INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

ETUDE ALGEBRIQUE ET RELATIONNELLE DES TYPES ABSTRAITS ET DE LEURS REPRESENTATIONS

THESE

PRÉSENTÉE POUR L'OBTENTION DU
GRADE DE DOCTEUR D'ÉTAT
MENTION SCIENCES

SOUTENUE LE 11 SEPTEMBRE 1979

PAR

Pierre LESCANNE

DEVANT LE JURY

RAPPORTEURS M. SINTZOFF

EXAMINATEURS D. COULON
J.C. DERNIAME
M. NIVAT

INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

ETUDE ALGEBRIQUE ET RELATIONNELLE DES TYPES ABSTRAITS ET DE LEURS REPRESENTATIONS

THESE

PRÉSENTÉE POUR L'OBTENTION DU
GRADE DE DOCTEUR D'ÉTAT
MENTION SCIENCES

SOUTENUE LE 11 SEPTEMBRE 1979

PAR

Pierre LESCANNE

DEVANT LE JURY

RAPPORTEURS M. SINTZOFF

EXAMINATEURS D. COULON
J.C. DERNIAME
M. NIVAT

Malgré ses multiples charges, Claude PAIR, Président de l'Institut National Polytechnique de Lorraine et directeur du Centre de Recherche en Informatique de Nancy a suscité puis dirigé et suivi de très près mes recherches, toujours prêt à proposer, critiquer et suggérer. On connaît sa contribution à l'approfondissement et la clarification des concepts de la programmation. Je sais la chance que j'ai eue de travailler avec lui et dans l'environnement scientifique qu'il a créé à Nancy, je lui suis très reconnaissant pour tout ce que je lui dois et lui adresse ici mes sincères remerciements.

Parrain de mes recherches désigné par le Centre National de la Recherche Scientifique, Jean-François PERROT, professeur à l'Université Pierre et Marie Curie m'a montré le sens qu'il donnait à cette charge en me conseillant toujours le plus judicieusement, qu'il trouve ici le témoignage de la reconnaissance de son filleul.

Michel SINTZOFF, du Philips Research Laboratory à Bruxelles, a, durant l'année qu'il a passée à Nancy en 1977, puis au cours de ses visites, participé à ces recherches par les discussions que nous avons eues. Ainsi, j'ai pu profiter de ses nombreuses, pertinentes et plaisantes remarques. Je lui adresse ici ma gratitude.

Maurice NIVAT, professeur à l'Université de Paris VII, a contribué à promouvoir l'approche algébrique dans la programmation, je le remercie de s'intéresser à mes travaux et de participer à ce jury.

Jean-Claude DERNIAME, maître de conférences à l'Université de Nancy I a étudié les types abstraits comme outil modulaire de programmation de grands logiciels. En tant que directeur du département de mathématiques appliquées, il m'a accueilli et a mis à ma disposition des moyens matériel. Je n'oublie pas que c'est lui qui m'a enseigné la programmation et c'est un plaisir pour moi de le voir figurer dans mon jury.

Daniel COULON, maître de conférences à l'Institut Polytechnique de Lorraine a bien voulu participer à mon jury ; dans ses travaux de programmation, il a traité beaucoup de types de données et je le remercie d'apporter ici l'oeil de l'utilisateur avisé.

Je remercie tous mes amis et collègues du Centre de Recherche en Informatique de Nancy mais parmi eux, je tiens à citer deux groupes avec lesquels j'ai particulièrement collaboré : le groupe de recherche Castor (noté C) et le groupe Livercy (noté L) parmi lesquels : tout d'abord Jean-Luc REMY (C & L) mais aussi Françoise BELLEGARDE (C), Jean-Pierre FINANCE (C & L), Monique GRANDBASTIEN (C & L), Jacques GUYARD (C), Jacques JARAY (C), Pierre MARCHAND (L), Jean MOROLDT (C), Roger MOHR (L), Alain QUERE (C & L) et Fernand REINIG (C).

Je tiens à remercier mes collègues de l'Institut de Recherche en Informatique et en Automatisme : Marie-Claude GAUDEL, Gérard HUET et Gérard TERRINE qui chacun m'ont apporté les aspects très intéressants de leurs recherches. Marie-Claude GAUDEL m'a entre autres, permis de rencontrer John GUTTAG du Massachusetts Institut of Technology avec lequel j'ai eu de passionnants entretiens, je tiens à le remercier spécialement, je dois beaucoup à ses travaux ainsi qu'il apparaîtra à la lecture de cette thèse.

J'adresse aussi mes remerciements à mes amis de l'Institut Universitaire de Calcul Automatique de Lorraine et du Département Informatique de l'Institut Universitaire de Technologie qui m'ont accueilli quand je programmais.

C'est à Martine TESOLIN que les lecteurs doivent cette belle dactylographie je pense qu'ils s'associent à moi pour la remercier de son très beau travail.

Enfin, je dis un grand merci à Monique, Etienne et Cyrille, qui ont accepté que je sacrifie parfois la famille concrète aux types abstraits.

ETUDE ALGEBRIQUE ET RELATIONNELLE DES TYPES

ABSTRAITS ET DE LEURS REPRESENTATIONS.

RESUME :

Les résultats essentiels de cette thèse concernent la représentation des types abstraits par des modèles algébriques : en effet, on construit un modèle de type abstrait ou algèbre de type (l'algébre mère) dont les objets sont aussi des algèbres, dites filles, et dont les opérations portent sur ces algèbres. Après avoir décrit une famille d'algèbres filles susceptibles de représenter le type abstrait, arbres binaires, le théorème de la représentation algébrique des arbres binaires affirme qu'il s'agit bien d'un modèle. Le théorème de la représentation canonique montre que, si un type abstrait est "correctement spécifié", un tel modèle existe toujours, la preuve est faite, en construisant explicitement ce modèle. En général, les modèles ne sont pas isomorphes ; de fait, différentes représentations sont proposées pour les ensembles et pour les graphes. La proposition de complète discrimination caractérise les types abstraits dont les modèles sont isomorphes.

D'autre part, dans cette thèse, le problème de l'indéterminisme est abordé à deux niveaux :

- au niveau des opérations du type abstrait, les théorèmes de C-stabilité et du P-stabilité donnent des conditions sur les équations pour que le type abstrait soit consistant quand on admet des opérations indéterministes.
- au niveau des représentations le théorème de la représentation relationnelle des listes est le symétrique du théorème de la représentation algébrique des arbres binaires, quand on admet que les opérations des algèbres filles soient des relations.

ALGEBRAIC AND RELATIONNAL STUDY OF

ABSTRACT DATA TYPES AND THEIR REPRESENTATIONS

ABSTRACT :

The main results of this thesis concern the representation of abs data types by algebraic models : in fact, a model of abstract data type a type algebra (the mother algebra) is constructed. The objects of the ty algebra are also algebras (daughter algebras) and its operations work on algebras. After a family of daughter algebras are described to represent abstract data types, binary trees, the algebraic representation of binar trees theorem saids that it is a model. The canonical representation the proofs that such a model allways exists if the abstract data type is "co specified". The proof is performed by constructing this model. Generally the models are not isomorphic. As a matter of fact, several representati are proposed for the sets and the graphs. The complete discrimination pr sition characterizes the abstract data types, the models of which are is

On the other hand, the indeterminism problem is approached on two

- at the level of the operations of an abstract data type, the C-stability theorem and the P-stability one give conditions abo the equations that the abstract data type may be consistent, wh indeterministic operations exist.
- at the level of the representations, the relationnal represent of lists theorem is the symmetrical one of the algebraic repres of binary trees theorem when the operations of the daughter alg are relations.

In conclusion the problem of the proof in algebraic systems start the relationnal algebras has been studied and some experimental results proposed.

INTRODUCTION

LES PROBLÈMES DE LA PROGRAMMATION.

L'activité de programmation se décompose en deux étapes majeures, la première consiste à passer d'un énoncé de problèmes à un algorithme, la seconde à passer d'un *algorithme sur des objets abstraits* à un *programme sur des données*, pour aboutir à une exécution sur une machine. Dans cette thèse nous ne parlerons que de la seconde étape. Dans le sens que l'on donne ici, un *algorithme* est une description des opérations que doivent subir les entrées du problème pour aboutir aux sorties, ou résultats.

L'algorithme porte sur des *objets abstraits* proches du langage mathématique. Si l'on dispose de bons outils (cf. figure 1 et 2) l'algorithme ainsi que la description des objets abstraits, ne font pas référence à une exécution ou à un déroulement de calcul ;
on peut employer à leur propos le qualificatif de *statique*.

On peut, par exemple, utiliser pour exprimer des algorithmes, des langages applicatifs permettant de définir des fonctions récursives, [BAC 78] , ou des langages itératifs "à affectation unique" définissant des suites (LUCID [ASH 77] , MEDEE [BEL 78]). Pour les objets abstraits, une description du même ordre est possible ; par exemple, une spécification algébrique permet de décrire les *types abstraits* uniquement par des relations entre des opérations portant sur eux.

Un *programme* est au contraire une description de *calculs*, exécutables par une machine ; aussi les *données* qu'il traite doivent-elles être manipulables par la machine. Les actions qui composent le calcul sont des modifications de l'état mémoire.

Si plusieurs méthodes ont été proposées pour le passage d'un algorithme à un programme [ARS 76], [BUR 76] , [PAI 79] , [BAJ 79] , le passage d'un type abstrait à une *structure de données* a eté jusqu'à présent moins

bien résolu. Les problèmes qui se posent peuvent être classés en trois catégories : problème de la représentation, problème de l'affectation, problème du partage de données.

