b = x)))))  :: true
From 2), and 36), )
(((a = x) # ((b = x) # ((a = x) & (b = x)))) !
                              (true #
                                  ((a = x) #
                                      ((b = x) # ((a = x) & (b = x))))))) ::
    true
(((a = x) # ((b = x) # ((a = x) & (b = x)))) #
     ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
         (((a = x) # ((b = x) # ((a = x) & (b = x)))) &
             (true # ((a = x) # ((b = x) # ((a = x) & (b = x))))))))
:: true
(((a = x) # ((b = x) # ((a = x) & (b = x)))) #
     ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
         (((a = x) & (true # ((a = x) # ((b = x) # ((a = x) & (b = x)))))
             #
             (((b = x) # ((a = x) & (b = x))) &
                 (true # ((a = x) # ((b = x) # ((a = x) & (b = x))))))))) ::
    true
(((a = x) # ((b = x) # ((a = x) & (b = x)))) #
     ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
         ((((a = x) & true) #
              ((a = x) & ((a = x) # ((b = x) # ((a = x) & (b = x)))))) #
```

```
                    (((b = x) # ((a = x) & (b = x))) &
                        (true # ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ;;
              true
41). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       ((a = x) & ((b = x) # ((a = x) & (b = x))))))) #
                  (((b = x) # ((a = x) & (b = x))) &
                      (true # ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ;;
              true
42). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x))))))) #
                  (((b = x) # ((a = x) & (b = x))) &
                      (true # ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ;;
              true
43). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x))))))) #
                  (((b = x) &
                       (true # ((a = x) # ((b = x) # ((a = x) & (b = x))))))) #
                      (((a = x) & (b = x)) &
                          (true #
                              ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ;;
              true
44). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x))))))) #
                  ((((b = x) & true) #
                       ((b = x) & ((a = x) # ((b = x) # ((a = x) & (b = x))))))
                      #
                      (((a = x) & (b = x)) &
                          (true #
                              ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ;;
              true
45). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x))))))) #
                  ((((b = x) & true) #
                       (((b = x) & (a = x)) #
                           ((b = x) & ((b = x) # ((a = x) & (b = x))))))) #
                      (((a = x) & (b = x)) &
                          (true #
                              ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ;;
              true
46). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x))))))) #
                  ((((b = x) & true) #
                       (((b = x) & (a = x)) #
```

```
                                (((b = x) & (b = x)) #
                                    ((b = x) & ((a = x) & (b = x)))))) #
                            (((a = x) & (b = x)) &
                                (true #
                                    ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ::
            true
47). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x)))))) #
                  ((((b = x) & true) #
                       (((b = x) & (a = x)) #
                           (((b = x) & (b = x)) #
                               ((b = x) & ((a = x) & (b = x)))))) #
                      ((((a = x) & (b = x)) & true) #
                          (((a = x) & (b = x)) &
                              ((a = x) # ((b = x) # ((a = x) & (b = x)))))))))) ::
            true
48). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x)))))) #
                  ((((b = x) & true) #
                       (((b = x) & (a = x)) #
                           (((b = x) & (b = x)) #
                               ((b = x) & ((a = x) & (b = x)))))) #
                      ((((a = x) & (b = x)) & true) #
                          ((((a = x) & (b = x)) & (a = x)) #
                              (((a = x) & (b = x)) &
                                  ((b = x) # ((a = x) & (b = x))))))))))
    :: true
49). (((a = x) # ((b = x) # ((a = x) & (b = x)))) #
          ((true # ((a = x) # ((b = x) # ((a = x) & (b = x))))) #
              ((((a = x) & true) #
                   (((a = x) & (a = x)) #
                       (((a = x) & (b = x)) #
                           ((a = x) & ((a = x) & (b = x)))))) #
                  ((((b = x) & true) #
                       (((b = x) & (a = x)) #
                           (((b = x) & (b = x)) #
                               ((b = x) & ((a = x) & (b = x)))))) #
                      ((((a = x) & (b = x)) & true) #
                          ((((a = x) & (b = x)) & (a = x)) #
                              ((((a = x) & (b = x)) & (b = x)) #
                                  (((a = x) & (b = x)) &
                                      ((a = x) & (b = x))))))))))  :: true
50). (true #
          ((a = x) #
              (false #
                  ((b = x) #
                      (false #
                          (((a = x) & (b = x)) #
                              (false #
                                  (false #
                                      (false #
                                          (false #
                                              ((true & (a = x)) #
                                                  ((true & (b = x)) #
                                                      (true &
                                                          ((a = x) &
                                                              (b =
                                                               x))))))))))))))) ::
```