Le problème de la représentation peut être imagé de la façon suivante : "un objet du type abstrait est une boîte noire munie de boutons : pour obtenir une valeur externe, on appuie sur le bouton approprié, certaines opérations permettent de combiner ces boîtes. L'utilisateur ne peut ni ne veut savoir ce qu'il y a à l'intérieur de ces boîtes, mais seulement, comment elles réagissent aux sollicitations extérieures.

```
Type alphabet
    fonctionnalité
        [',']','(',')' , 'a', 'b', 'c', '#' :() →  Alphabet ;
        EG :(Alphabet  ,  Alphabet)→ Bool ;
        E Alph :(Alphabet) → Bool
    axiomatique
        EG ('(', ')') = VRAI, ....
        EG ('(', ' ') = FAUX, ....
        E Alph (a) = EG (a, 'a') ou  EG (a, 'b') ou  EG (a, 'c')

Type Pile [Alphabet]
    fonctionnalité
        PILE VIDE :() → Pile ;
        EMPIL :(Pile , Alphabet) → Pile  ;
        DEPIL :(Pile)→ File ;
        SOMMET :(Pile) →  Alphabet U {INDEF}  ;
        PVIDE :(Pile) → Bool
    axiomatique
        DEPIL (PILEVIDE) = PILEVIDE ; SOMMET (PILEVIDE) = INDEF ;
        DEPIL EMPIL (p, a)) = p ;     SOMMET (EMPIL (p,a)) = a ;
        PVIDE(PILEVIDE) = VRAI  ;
        PVIDE(EMPIL (P, a)) = FAUX  ;
```

Figure 2 : Une spécification algébrique d'un type abstrait.

parenthese : vrai si le mot donné est correctement parenthésé et faux sinon

pfinal : pile de reconnaissance résultant de l'examen d'un mot

erreurfinal : cohérence du parenthésage dans le mot

pile : suite d'états d'une pile de reconnaissance

erreur : suite qui devient fausse dès qu'une erreur est détecté au cours de l'analyse

fin-de-mot-ou-erreur : condition d'arrêt de l'examen de caractères

Figure 1 :
une description d'algorithme
à la MEDEE

pfinal, errfinal = *dernier* (pile), *dernier*(erreur)
où pile(i), erreur(i) : *pour* i *dans* 0...
jqa fin-de-mot-ou-erreur (i)

pile (i), erreur (i) , fin-de-mot-ou-erreur (i)
caractère = *donnée* caractère

pile, erreur = *cas* caractère *eg* [' *alors* erreur = FAUX
pile = EMPIL(pile,'[')
caractère *eg* ']' *alors* erreur = *non* (SOMMET(pile)*eg* '[')
pile = DEPIL (pile)
caractère *eg* '(' *alors* erreur = FAUX, pile = EMPIL(pile,'(')
caractère *eg* ')' *alors* erreur = *non* (SOMMET(pile)*eg*'('),
pile = DEPIL(pile)
caractère ∈ Alph *alors* erreur : FAUX, pile = pile
fin-de-mot-ou-erreur = erreur *ou* caractère *eg* '#'
premier (pile) = PILE VIDE
premier (caractère) = *premier*
premier (fin-de-mot-ou-erreur) =(*donnée* (caractère) '#'

Représenter un type abstrait c'est "donner une description possible de la constitution interne des boîtes". On peut envisager de construire plus ou moins automatiquement cette représentation.

Le *problème de l'affectation* intervient dès qu'on veut exécuter une suite de calculs et faire varier des valeurs. L'utilisation d'un type abstrait dans un algorithme admet que l'on dispose à tout instant de tous les objets du type, au moins potentiellement ; cela est tout à fait possible dans des cas très particuliers , par exemple, dans un calculateur, les entiers peuvent être notés de façon simple et sont accessibles à tout moment. Mais dans le cas de types constuits comme les piles, il ne peut en être question : à un instant donné, on ne dispose que d'un nombre restreint d'objets accessibles via des identificateurs et ainsi effectivement présents en mémoire ; ces identificateurs peuvent d'ailleurs se partager des objets. Dans l'algorithme de la figure 1, par exemple, un identificateur est associé à une suite de piles obtenues par des empilements et dépilements successifs ; dans un programme, il désignera, à un instant donné, la dernière pile de la suite ainsi il est inutile de pouvoir accéder à toutes les piles. Les questions qui se posent sont alors : comment représenter les piles pour que l'opération d'empilement soit la plus simple possible ? Dans la mesure où seule la dernière pile calculée reste intéressante, on peut l'obtenir par modification de la pile précédente. Une autre question alors se pose : comment minimiser ces modifications ? Cette dernière question conduit à l'idée de modification in situ de l'objet, autrement dit, on fait sur l'objet le minimum d'opérations sans le "déplacer". En résumé, on doit chercher à minimiser les duplications d'objets dans le temps.

Le *problème du partage des données* intervient dès qu'on veut utiliser au mieux la place en mémoire, en faisant en sorte que deux objets différents aient accès à des sous-objets communs qui n'ont pas besoin d'être dupliqués. Là, il s'agit de minimiser la duplication dans l'espace.

La réponse à ces trois problèmes suppose que l'on connaisse bien la représentation des objets et que l'on en possède une bonne modélisation

mathématique,ainsi les raisonnements seront très rigoureux et on pourra même envisager de les automatiser.
Il semble que beaucoup d'aspects découlent d'une bonne approche du concept de représentation. C'est ce à quoi je vais m'attacher en essayant d'en tirer certaines conséquences en ce qui concerne les deux autres problèmes posés.

Dans cette thèse, deux études sont proposées :
une étude algébrique qui formalise le concept d'ensemble muni d'opérations et une étude relationnelle qui axiomatise les relations dans le but de décrire les objets. Une troisième approche existe, elle est fondée sur le calcul des prédicats du premier ordre ; on la trouvera exposée dans la thèse de REMY [REM 74] , ou les articles de PAIR [PAI 74] ou FINANCE [FIN 76] .

L'ACTIVITÉ DU PROGRAMMEUR AUX PRISES AVEC LES REPRÉSENTATIONS D'OBJETS.

En résumé la démarche qui consiste à représenter des types abstraits par les objets informatiques disponibles dans un langage de programmation passe par une étape qui est la représentation en termes d'objets mathématiques ; ce sont ces objets qui sont décrits ensuite par les outils informatiques. En fin de compte, ces trois entités forment les sommets du triangle dessiné dans la figure 3. Qand on a trouvé la fonction décrite par le côté pointillé (c) , on a résolu le problème de la représentation : comme on connaît b, le problème central est a .

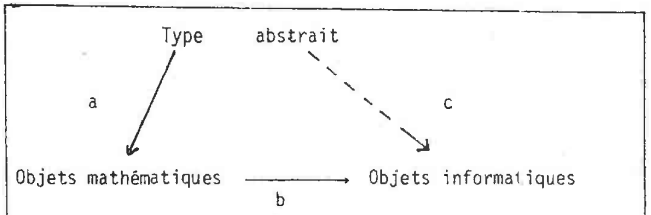


Figure 3 : Les démarches de la représentation.

Les objets mathématiques, dits de représentation, peuvent être considérés d'une autre manière ; ce sont les premiers objets que le programmeur envisage et sur lesquels il exerce son intuition. Une fois qu'à l'aide de ces objets, il a dégagé les relations entre les opérations, il en déduit si toutefois il ne le connaît pas déjà, le type abstrait avec lequel il va écrire l'algorithme. Eventuellement, il ajoute de nouvelles opérations à ce type abstrait. Il se peut que lors de l'utilisation du type abstrait dans des algorithmes telle opération apparaisse plus souvent que telle autre et demande à être implantée plus efficacement, cela peut amener le programmeur à réexaminer les objets mathématiques sur lesquels il exerce son intuition et son raisonnement, cela peut le conduire à envisager une autre famille de représentations qui améliorera les implantations. (Voir par exemple [FRA 78]

LES OBJECTIFS DE CETTE THÈSE

Les résultats essentiels de cette thèse concernent donc la représentation des types abstraits par des modèles algébriques : en effet, on construit un modèle de type abstrait ou algèbre de type (l'algèbre mère) dont les objets sont aussi des algèbres, dites filles, et dont les opérations portent sur ces algèbres. Après avoir décrit une famille d'algèbres filles susceptibles de représenter le type abstrait, arbres binaires, le théorème de la représentation algébrique des arbres binaires affirme qu'il s'agit bien d'un modèle. Le théorème de la représentation canonique montre que, si un type abstrait est "correctement spécifié", un tel modèle existe toujours, la preuve est faite, en construisant explicitement ce modèle. En général, les modèles ne sont pas isomorphes ; de fait, différentes représentations sont proposées pour les ensembles et pour les graphes .
La proposition de complète discrimination caractérise les types abstraits dont les modèles sont isomorphes

D'autre part, dans cette thèse, le problème de l'indéterminisme est abordé à deux niveaux :

- au niveau des opérations du type abstrait, les théorèmes de $\mathfrak{C}$-stabilité et de $\mathbb{P}$-stabilité donnent des conditions sur les équations pour que le type abstrait soit consistant quand on admet des opérations indéterministes.
- au niveau des représentations, le théorème de la représentation relationnelle des listes est le symétrique du théorème de la représentation algébrique des arbres binaires, quand on admet que les opérations des algèbres filles soient des relations.

Enfin, posé à l'occasion de l'algèbre relationnelle, le problème de la preuve dans les systèmes algébriques est abordé, des résultats expérimentaux sont proposés.

LE PLAN DE CETTE THÈSE.

Ainsi cette thèse se décompose en deux parties : l'étude algébrique (chapitres 1 2, 3 et 4) et l'étude relationnelle (chapitres 5 et 6).

Chapitre 1 :

Ce chapitre prépare l'étude algébrique de la représentation de trois façons. Dans un paragraphe introductif (§ 1), nous précisons la position du problème. D'autre part, sur divers exemples, nous étudions l'approche algébrique de la représentation (§ 2 et § 6). L'un deux (§ 2) est particulièrement développé, il s'agit des arbres binaires, il est simple mais suffisamment riche pour bien introduire au problème, les conséquences proprement informatiques, en rapport avec les langages de programmation, sont aussi envisagées. On démontre pour ce type un théorème de la représentation algébrique. D'autre part, enfin, divers paragraphes (§ 3, § 4, § 5) sont consacrés à la présentation des concepts algébriques utiles dans la spécification algébrique d'un type abstrait : algèbres homogènes, algèbres hétérogènes, algèbres paramétrées.

Chapitre 2 :

Ce chapitre comporte essentiellement deux parties : dans la première partie, sont présentés les divers aspects algébriques des types abstraits : réécriture (§ 1), les algèbres de type (§ 2) , la représentation algébrique (§ 3).
A l'occasion de la réécriture on définit les concepts de confluence, de noethérianité et de suffisante complétude qui précisent ce qu'on entend par type abstrait "correctement spécifié". Dans la deuxième partie (§ 4), est exposée la représentation canonique d'un type abstrait.

Chapitre 3 :

On trouvera, dans ce chapitre, une étude algébrique du problème qui apparaît dès qu'on cherche à introduire l'indéterminisme dans un type abstrait, à savoir : quand peut-on étendre les opérations à l'ensemble des parties en conservant les équations ? La condition de linéarité (chaque variable apparaît au plus une fois dans chaque membre) donne la $\mathfrak{C}$-stabilité i.e. la stabilité quand on étend les opérations aux parties non vides. La régularité (même ensemble de variables dans chaque membre) et la linéarité donne la $\mathfrak{P}$-stabilité i.e la stabilité quand on étend les opérations à l'ensemble de toutes les parties.

Chapitre 4 :

Dans ce chapitre, sont traités des exemples. Les différents types, qui peuvent avoir les mots sur un alphabet comme modèles, sont envisagés (§ 1). Le type abstrait graphe (§ 3) est une alternative du type ensemble comme exemple de type sans totale discrimination,c'est à dire ayant plusieurs modèles non isomorphes. Les tables de symboles illustrent le chapitre 3 (§ 4).

Chapitre 5 :

Après avoir présenté des raisons qui orientent vers des opérations relationnelles dans les algèbres filles (§ 1), une approche des algèbres

relationnelles est proposée (§ 2 et § 3). La représentation des listes linéaires (§ 4), largement développée, conduit au théorème de la représentation relationnelle des listes linéaires (§ 4.9).

Chapitre 6 :

Dans le but de réfléchir aux possibilités pratiques d'automatisation des preuves dans les systèmes algébriques, j'ai développé un outil essentiellement fondé sur l'unification. Celle-ci est décrite en termes de types abstraits, ensuite divers résultats expérimentaux sont proposés notamment au sujet des groupes.

TABLE DES MATIERES

CHAPITRE 1 : CADRE ALGEBRIQUE POUR L'ETUDE DES TYPES ABSTRAITS

1.- Introduction 16

2.- Arbres binaires 18

2.1.- Algèbres "les arbres binaires étiquetés par Alph

2.2.- Algèbres "un arbre binaire"

2.3.- Liens avec la programmation

2.4.- Arbres binaires et partages

3.- Rappels sur les algèbres homogènes 37

3.1.- Algèbres et algèbres partielles

3.2.- Constructions d'algèbres

3.3.- Algèbres libres et algèbres initiales

3.4.- Algèbres engendrées et algèbres premières

3.5.- Fonctions polynomiales

3.6.- Identités sur les algèbres

4.- Rappels sur les algèbres hétérogènes 45

4.1.- Algèbres hétérogènes et algébres hétérogènes partielles

4.2.- Algèbres hétérogènes libres et algèbres hétérogènes initiales

4.3.- Algèbres hétérogènes engendrées et algèbres hétérogènes premières

4.4.- Algèbres terminales

4.5.- Fonctions polynomiales

4.6.- Identités sur les algèbres hétérogènes

5.- Algèbres homogènes paramétrées 52

6.- Deux exemples : les files et les ensembles 55

6.1.- Les files

6.2.- Les ensembles

CHAPITRE 2 : ETUDE DE LA REPRESENTATION ALGEBRIQUE D'UN TYPE ABSTRAIT

1.- Types abstraits et réécritures 62

1.1.- Systèmes de réécritures

1.2.- Confluence dans le cas des types abstraits

1.3.- Terminaisons dans le cas des types abstraits

2.- Types abstraits et algèbres de types 71

3.- Algèbres hétérogènes et paramétrées et représentation algébrique d'un type abstrait 75

3.1.- Description des petites algèbres

3.2.- Les constructions comme représentation des opérations

3.3.- Eléments d'une méthodologie

4.- Une représentation canonique 79

4.1.- Rappels sur les ramifications

4.2.- Description de termes comme des ramifications et réécritures

4.3.- Le théorème de la représentation canonique

4.4.- L'exemple des listes circulaires

CHAPITRE 3 : OUTILS ALGEBRIQUES POUR LE CALCUL SUR LES PARTIES D'UNE ALGEBRE DE TYPE

1.- Introduction 92

2.- Etude d'exemples 93

2.1.- Les groupes

2.2.- Les algèbres linéaires

2.3.- La conditionnelle

2.4.- Les arbres binaires

3.- Equations linéaires et équations linéaires et régulières 98

4.- Les ensembles 99

5.- Présentation informelle des résultats sur les extensions 101

6.- C-extension d'une algèbre et $\mathbb{P}$-extension 102

6. 6.1.- Ensembles stables de sortes

6.2.- $\mathbb{P}$-extension d'une algèbre

6.3.- $\mathbb{C}$-extension d'une algèbre

7.- Prolongement des identités et variétés C-stables 105

8.- Variétés $\mathbb{P}$-stables 112

9.- Ensembles algébriques 115

9.1.- Solution dans le cas des variétés $\mathbb{P}$-stables

9.2.- Solution d'un système $\mathbb{C}$-acceptable dans le cas des variétés C-stables

10.- Bibliographie 119

CHAPITRE 4 : EXEMPLES DE TYPES ABSTRAITS

1.- Mots 121

1.1.- Algèbres à deux sortes : mots et booléens

1.2.- Algèbres à trois sortes : mots, alphabet et booléens

1.2.1.- Spécification du type Listlin

1.2.2.- Propriétés du type Listlin

1.2.3.- Une représentation du type Listlin

2.- Couples 134

3.- Graphes 134

3.1.- Une première représentation

3.2.- Une deuxième représentation

4.- Tables des symboles 139

CHAPITRE 5 : PRESENTATION GENERALE DU CALCUL RELATIONNEL ET SON APPLICATION A LA REPRESENTATION D'UN TYPE ABSTRAIT

1.- Introduction 142

1.1.- Un exemple préliminaire

1.2.- Quelques avantages des algèbres relationnelles

1.3.- Les algèbres relationnelles , une spécification de type abst

2.- Les algèbres relationnelles : partie syntaxique 145

2.1.- Description des relations

2.1.1.- Restriction

2.1.2.- Priorité et parenthésage

2.1.3.- Inutilité de la complémentation

2.2.- Ecriture d'un prédicat

2.3.- Ecriture d'une assertion

3.- Les algèbres relationnelles : partie axiomatique 149

3.1.- Axiomes pour les algèbres relationnelles

3.2.- Autres axiomes

3.3.- Axiomes d'itération

4.- Etude d'un exemple de représentation:les listes linéaires 155

4.1.- Représentation des objets

4.2.- Représentation de l'opération Rest

4.3.- Preuve de la correction de l'opération Rest

4.4.- Représentation de l'opération ⊂

4.5.- Preuve de la correction de ⊂

4.6.- La liste linéaire Vide

4.7.- La représentation de Tête

4.8.- Preuve de l'équation Reste(⊂ (A. $\mathcal{L}$))\ = $\mathcal{L}$

4.9.- Théorème de la représentation relationnelle des listes

5.- Conclusions et bibliographie 168

CHAPITRE 6 : EXPERIENCES D'AUTOMATISATION DES PREUVES DANS LES SYSTEMES ALGEBRIQUES

1.- Introduction 169

2.- Unification 171

2.1.- Principes généraux

2.2.- La précédure UNIFIER une première forme d'unification

2.3.- Une première version de UNIF

2.4.- Justification de la première version de UNIF

2.5.- Une deuxième version de UNIF

2.6.- Justification de la deuxième version de UNIF

2.7.- Commentaires

3.- Résolution 185

4.- Réfutation 186

5.- Le cas des groupes 194

6.- Conclusions et perspectives 210

CHAPITRE I

cadre algébrique pour l'étude des types abstraits

CHAPITRE I

CADRE ALGÉBRIQUE POUR L'ÉTUDE DES TYPES ABSTRAITS

1.- INTRODUCTION.

L'objectif de cette étude est double :

- *harmoniser* autour du concept d'algèbre, diverses approches des structures de données

- proposer *un cadre algébrique pour une théorie de la représentation* des types abstraits dans un langage de programmation avec affectation, donc avec une sémantique comportant des changements d'état.

On peut classer en deux grandes familles, les formalismes à propos des structures de données [GUT 79] .

- Dans un premier type d'approche un objet d'une certaine structure est susceptible d'évoluer dans le temps, de se modifier ; mais, pour rester dans la structure, il doit vérifier au cours du temps, soit un prédicat invariant (c'est l'approche de HOARE [HOA 72] , WULF [WUL 76]), soit une famille d'axiomes et ses conséquences (c'est l'approche de PAIR [PAI 74] , REMY [REM 74] et FINANCE [FIN 76], voir aussi [GAU 77]).