```
          true
51). (true #
        ((a = x) #
            ((b = x) #
                (false #
                    (((a = x) & (b = x)) #
                        (false #
                            (false #
                                (false #
                                    (false #
                                        ((true & (a = x)) #
                                            ((true & (b = x)) #
                                                (true &
                                                    ((a = x) &
                                                        (b = x)))))))))))))))) ;;
          true
52). (true #
        ((a = x) #
            ((b = x) #
                (((a = x) & (b = x)) #
                    (false #
                        (false #
                            (false #
                                (false #
                                    ((true & (a = x)) #
                                        ((true & (b = x)) #
                                            (true &
                                                ((a = x) &
                                                    (b = x))))))))))))))))
    ;; true
53). (true #
        ((a = x) #
            ((b = x) #
                (((a = x) & (b = x)) #
                    (false #
                        (false #
                            (false #
                                ((true & (a = x)) #
                                    ((true & (b = x)) #
                                        (true & ((a = x) & (b = x)))))))))))))) ;;
          true
54). (true #
        ((a = x) #
            ((b = x) #
                (((a = x) & (b = x)) #
                    (false #
                        (false #
                            ((true & (a = x)) #
                                ((true & (b = x)) #
                                    (true & ((a = x) & (b = x))))))))))))
    ;; true
55). (true #
        ((a = x) #
            ((b = x) #
                (((a = x) & (b = x)) #
                    (false #
                        ((true & (a = x)) #
                            ((true & (b = x)) #
                                (true & ((a = x) & (b = x))))))))))) ;; true
56). (true #
        ((a = x) #
            ((b = x) #
                (((a = x) & (b = x)) #
                    ((true & (a = x)) #
                        ((true & (b = x)) #
                            (true & ((a = x) & (b = x))))))))) ;; true
```

```
          ((a = x) #
              ((b = x) #
                  (((a = x) & (b = x)) #
                      ((a = x) #
                          ((true & (b = x)) #
                              (true & ((a = x) & (b = x))))))))))  ;; true
58). (true #
          ((a = x) #
              ((b = x) #
                  (((a = x) & (b = x)) #
                      ((a = x) # ((b = x) # (true & ((a = x) & (b = x))))))))) ;;
           true
59). (true #
          ((a = x) #
              ((b = x) #
                  (((a = x) & (b = x)) #
                      ((a = x) # ((b = x) # ((a = x) & (b = x))))))))  ;; true
60). (true # (false # (false # false)))  ;; true
61). (true # (false # false))  ;; true
62). (true # false)  ;; true
63). (false # true)  ;; true
64). true  ;; true
65). true


24.150000

Aujourd'hui vous allez rencontrer celle qui vous aime en secret.

-> nn

Please give me the term for which you would like the normal form computed:
ssuupprerell99aajjrerell((aajrjelrel((imimasas;; r r11; ;x ;x;yy));; r r11; ; a;a ; bb));
There is error in your term. I'm confused.  I'll ignore it.

0.100000

Vous etes suivi.

m> n

Please give me the term for which you would like the normal form computed:
susupprerell((ajajrerell((imimasa;s;  rr1;i; x x;;  yy)); ; rr11; ; aa; ; bb));;  rr11;;  ;
There is error in your term. I'm confused.  I'll ignore it.

0.166667

Vous etes une merveille de dispositif rare.

m> n

Please give me the term for which you would like the normal form computed:
suspurperell((ajarjelrel((aajrjelrel((imiamsas;;  rir1; ;x ;x; y y));;  rir1; ;a a; ; vvb
The set of normal forms of your term is:
((a = x) & (b = y))  ;; suprel(imas; r1; x; y)
(true # ((a = x) & (b = y)))  ;; ajrel(suprel(imas; r1; x; y); r1; a; b)

Would you like to see the sequences of reduction of your term ? (y/n)

The sequence of term reductions leading to one of the normal forms of your term is:
1). suprel(ajrel(ajrel(imas; r1; x; y); r1; a; b); r1; x; y)
2). ((b = y) & ((a = x) & (r1 = r1)))
     ;; suprel(ajrel(imas; r1; x; y); r1; x; y)
3). ((b = y) & ((a = x) & true))  ;; suprel(ajrel(imas; r1; x; y); r1; x; y)
4). (true & ((a = x) & (b = y)))  ;; suprel(ajrel(imas; r1; x; y); r1; x; y)
```

```
a = x) & (b = y))  :: suprel(ajrel(imas, r1, x, y), r1, x, y)
((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1)))) ::
      suprel(imas, r1, x, y)
(a = x) & (b = y)) & (true & ((x = x) & (r1 = r1)))) ::
      suprel(imas, r1, x, y)
(a = x) & (b = y)) & ((x = x) & (r1 = r1)))  :: suprel(imas, r1, x, y)
(a = x) & (b = y)) & (true & (r1 = r1)))  :: suprel(imas, r1, x, y)
((a = x) & (b = y)) & (r1 = r1))  :: suprel(imas, r1, x, y)
((a = x) & (b = y)) & true)  :: suprel(imas, r1, x, y)
true & ((a = x) & (b = y)))  :: suprel(imas, r1, x, y)
(a = x) & (b = y))  :: suprel(imas, r1, x, y)

 nn

33

ion;votre terminal va s'autodetruire (et vous avec) si vous
z la suite de touches 'uygiuygiugiug'
s aura prevenu.



 give me the term for which you would like the normal form computed:
nerlel((aajjrerell((aajrjelrel((imimvvisis                dede, , rr11, ,x x, , yy)),,  r1r1, ,a a, , bb)),,  xxr
t of normal forms of your term is:
x) & (b = y))  :: imvide
# ((a = x) & (b = y)))  :: ajrel(imvide, r1, a, b)

you like to see the sequences of reduction of your term ? (y/n)

quence of term reductions leading to one of the normal forms of your term is:
prel(ajrel(ajrel(imvide, r1, x, y), r1, a, b), r1, x, y)
b = y) & ((a = x) & (r1 = r1)))
: suprel(ajrel(imvide, r1, x, y), r1, x, y)
b = y) & ((a = x) & true))  :: suprel(ajrel(imvide, r1, x, y), r1, x, y)
rue & ((a = x) & (b = y)))  :: suprel(ajrel(imvide, r1, x, y), r1, x, y)
a = x) & (b = y))  :: suprel(ajrel(imvide, r1, x, y), r1, x, y)
(a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1)))) ::
      suprel(imvide, r1, x, y)
(a = x) & (b = y)) & (true & ((x = x) & (r1 = r1)))) ::
      suprel(imvide, r1, x, y)
(a = x) & (b = y)) & ((x = x) & (r1 = r1)))  :: suprel(imvide, r1, x, y)
(a = x) & (b = y)) & (true & (r1 = r1)))  :: suprel(imvide, r1, x, y)
((a = x) & (b = y)) & (r1 = r1))  :: suprel(imvide, r1, x, y)
((a = x) & (b = y)) & true)  :: suprel(imvide, r1, x, y)
true & ((a = x) & (b = y)))  :: suprel(imvide, r1, x, y)
(a = x) & (b = y))  :: suprel(imvide, r1, x, y)
(a = x) & (b = y))  :: imvide



quence of term reductions leading to one of the normal forms of your term is:
prel(ajrel(ajrel(imvide, r1, x, y), r1, a, b), r1, x, y)
rue # ((b = y) & ((a = x) & (r1 = r1)))) ::
      ajrel(suprel(ajrel(imvide, r1, x, y), r1, x, y), r1, a, b)
rue # ((b = y) & ((a = x) & true))) ::
      ajrel(suprel(ajrel(imvide, r1, x, y), r1, x, y), r1, a, b)
rue # (true & ((a = x) & (b = y)))) ::
      ajrel(suprel(ajrel(imvide, r1, x, y), r1, x, y), r1, a, b)
rue # ((a = x) & (b = y)))
: ajrel(suprel(ajrel(imvide, r1, x, y), r1, x, y), r1, a, b)
true # ((a = x) & (b = y))) & ((y = y) & ((x = x) & (r1 = r1)))) ::
      ajrel(suprel(imvide, r1, x, y), r1, a, b)
true & ((y = y) & ((x = x) & (r1 = r1)))) #
   (((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1))))) ::
      ajrel(suprel(imvide, r1, x, y), r1, a, b)
```

```
(((y = y) & ((x = x) & (r1 = r1))) #
     (((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1))))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
((true & ((x = x) & (r1 = r1))) #
     (((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1))))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(((x = x) & (r1 = r1)) #
     (((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1))))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
((true & (r1 = r1)) #
     (((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1))))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
((r1 = r1) # (((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1))))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # (((a = x) & (b = y)) & ((y = y) & ((x = x) & (r1 = r1))))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # (((a = x) & (b = y)) & (true & ((x = x) & (r1 = r1))))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # (((a = x) & (b = y)) & ((x = x) & (r1 = r1)))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # (((a = x) & (b = y)) & (true & (r1 = r1)))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # (((a = x) & (b = y)) & (r1 = r1))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # (((a = x) & (b = y)) & true)) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(truu # (true & ((a = x) & (b = y)))) ::
       ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # ((a = x) & (b = y)))
  :: ajrel(suprel(imvide, r1, x, y), r1, a, b)
(true # ((a = x) & (b = y)))  :: ajrel(imvide, r1, a, b)

50000

analystes numericiens le font a une approximation pres.
```

A recoller sur feuille reliée

RESUME

L'objet de cette thèse est la spécification, à l'aide de types abstraits algébriques, d'un système permettant la recherche d'images par le biais d'une base contenant leurs descriptions, système bâti autour d'un système expert simulant l'expertise d'un documentaliste de l'image.

La première partie est consacrée à la présentation du projet EXPRIM (EXpert Pour la Recherche d'IMages), à la mise en évidence de la spécificité de l'information image, à la présentation de l'architecture du système avec ses composantes logicielles et matérielles, et à la description du poste de manipulation d'images (poste de travail) et ses diverses fonctions.

La deuxième partie est consacrée à une approche formelle des systèmes experts. On y décrit de façon algébrique différents modes de fonctionnement d'un moteur d'inférence et on y aborde les problèmes de stratégies et de prise en compte du raisonnement approximatif dans un système expert.