Une *construction* d'un nouvel objet à partir d'autres objets est décrite par un programme dans l'approche de Hoare et par adjonction ou modification d'axiomes dans l'approche des Nancéens. Pour des raisons d'harmonisation et d'une plus grande simplicité mathématique que nous justifierons plus loin, nous proposons ici de décrire un objet comme une algèbre ; nous

parlerons "d'algèbres filles" par opposition à l'"algèbre mère" ou *algèbre de type* fondement mathématique de la description à l'aide des types abstrait L'invariance ou l'appartenance à la structure se traduit alors par le maintien de l'algèbre fille dans une *classe d'algèbres*.Cette classe d'algèbres est caractérisée par des propriétés du premier et du second ordre, en particulier cette classe n'est pas une classe équationnelle ou variété. On construit une nouvelle algèbre fille en exprimant ses opérations à partir de celles d'une ou plusieurs autres algèbres. Si l'on désire non pas construire une nouvelle algèbre, mais fournir un élément fixé d'une algèbre, on parle alors de *sélection*. Les opérations dans les algèbres filles qui servent à "retrouver" un élément de l'algèbre à partir d'autres éléments sont appelées des *accès*.construction et accès sont les deux concepts qui gouvernent une telle approche, le vocabulaire algébrique permet de les caractériser agréablement.

Au chapitre 5 une autre approche que l'on peut toujours classer dans la même famille, est proposée, elle est fondée sur la théorie des algèbres relationnelles de Tarski et reprend et développe des idées proposées par de Bakker , Hitchcock, Park et de Roever ([BAK 72] , [HIT 72] , [ROE 74] voir aussi [LIV 78]). Elle rappelle l'approche de Finance, Pair et Remy et profite de l'outil algébrique pour apporter une grande rigueur et envisager une certaine automatisation (chapitre 6).

- Dans la deuxième famille d'approches on utilise une description algébrique, à l'aide d'opérations et d'axiomes sur ces opérations, pour caractériser un *type abstrait*, c'est à dire l'ensemble des objets auxquels on s'intéresse et les transformations qui les affectent. [GUT 77] [GOG 75] [LIS 77] . Cette approche est plus abstraite, puisque d'un seul coup on envisage tous les objets et leurs constructions.

2.- ARBRES_BINAIRES.

2.1.- ALGÈBRES "LES ARBRES BINAIRES ÉTIQUETÉS PAR ALPH".

On s'aperçoit immédiatement que deux *sortes* d'objets vont intervenir, ce sont les arbres proprement dits et les valeurs ; notons les premiers Arb et les seconds Val ; ici Val = Alph U {INDEF} où INDEF $\notin$ Alph . On définit ensuite diverses opérations qui vont permettre de manipuler les arbres, plus précisément d'extraire des sous-arbres et de construire de nouveaux arbres. Chaque opération a un profil, c'est à dire une description de ses opérandes et de son résultat. Ainsi CONS qui construit un arbre binaire, à partir de deux arbres et d'une valeur, a pour profil

$$(Arb, Val, Arb) \longrightarrow Arb \quad .$$

ce qui signifie que son premier opérande est un arbre que son deuxième opérande est une valeur , que son troisième opérande est un arbre et que son résultat est un arbre.
La notation sera conservée dans la suite

De même les opérations GAU, DRO ont pour profil

$$(Arb) \longrightarrow Arb$$

elles font correspondre à un arbre x , un autre arbre appelé partie gauche ou partie droite de l'arbre x

TET a pour profil

$$(Arb) \longrightarrow Val$$

elle fait correspondre à un arbre une valeur : l'étiquette de la tête de l'arbre.

On introduit ainsi une structure algébrique que l'on appelle *algèbre hétérogène* ou algèbre à plusieurs sortes d'objets, une telle algèbre est caractérisée par une famille d'ensembles : les *sortes** et une famille d'*opérations* de profil donné.

* Certains auteurs, notamment les logiciens, parlent dans ce cas de types ; ce mot a divers sens chez les informaticiens, rarement celui d'ensemble d'objets [MOR 73] , c'est pourquoi nous avons préféré le mot sorte.

Les opérations CONS, GAU, DRO, TET vérifient entre elles, un certain nombre d'identités qui caractérisent les arbres binaires et traduisent des propriétés bien connues du programmeur :

(1) GAU (CONS (x, v, y)) = x
(2) DRO (CONS (x, v, y)) = y
(3) TET (CONS (x, v, y)) = v

Ces trois identités donnent les propriétés des opérations sur les arbres binaires, mais ne permettent pas de dire ce qu'est un arbre binaire. Voici une réponse possible : un arbre binaire est un objet qui peut être décrit par une expression constante. Or pour parler d'expression constante, il faut au moins une constante ; la plus simple est certainement celle qui décrit l'arbre vide, noté VID ; de plus, elle permet par composition avec CONS de décrire tous les arbres binaires. Sans paramètre à résultat un arbre, c'est une opération 0-aire ou nullaire, on notera son profil

() → Arb

De même on introduit $\nu + 1$ opérations nullaires (où ν est le cardinal de Alph) de profil :

() → Val

qui correspondent aux éléments de V et à un élément indéfini noté INDEF. On a l'équation

(4) TET (VID) = INDEF .

Tout arbre peut être représenté à partir des opérations CONS, GAU, DRO, TET, VID et des éléments de la famille Alph U {INDEF} . Il est intéressant d'étudier les algèbres qui ne contiennent que ces éléments, c'est à dire les algèbres qui vérifient les équations (1), (2), (3) et (4) et qui ne contiennent que des éléments construits à l'aide des opérations de base CONS, GAU, DRO, TET, VID et la famille Alph U {INDEF} . Une telle algèbre s'appelle une *algèbre première* *. Notons qu'une telle algèbre peut contenir une infinité d'éléments tous différents de la forme suivante GAU(VID), DRO(VID), GAU(GAU(VID)), GAU(DRO(VID)) etc... En informatique

* Cette appellation vient du fait que $\mathbb{Z}/p\mathbb{Z}$ est un anneau unitaire sans sans-anneau unitaire propre si et seulement si p est premier.

cela n'a pas de sens de considérer ces éléments comme différents, deux solutions sont possibles, créer un nouvel élément ERR : () → Arb qui représente un arbre erroné résultat d'une erreur dans l'application de GAU ou DRO, on pose alors

GAU (VID) = GAU (ERR) = DRO (VID) = DRO (ERR) = ERR

ainsi que les axiomes affirmant que le résultat d'une opération sur un uple contenant ERR est ERR. Une autre solution qui évite d'introduire ERR mais qui ne sert pas à distinguer les arbres résultant d'opérations erronées est de poser :

(5) GAU (VID) = VID = DRO (VID)

Par souci de simplicité, adoptons la deuxième solution.

Notation :

si $\mathcal{A}$ est une algèbre $Arb_{\mathcal{A}}$ et $Val_{\mathcal{A}}$ notent les ensembles de la sorte Arb et Val dans l'algèbre $\mathcal{A}$.

On remarque alors que :

Proposition 1 :

Soit $\mathcal{A}$ une algèbre première

$x \in Arb_{\mathcal{A}}$ et $x \neq VID$ implique $\exists y \in Arb_{\mathcal{A}}$ $\exists z \in Arb_{\mathcal{A}}$

$\exists v \in Val_{\mathcal{A}}$ et $x = CONS\ (y, v, z)$

Démonstration :

On raisonne par induction sur la complexité des représentations possibles de x , puisque tous les éléments de Arb admettent une représentation à partir des éléments de base

1er cas :

$x = GAU\ (x')$ si $x' = VID$ alors $x = VID$ et le résultat est immédiat, sinon par induction $x' = CONS\ (y', v', z')$ puisqu'assurément x' admet une représentation plus simple et donc

$x = GAU\ (CONS\ (y', v', z')) = y'$ où y' a une description plus simple que x

et toujours par induction

$y' = CONS\ (y'', v'', z'') = x$

2ème cas :

x = DRO (x) se démontre comme le 1er cas .

3ème cas :

x = CONS (y, v, z) n'entraîne aucune preuve □

Guttag [GUT 76] appelle cela un lemme de forme normale (normal form lemma) (voir le paragraphe 1.1 du chapitre 2).
De cela on déduit

Proposition 2 :

x ≠ VID implique CONS (GAU(x), TET(x), DRO(x)) = x

Démonstration :

D'après le résultat obtenu plus haut
x = CONS (y, v, z) donc

CONS (GAU(x), TET(x), DRO(x)) = CONS (GAU(CONS(y, v, z) TET (CONS(y,v,
DRO(CONS(y, v, z)))

en appliquant les premières équations aux trois paramètres de CONS qui se trouvent les plus extérieurs, on obtient :

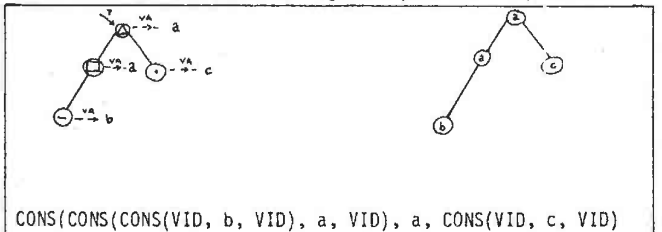
= CONS (y, TET(CONS(y, v, z)), DRO(CONS(y, v, z)))
= CONS (y, v, DRO(CONS(y, v, z)))
= CONS (y, v, z) = x □

On montrerait aisément que toutes les algèbres qui vérifient (1), (2), (3), (4), (5) , sont isomorphes, autrement dit chacune d'elles définit les arbres binaires. L'une d'elles est par exemple constituée des classes de termes sous variables modulo ces cinq équations.

2.2.- ALGÈBRES. "UN ARBRE BINAIRE".

Si l'on regarde intuitivement un arbre, on s'aperçoit qu'il y a des noeuds, que de chacun de ces noeuds, on peut atteindre un noeud, soit à gauche, soit à droite, soit obtenir une valeur ; par conséquent, on peut concevoir un arbre binaire comme une algèbre finie avec un ensemble Noe de noeuds, un ensemble Val de valeurs (que nous supposerons égal à Alph U {INDEF} , des opérations G, D : (Noe) → Noe, VA :(Noe) → Val, T : → Noe. Ce qui nous intéresse, ce n'est pas une algèbre, mais une famille d'algèbres qui vérifie certaines propriétés.
Dans ces algèbres, les opérations G et D ne sont pas partout définies, on dit qu'on a affaire à des *algèbres partielles*. Ces algèbres sont les *algèbres filles* du type abstrait les algèbres binaires.

Ainsi l'arbre binaire de la figure 1 peut être représenté :



CONS(CONS(CONS(VID, b, VID), a, VID), a, CONS(VID, c, VID)

Figure 1.

par l'algèbre finie où Noe = { , , , } , Val = {a, b, c} et où les opérations partielles sont définies par

	G	D	VA
◉	⊖	⊙	a
⊖			b
⊙			c

T = ◉

Voici les propriétés que doivent vérifier ces algèbres

a) un noeud est à gauche d'un autre noeud au plus ou à droite d'un autre noeud au plus ; pour affirmer cette propriété , on a besoin d'un autre ensemble Bool et d'une autre opération totale EG : (Noe, Noe) → Bool ainsi que VRAI, FAUX : () → Bool .

$$(6)\quad \begin{cases} EG\ (x, y) \sqsubseteq EG\ (y, x) \\ EG\ (G(x), G(y)) \sqsubseteq EG\ (x,y) \\ EG\ (D(x), D(y)) \sqsubseteq EG\ (x,y) \\ EG\ (D(x), G(y)) \sqsubseteq FAUX \\ EG\ (D(x), T) \sqsubseteq FAUX \\ EG\ (G(x), T) \sqsubseteq FAUX \\ EG\ (T, T) \sqsubseteq VRAI \end{cases}$$

Ici on utilise le signe $\sqsubseteq$ qui signifie que le membre de gauche est moins défini que le membre de droite ou encore que si le membre de gauche est défini alors le membre de droite l'est et les deux membres sont égaux.

Ainsi l'algèbre telle que Noe = { *, Δ, □ } et telle que G, D et VA sont définis par

	G	D	VA
*	Δ	□	a
Δ		□	b
□			c

et T = * , n'est pas candidate à représenter un arbre (voir figure 2)

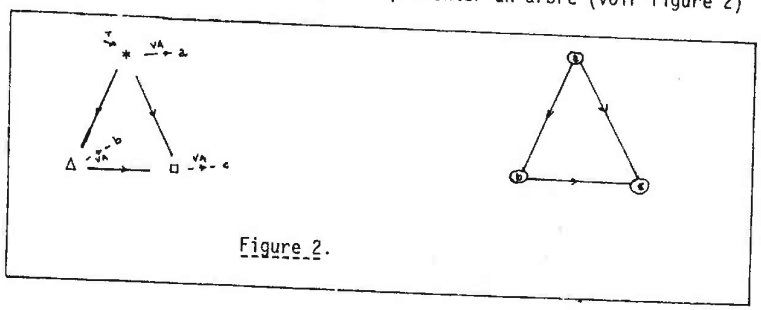


Figure 2.

En effet EG ne vérifie pas la relation

$$EG\ (D(x), D(y)) \sqsubseteq EG(x, y)$$

puisque pour x = * et y = Δ D(x) = D(y) = □

b) Chaque noeud de l'algèbre est atteint, cela s'exprime en disant que les opérations composées, de la forme : () → Noe et qui sont définies, représentent tous les noeuds. En termes d'algèbres, cela signifie que l'algèbre est engendrée par Ø (cf [PIE 68] p. 105) ou, ce qui est équivalent, qu'elle ne contient pas de sous-algèbre propre comme précédemment ; nous appellerons de telles algèbres, des *algèbres premières*.
Ainsi l'algèbre telle que Noe = {θ, θ, θ, 0, Ø }

	G	D	VA
θ	θ		a
θ			b
θ	θ	Ø	a
θ		Ø	b
Ø		θ	c

T = θ

EG	θ	θ	θ	0	Ø
θ	VRAI	FAUX	FAUX	FAUX	FAUX
θ	FAUX	VRAI	FAUX	FAUX	FAUX
θ	FAUX	FAUX	VRAI	VRAI	VRAI
0	FAUX	FAUX	VRAI	VRAI	VRAI
0	FAUX	FAUX	VRAI	VRAI	VRAI

vérifie les équations et inéquations de a) et pourtant ce n'est pas un arbre (voir figure 3)

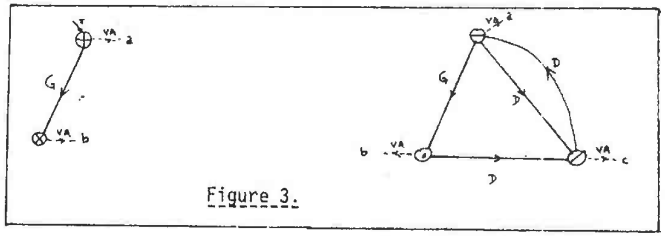


Figure 3.

c) Les opérations VA et EG qui fournissent des valeurs externes sont totales.

d) Pour chaque algèbre $\mathcal{A}$ les ensembles $Val_{\mathcal{A}}$ et $Bool_{\mathcal{A}}$ sont les mêmes ensembles Alph et Bool .

Nous noterons $\underline{\underline{AA}}$ [Alph, Bool] la classe d'algèbres vérifiant les propriétés a) b) c) et d).

Considérons sur $\underline{\underline{AA}}$ [Alph, Bool] diverses constructions :

- $\mathcal{C}ons$: $\underline{\underline{AA}}$ [Alph, Bool] × Alph × AA [Alph, Bool] → AA [Alph, Bool] est définie ainsi :

Etant données deux algèbres filles (ou arbres binaires) et et $v \in V$, $\mathcal{C}ons(\mathcal{A}, v, \mathcal{B})$ est l'algèbre $\mathcal{C}$ telle que

$C = * \cup g \times A \cup d \times B$, où $A = Noe_{\mathcal{A}}$, $B = Noe_{\mathcal{B}}$, $C = Noe_{\mathcal{C}}$

$T = *$

$G_{\mathcal{C}}(*) = (g, T_{\mathcal{A}})$,

$D_{\mathcal{C}}(*) = (d, T_{\mathcal{B}})$,

$G_{\mathcal{C}}((g, x)) = (g, G_{\mathcal{A}}(x))$, $D_{\mathcal{C}}((g, x)) = (g, D_{\mathcal{A}}(x))$

$G_{\mathcal{C}}((d, x)) = (d, G_{\mathcal{B}}(x))$, $D_{\mathcal{C}}((d, x)) = (d, D_{\mathcal{B}}(x))$

et que

$EG_{\mathcal{C}}(*, *) = VRAI$

$EG_{\mathcal{C}}((g, x), (g, y)) = EG_{\mathcal{A}}(x, y)$

$EG_{\mathcal{C}}((d, x), (d, y)) = EG_{\mathcal{B}}(x, y)$

$EG_{\mathcal{C}}(*, (f, x)) = EG_{\mathcal{C}}((f, x), *) = FAUX$ où $f = g$ ou d

$EG_{\mathcal{C}}((g, x), (d, y)) = EG_{\mathcal{C}}((d, x'), (g, y')) = FAUX$

et que

$VA_{\mathcal{C}}(*) = v$

$VA_{\mathcal{C}}(g, x) = VA_{\mathcal{A}}(x)$

$VA_{\mathcal{C}}(d, x) = VA_{\mathcal{B}}(x)$

Il reste à prouver que :

Lemme 1 : $\mathcal{C}ons(\mathcal{A}, v, \mathcal{B})$ est une algèbre de $\underline{\underline{AA}}$ [Alph, Bool]

Eléments de démonstration :

a) on vérifie par cas que $EG_{\mathcal{C}}$ satisfait les inéquations de ( ) par exemple

$EG_{\mathcal{C}}(G(x), G(y)) \sqsubseteq EG_{\mathcal{C}}(x, y)$ dans le cas où $x = *$ et $y = (g, y')$

$EG_{\mathcal{C}}(G_{\mathcal{C}}(*), G_{\mathcal{C}}(g, y')) = EG_{\mathcal{C}}((g, T_{\mathcal{A}}), (g, G_{\mathcal{A}}(y'))$

$= EG_{\mathcal{A}}(T_{\mathcal{A}}, G_{\mathcal{A}}(y'))$ FAUX $= EG_{\mathcal{C}}(*, (g, y'))$

b) $\mathcal{C}$ est une algèbre première en effet chaque point de A est représenté par une opération composée $p_{\mathcal{A}}(T_{\mathcal{A}}) :\rightarrow Noe$, par conséquent, tous les points de la forme (g, x) sont atteints par des opérations composées

$(g, p_{\mathcal{A}}(T_{\mathcal{A}})) = p_{\mathcal{C}}(g, T_{\mathcal{A}}) = p_{\mathcal{C}}(G_{\mathcal{C}}(T_{\mathcal{C}}))$

où $p_{\mathcal{C}}$ est le correspondant dans $\mathcal{C}$ de $p_{\mathcal{A}}$.

On raisonne de même pour les points de la forme (d, x) ; quant à * il n'y a rien à prouver puisqu'il est représenté par $T_{\mathcal{C}}$

c) les opérations $VA_{\mathcal{C}}$ et $EG_{\mathcal{C}}$ est trivialement totale

d) V et Bool sont conservés par cette opération. □

- $\mathcal{G}au$: $\underline{\underline{AA}}$ [Alph, Bool] → $\underline{\underline{AA}}$ [Alph, Bool] est définie ainsi : soit $\mathcal{A}$ un arbre binaire, considérons l'algèbre $\mathcal{B}$ telle que

$Noe_{\mathcal{B}} = Noe_{\mathcal{A}}$, $T_{\mathcal{B}} = G_{\mathcal{A}}(T_{\mathcal{A}})$, $G_{\mathcal{B}} = G_{\mathcal{A}}$, $D_{\mathcal{B}} = D_{\mathcal{A}}$, $EG_{\mathcal{B}} = EG_{\mathcal{A}}$, $VA_{\mathcal{B}} = VA_{\mathcal{A}}$

$\mathcal{G}au(\mathcal{A})$ est la plus petite sous-algèbre de $\mathcal{B}$, c'est à dire la sous-algèbre engendrée par $\emptyset$.

Lemme 2 :

$\mathcal{G}au(\mathcal{A})$ est une algèbre de $\underline{\underline{AA}}$ [Alph, Bool] .

Démonstration :

b), c) et d) sont évidents; pour a) il suffit de prouver les équations faisant intervenir T :

$EG_{\mathfrak{B}}(D_{\mathfrak{B}}(x), T_{\mathfrak{B}}) = EG_{\mathfrak{A}}(D_{\mathfrak{A}}(x), G_{\mathfrak{A}}(T_{\mathfrak{A}})) = FAUX$

$EG_{\mathfrak{B}}(G_{\mathfrak{B}}(x), T_{\mathfrak{B}}) = EG_{\mathfrak{A}}(G_{\mathfrak{A}}(x), G_{\mathfrak{A}}(T_{\mathfrak{A}})) = EG_{\mathfrak{A}}(x, T_{\mathfrak{A}})$

puisque $x \in B$ et que $\mathfrak{B}$ est une algèbre première, x est de la forme $x = p_{\mathfrak{B}}$ ou $p_{\mathfrak{B}}$ est une expression sous variables de $\mathfrak{B}$, donc il existe $y \in A$ tel que $x = G_{\mathfrak{A}}(y)$ ou $x = D_{\mathfrak{A}}(y)$

donc $EG_{\mathfrak{A}}(x, T_{\mathfrak{A}}) = EG_{\mathfrak{A}}(G_{\mathfrak{A}}(y), T_{\mathfrak{A}}) = FAUX$ ou

$EG_{\mathfrak{A}}(x, T_{\mathfrak{A}}) = EG\ (D_{\mathfrak{A}}(y), T_{\mathfrak{A}}) = FAUX$

enfin

$EG_{\mathfrak{B}}(T_{\mathfrak{B}}, T_{\mathfrak{B}}) = EG_{\mathfrak{A}}(G_{\mathfrak{A}}(T_{\mathfrak{A}}), G_{\mathfrak{A}}(T_{\mathfrak{A}})) = EG_{\mathfrak{A}}(T_{\mathfrak{A}}, T_{\mathfrak{A}})$

$= VRAI$ □

- Dro : AA [Alph, Bool] → AA [Alph, Bool] est définie de la même manière que Gau, mais à la différence que l'on pose

$T_{\mathfrak{B}} = D_{\mathfrak{A}}(T_{\mathfrak{A}})$

Enfin nous proposons une sélection

- Tet : AA [Alph, Bool] → Alph définie par

$Tet(\mathfrak{A}) = VA_{\mathfrak{A}}(T_{\mathfrak{A}})$

Dans AA [Alph, Bool] les *morphismes* de $\mathfrak{A}$ vers $\mathfrak{B}$ sont des applications $h : (Noe_{\mathfrak{A}}) \rightarrow Noe_{\mathfrak{A}}$ qui vérifient :

si $T_{\mathfrak{A}}$ est définie alors $T_{\mathfrak{B}}$ l'est et

$h(T_{\mathfrak{A}}) = T_{\mathfrak{B}}$

si $G_{\mathfrak{A}}(x)$ est définie il en est de même de $G_{\mathfrak{B}}(h(x))$ et

$h(G_{\mathfrak{A}}(x)) = G_{\mathfrak{B}}(h(x))$

et si $D_{\mathfrak{A}}(x)$ est définie, il en est de même de $D_{\mathfrak{B}}(h(x))$ et

$h(D_{\mathfrak{A}}(x)) = G_{\mathfrak{B}}(h(x))$

et $VA_{\mathfrak{A}}(x) = VA_{\mathfrak{B}}(h(x))$

$EG_{\mathfrak{A}}(x, y) = EG_{\mathfrak{B}}(h(x), h(y))$

si h est bijective, on dit qu'il s'agit d'un *isomorphisme*.
Puisque les algèbres sont des algèbres partielles, il existe une unique algèbre, arbre binaire, de support Noe vide que nous noterons Vid, c'est celle où T n'est pas défini. Elle est évidemment première.
De manière naturelle, elle représente l'arbre vide. Remarquons que si $\mathfrak{A}$ est une algèbre de AA [Alph, Bool] différente de Vid, $T_{\mathfrak{A}}$ est défini, sinon $\mathfrak{A}$ admettrait une sous-algèbre propre isomorphe à Vid.

Intéressons-nous à une classe d'algèbres de AA [Alph, Bool] telles que Noe est fini et inclus dans un ensemble infini dénombrable donné D. Si l'on veut que $Cons$, Gau et Dro soient stables dans cette classe d'algèbres, il faut que $* \in D$ et que si $A \subseteq D$ alors $\{g\} \times A \subseteq D$ et $\{d\} \times A \subseteq D$. Nous choisissons par conséquent pour D le plus petit sous-ensemble vérifiant

$D \supseteq \{*\} \cup \{g, d\} \times D$

L'isomorphisme $\simeq$ c'est à dire l'existence d'un isomorphisme est une relation d'équivalence dans l'ensemble des algèbres de support D. On montre facilement que $Cons$, Gau et Dro conservent l'isomorphie, que la valeur de Tet ne change pas si l'on prend des algèbres isomorphes et que Vid n'admet pas d'algèbres isomorphes autres qu'elle-même. Posons AAF [Alph, Bool] l'ensemble des classes d'isomorphie d'algèbres à noeud dans D. Posons $Cons'$, Gau', Dro', Tet' et Vid' les opérations déduites de $Cons$, Gau, Dro, Tet et Vid sur AAF [Alph, Bcol]. On a le théorème suivant :

Théorème de la représentation algèbrique des arbres binaires :

L'algèbre de ARB [Alph] où Arb = AAF [Alph, Bool] et qui est munie des opérations $Cons'$, Gau', Dro', Tet' et Vid' est une algèbre de ARB [Alph] ; c'est de plus l'algèbre initiale de ARB [Alph].

Démonstration :

Montrons d'abord que :

$\mathcal{C} = \mathcal{G}_{au}(\mathcal{C}_{ons}(\mathcal{A}, v, \mathcal{B})) \simeq \mathcal{A}$

Considérons l'application bijective $h : A \to \{g\} \times A$ telle que $h(a) = g \times a$; c'est un morphisme de $\mathcal{A}$ vers $\mathcal{C}$: en effet posons

$\mathcal{D} = \mathcal{C}_{ons}(\mathcal{A}, v, \mathcal{B})$

- $h(T_{\mathcal{A}}) = T_{\mathcal{C}}$

car

$T_{\mathcal{C}} = G_{\mathcal{D}}(T_{\mathcal{D}}) = G_{\mathcal{D}}(*) = (g, T_{\mathcal{A}}) = h(T_{\mathcal{A}})$

- $h(G_{\mathcal{A}}(x)) = G_{\mathcal{C}}(h(x))$

car

$G_{\mathcal{C}}(h(x)) = G_{\mathcal{C}}((g, x)) = G_{\mathcal{D}}(g, x)) = (g, G_{\mathcal{A}}(x)) = h(G_{\mathcal{A}}(x))$

La démonstration de l'égalité $h(D_{\mathcal{A}}(x)) = D_{\mathcal{C}}(h(x))$ est identique à la précédente ; celles de $VA_{\mathcal{A}}(x) = VA_{\mathcal{C}}(h(x))$ et $EG_{\mathcal{A}}(x,y) = EG_{\mathcal{C}}(h(x), h(y))$ sont immédiates et sur le même modèle. On montre de même que

$\mathcal{D}_{ro}(\mathcal{C}_{ons}(\mathcal{A}, v, \mathcal{B})) \simeq \mathcal{B}$

On a de plus

$\mathcal{T}_{et}(\mathcal{C}_{ons}(\mathcal{A}, v, \mathcal{B})) = v$

en effet

$\mathcal{T}_{et}(\mathcal{C}_{ons}(\mathcal{A}, v, \mathcal{B})) = VA_{\mathcal{D}}(T_{\mathcal{D}}) = VA_{\mathcal{D}}(*) = v$

Quant à démontrer que $\mathcal{G}_{au}(\mathcal{V}_{id}) = \mathcal{V}_{id}$, il suffit d'examiner de près la définition de $\mathcal{G}_{au}(\mathcal{V}_{id})$. Considérons l'algèbre $\mathcal{B}$ telle que $Noe_{\mathcal{B}} = Noe_{\mathcal{V}_{id}} = \emptyset$, $T_{\mathcal{B}} = G_{\mathcal{V}_{id}}(T_{\mathcal{V}_{id}})$ n'est donc pas défini, il en est de même de $G_{\mathcal{B}} = G_{\mathcal{V}_{id}}$, $D_{\mathcal{B}} = D_{\mathcal{V}_{id}}$, $EG_{\mathcal{B}} = EG_{\mathcal{V}_{id}}$, $VA_{\mathcal{B}} = VA_{\mathcal{V}_{id}}$ ainsi $\mathcal{G}_{au}(\mathcal{V}_{id})$ qui est la plus petite sous-algèbre de $\mathcal{B}$ est $\mathcal{V}_{id}$ elle-même ainsi $\mathcal{G}_{au}(\mathcal{V}_{id}) = \mathcal{V}_{id}$.

On montre de même que $\mathcal{D}_{ro}(\mathcal{V}_{id}) = \mathcal{V}_{id}$.

Pour démontrer qu'il s'agit d'une algèbre initiale, on s'appuie sur un lemme.

Lemme 3 :

Si t est un terme sans variable, construit à l'aide des opérations de la variété ARB , il existe un terme t' sans variable construit seulement à l'aide des opérations CONS et VID tel que $t = t'$.

Démonstration du lemme :

On raisonne par récurrence sur le nombre d'occurrence de GAU et DRO dans t . Supposons dans t , qu'il existe des occurrences de GAU et DRO , il en existe au moins une , notée f , qui apparaît sous l'une des formes :

f(VID) ou f(CONS(g, v, h))

S'il s'agit de la première forme, le terme t" où l'on a substitué VID à f(VID) dans t , contient moins d'occurrence de GAU ou DRO et vérifie $t" = t$.

S'il s'agit de la seconde forme, le terme t", où l'on a substitué g à f(CONS(g, v, h)) , si f = GAU , et h à f(CONS(g, v, h)) si f = DRO , dans t , contient moins d'occurrences de GAU ou DRO et vérifie $t" = t$ □ .

Suite de la démonstration du théorème :

Chaque petite algèbre de AAF [Alph, Bool] n'a qu'un nombre fini de points dans Noe , d'autre part, si $\mathcal{A}$ n'est pas $\mathcal{V}_{id}$, $\mathcal{G}_{au}(\mathcal{A})$ et $\mathcal{D}_{ro}(\mathcal{A})$ définissent des algèbres ayant moins de points (dans Noe) avec la propriété que :

$\mathcal{A} = \mathcal{C}_{ons}(\mathcal{G}_{au}(\mathcal{A}), v, \mathcal{D}_{ro}(\mathcal{A}))$.

Ainsi on peut montrer, par récurrence sur le nombre de points de Noe que $\mathcal{A}$ se décompose de manière unique, à un isomorphisme près en $\mathcal{C}_{ons}$ et $\mathcal{V}_{id}$. Ce résultat et le lemme contribuent à démontrer que l'algèbre est initiale. La proposition 3 du paragraphe 2 du chapitre 2 permettra de prouver plus simplement l'initialité de AAF [Alph, Bool] □ .

2.3.- LIENS AVEC LA PROGRAMMATION.

Le théorème que l'on vient de montrer peut se traduire ainsi : "L'ensemble AAF [Alph, Bool] est une représentation du type arbre binaire par des objets et des opérations de plus bas niveau d'abstraction". Ces objets sont plus concrets, car construits, en particulier, à l'aide de pointeurs. Cette modélisation est une étape vers la programmation car les algèbres filles cad les objets de AAF [Alph, Bool] constituent une bonne approche des objets représentés dans un ordinateur, par un langage de programmation. Les propriétés a) b) c) et d) sont des invariants des différentes constructions. Par exemple, la propriété :"l'agèbre est première" est, pour le programmeur, l'affirmation naturelle suivante :
"à tout moment, un arbre contient seulement les points qui peuvent être atteints depuis la tête".

Traduction dans un langage

a titre d'illustration, nous allons montrer comment les objets de AAF [Alph, Bool] se traduisent en ALGOL 68. Les sortes sont des modes, nous supposerons que le mode Alph est connu

mode Alph ....

Le mode Noe doit fournir le moyen d'accéder par VA à une valeur et par G et D à deux autres noeuds, il est par conséquent assez naturellement traduit par :

mode Noe = struct (rep Noe G, Alph VA, rep Noe D) .

Un arbre comme nous l'avons vu est la donnée d'un ensemble de noeuds et de valeur et d'un certain nombre de fonctions d'accès : G et D fournissent un noeud, à partir d'un autre noeud par un pointeur ; VA est aussi défini en chaque noeud, par conséquent T est le seul accès qu'il faut fournir quand on donne un arbre et ainsi un arbre est décrit de façon naturelle par le seul accès T cela donne alors

(7) *mode Arb = struct (rep Noe T)*

Cette définition peut paraître sophistiquée à un expert programmeur (nous discuterons plus loin de la définition d'un arbre par *mode Arb = rep Noe* et de la suppression de toutes les expressions "T de" . Nous garderons d'abord la définition (7) pour bien montrer au lecteur comment cela s'harmonise avec les algèbres que nous avons construites).

Les constructions et les sélections sont décrites par des procédures. Ainsi Gau est décrite par

```
proc gau = (Arb a) Arb  : (G de T de a)  ;
```

Dro est décrite par

```
proc dro = (Arb a) Arb  : (D de T de a) ;
```

Tet est décrite par

```
proc tet = (Arb a) Alph    : VA de T de a  ;
```

Vid est décrite par une procédure sans paramètres

```
proc vid = Arb  : (nil )  ;
```

Cons est décrite par

```
proc cons = (Arb  a, Alph   v, Arb b) arb   :
     début tas Noe* : = (T de a, v, T de b) ; (étoile) fin
```

Nous allons maintenant donner la traduction des mêmes objets et des mêmes procédures dans un langage qui diffère de Pascal par la possibilité d'avoir des fonctions à résultat de tous les types décrits par le langage :

```
type     alph = ... ; noe = record G : ↑ noe ; VA : alph ; D : ↑ noe  end ;
         arb = record  T : ↑ noe   end  ;
function gau(a : arb) : arb ; begin gau. T : = a. T ↑. G  end  ;
function dro(a : arb) : arb ; begin dro. T : = a. T ↑. D  end  ;
function tet(a : arb) : alph ;begin  tet : = a. T ↑. VA   end  ;
function cons(a : arb ; v : alph ; b : arb) : arb ;
     begin var étoile : ↑ noe ; new (étoile) ;
          étoile ↑.G : = a. T ; étoile ↑.VA : = v ; étoile ↑.D : = b.T
          cons.T : = étoile
     end
```

Si maintenant dans le programme en Algol 68, on remplace la déclaration *mode* Arb = *struct* (*rep Noe* T) par *mode Arb = rep Noe* et si l'on supprime les expressions "T *de* " on obtient :

```
mode Alph = ... ; mode Noe = struct (rep Noe  G ; Alph  VA ; rep Noe
mode Arb = rep Noe ;
proc gau = (Arb a ) Arb : G de a  ;
proc dro = (Arb a ) Arb : D de a  ;
proc tet = (Arb a ) Alph :  VA  de  a  ;
proc vid = Arb : nil    ;
proc cons = (Arb  a ; Alph  v ; Arb b)  Arb :
     tas Noe :=(a, v, b)  .
```

En examinant bien les déclarations des modes *Noe* et *Arb* que l'on rencontre dans le programme précédent :

```
mode Noe = struct (rep Noe  G ; Alph  VA ; rep Noe  D)
mode Arb = rep Noe ;
```

On s'aperçoit que l'on peut les remplacer, sans rien changer ou presque au programme, par l'unique déclaration

```
mode Arb = rep struct (Arb G  , Alph  VA , Arb  D)
```

seul cons deviendra

```
proc cons = (Arb a, Alph v, Arb b) Arb :
     tas struct (Arb  G ; Alph  VA ; Arb D ) : = (a, v, b)
```

Discussion :

Poser *mode* Arb = *rep struct* (*Arb* G , *Alph* VA , *Arb* D) c'est définir le domaine Arb par l'équation à point fixe

Arb = {VID} + (Arb × Alph × Arb)

C'est une définition extensionnelle "à la Scott" (cf [LEH 77]) où l'aspect algébrique a disparu. Une étude tendant à rapprocher les aspects extensionnels et algébriques serait, semble-t-il, particulièrement fructueuse.

Cependant la définition du type arbre par

mode Arb = rep struct (*Arb* G , *Alph* VA, *Arb* D) présente deux dangers (se référer notamment à l'article de J. H. Morris "Types are not set" [MOR 73])

- le plus connu est certainement l'utilisation sur les objets de ce mode d'opérations non prévues lors de la spécification du type, avec tous les risques que cela comporte dans l'exécution du programme ; cette exécution a toutes chances d'être erronée. Des affectations du genre

  G *de* z : = z

  engendrent des objets monstrueux qui ne sont pas des arbres, du moins pas des arbres finis ; en effet l'objet α représenté par z vérifie :

  G *de* α = α

  Remarquons que des langages comme CLU, ALPHARD et ATM protègent le programmeur contre de tels risques ([LIS 74][WUL 76] [CHA 79] [MIN 79]

- le second danger est la confusion de deux concepts très différents celui de noeud d'un arbre et celui d'arbre lui-même. La figure 4 voudrait illustrer la différence qu'il y a entre ces deux concepts. Tout d'abord signalons qu'un noeud est un objet interne d'un arbre qui doit être caché à l'utilisateur d'arbres. Ce principe d'*information cachée* est pour de nombreux spécialistes l'un des garants d'une construction de logiciels efficaces, car il évite de multiples erreurs dues à la modification de valeurs auxquelles l'utilisateur ne devrait pas avoir accès. Cette protection pourrait être assurée en Algol 68, à condition d'interdire à l'utilisateur de connaître le mode *Noe*. Il faudrait pour cela repousser la déclaration *mode Arb = struct* (*rep Noe* T) dans une partie implantation qui pourrait être le prologue. Ainsi la distinction entre arbre et noeud,

et les distinctions, qui s'en déduisent, entre GAU et G , DRO et D TET et T ne permettent pas par exemple l'affectation G *de* z : = z mentionnée, puisqu'on ne peut accéder aux noeuds qu'à travers les procédures au, dro et tet .

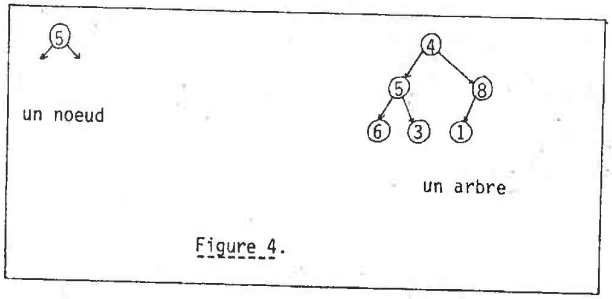


Figure 4.

Signalons qu'à un noeud est attachée une valeur exactement, tandis qu'à un arbre est attachée une famille de valeurs dont l'une peut être privilégiée, la tête.

Avantages d'une approche algébrique.

Remarquons que lors de l'implantation d'un type abstrait, dans un langage de programmation, il est indispensable de *prouver* que les procédures proposées représentent bien les opérations désirées. Le passage par le modèle des algèbres filles est un outil commode pour faire cette preuve.

Enfin nous terminerons, en précisant que l'exemple des arbres binaires a été choisi pour sa simplicité et le fait qu'il contient des opérations dyadiques, dans le but d'illustrer les concepts d'algèbres de base et d'algèbres filles. Mais cet exemple n'est plus très convaincant quand on passe à la représentation en Algol 68, car ce langage contient, avec les structures, des notions très proches de celles que l'on trouve dans les arbres. Les exemples du paragraphe 6 et du chapitre 4 fourniront des arguments plus convaincants.

2.4.- ARBRES BINAIRES ET PARTAGES.

Dans les arbres binaires que nous avons rencontrés aucun partage de descendance n'était possible. Dans la représentation qui suit nous accepterons une forme très restreinte de partages : un même noeud peut être à la fois le fils gauche et le fils droit de son père (figure 5). Autrement dit, on remplace

$$EG(D(x), G(y)) \sqsubseteq FAUX$$

par

$$EG(D(x), G(y)) \Rightarrow EG(x, y) \sqsubseteq VRAI$$

$\Rightarrow$ est une opération sur les booléens, bien connue.

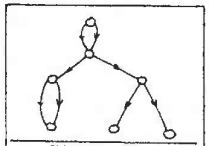


Figure 5.

Sur ces algèbres, les opérations gau, dro, tet et Vid ne sont pas modifiées. Si $\mathcal{A}$ et $\mathcal{B}$ ne sont pas isomorphes, on définit $Cons(\mathcal{A}, a, \mathcal{B})$ comme avant, mais si les algèbres sont isomorphes on définit $\mathcal{C} = Cons(\mathcal{A}, a, \mathcal{B})$ ainsi

$C = \{*\} \cup A$

$T_{\mathcal{C}} = *$

$G_{\mathcal{C}}(*) = D_{\mathcal{C}}(*) = T_{\mathcal{A}}$

si $x \neq *$ alors $G_{\mathcal{C}}(x) = G_{\mathcal{A}}(x)$ et $D_{\mathcal{C}}(*) = D_{\mathcal{A}}(x)$

$EG_{\mathcal{C}}(*, *) = VRAI$

si $x \neq *$ $EG_{\mathcal{C}}(*, x) = EG_{\mathcal{C}}(x, *) = FAUX$

si $x \neq *$ et $y \neq *$ $EG_{\mathcal{C}}(x, y) = EG_{\mathcal{A}}(x, y)$

$VA_{\mathcal{C}}(*) = a$

si $x \neq *$ $VA_{\mathcal{C}}(x) = VA_{\mathcal{A}}(x)$

Il est facile de prouver que les algèbres, ainsi construites, vérifient les axiomes de la figure 6 :

EG(x, y) ⊑ EG(g, x)
EG(G(x), G(y)) ⊑ EG(x,y)
EG(D(x), D(y)) ⊑ EG(x,y)
EG(D(x), G(y)) → EG(x, y) ⊑ VRAI
EG(D(x), T) ⊑ FAUX
EG(G(x), T) ⊑ FAUX
EG(TT) ⊑ VRAI

Figure 6 :

Axiomes des arbres binaires avec partages.

Soit $Arb_{\mathcal{F}}$ l'ensemble dont les éléments sont les classes d'isomorphie d'algèbres qui vérifient les axiomes de la figure 6. Sur $Arb_{\mathcal{F}}$ et Alph on définit les opérations "Vid", "Cet", "Gau", "Dro", "Cons" par passage au quotient des opérations correspondantes Vid, Cet, Gau, Dro, Cons. On a ainsi défini une algèbre de ARB [Alph] qui est une algèbre initiale de cette classe.

3.- RAPPELS SUR LES ALGÈBRES HOMOGÈNES.

Ce paragraphe et le suivant peuvent être sautés en première lecture ; ils serviront essentiellement de référence, dans la suite, pour les concepts et les notations. Notons cependant trois notions assez fondamentales, exposées ici : les algèbres premières (§ 3.4 et § 4.3.), les identités dans les algèbres (§ 3.6 et §4.6), les algèbres terminales (§ 4.4). La présentation des algèbres homogènes avant celle des algèbres hétérogènes est simplement didactique. Le paragraphe 5 introduit un aspect mathématique nouveau encore incomplètement approfondi : les algèbres homogènes paramétrées.

3.1.- ALGÈBRES ET ALGÈBRES PARTIELLES.

Une *algèbre homogène* sur un ensemble F de symboles d'arités données est un couple $\mathcal{A} = < A ; F >$ où A est un ensemble et $F_{\mathcal{A}}$ une famille d'opérations sur A indexée par F, c'est à dire d'applications $f_{\mathcal{A}}$ telles que $f_{\mathcal{A}} : A^n \to A$ si n est l'arité de f. [BIR 35] .

Notations :

Une algèbre est notée par une majuscule gothique, son support par la majuscule latine correspondante (voir l'annexe 1 pour la correspondance entre les gothiques et les latines). L'opération associée à f est notée par $f_{\mathcal{A}}$ ou encore f simplement si l'on n'y a pas d'ambiguïté sur l'algèbre considérée.

Une algèbre où $F_{\mathcal{A}}$ est l'image d'une famille F de symboles d'arité donnée est appelée une F-algèbre [GOG 77] , ou un F-magma [NIV 75] ; on notera Alg (F) la classe de ces algèbres ; si deux algèbres appartiennent à la même classe elles sont dites *similaires* ou de même type.

Une *algèbre partielle* (homogène) est un couple $\mathcal{A} = < A ; F >$ où A est un ensemble de $F_{\mathcal{A}}$ une famille d'opérations partielles sur A (i.e. qui ne sont pas forcément partout définies). Dans le cas où f est une opération nullaire ou d'arité 0 et où $\mathcal{A}$ est une algèbre (non partielle, nous dirons aussi totale dans ce cas pour préciser) $f_{\mathcal{A}}$ est une constante. Si $\mathcal{A}$ est une algèbre partielle, $f_{\mathcal{A}}$ est aussi une constante, mais cette constante peut ne pas être définie. On notera Alg p(F) la classe des F-algèbres partielles.

Soit $\mathcal{A} = < A ; F >$ une algèbre partielle, soit $B \subset A$; B est une partie F-*stable* de A si pour tout $(a_1, \ldots, a_n) \in B^n$, $f_{\mathcal{A}}(a_1,\ldots,a_n)$

défini implique $f\ (a_1, \ldots, a_n) \in B$, on définit alors la sous-algèbre $\mathfrak{B} = < B ; F_{\mathfrak{B}} >$ induite par B en posant $f_{\mathfrak{B}}(a_1, \ldots, a_n) = f_{\mathfrak{A}}(a_1, \ldots,$ En particulier, si $\mathfrak{A}$ est une algèbre totale, cela signifie que $\mathfrak{B}$ est une algèbre totale similaire et que $\mathfrak{B}$ est une sous-algèbre si et seulement si les opérations de $F_{\mathfrak{A}}$ sont stables dans B, $F_{\mathfrak{B}}$ est constituée des restrictions à B de ces opérations. Si $\mathfrak{A}$ est une algèbre partielle qui contient des opérations d'arité 0 , alors B contient tous les éléments ou constantes que ces opérations désignent.

Soient $\mathfrak{A}$ et $\mathfrak{B}$ deux algèbres partielles similaires, une application $\varphi : A \longrightarrow B$ est un *morphisme* de $\mathfrak{A}$ vers $\mathfrak{B}$ si pour tout f d'arité n et tous $a_1, \ldots, a_n$ de A , $f_{\mathfrak{A}}(a_1, \ldots, a_n)$ défini implique que $f_{\mathfrak{B}}(\varphi(a_1), \ldots, \varphi(a_n))$ est défini et que

$$\varphi(f_{\mathfrak{A}}(a_1, \ldots, a_n)) = f_{\mathfrak{B}}(\varphi(a_1), \ldots, \varphi(a_n))$$

Si φ est une application surjective, φ est appelé un *épimorphisme*. Si φ est une application bijective , φ est appelé un *isomorphisme* ; si il existe un isomorphisme de $\mathfrak{A}$ vers $\mathfrak{B}$ alors $\mathfrak{A}$ et $\mathfrak{B}$ sont dits isomorphes ce que l'on note $\mathfrak{A} \simeq \mathfrak{B}$

Exemples d'algèbres :

1) Une *algèbre monadique* est une algèbre dont toutes les opérations sont unaires ou monadiques, c'est à dire d'arité 1 ; une sous-algèbre $\mathfrak{B}$ est telle que si $b \in B$, $f(b) \in B$. L'algèbre de support V^* l'ensemble des mots sur V est munie d'opérations $\alpha \longrightarrow a.\alpha$ pour chaque $a \in V$ est une algèbre monadique $< V^* ; V >$; les algèbres de support l'ensemble des mots de longueur au plus n et telle que $a(\alpha)$ est définie par $a.\alpha$ si et seulement si la longueur de α est inférieure à n , sont des algèbres monadiques partielles.

2) une *algèbre dyadique* est une algèbre dont toutes les opérations sont binaires ou dyadiques c'est à dire d'arité 2.

3) un *corps* est une algèbre partielle $\mathfrak{K} = < K ; 0, 1, -, {}^{-1}, +, \times >$ où les opérations $0, 1, -, +, \times$ sont totales d'arité $0, 0, 1, 2, 2$ tandis que $^{-1}$ est une opération d'arité 1 définie sur $K - \{0\}$. □

3.2.- CONSTRUCTIONS D'ALGÈBRES.

Le *produit* des algèbres $(\mathfrak{A}_i / i \in I)$ a pour support $\underset{i \in I}{\pi} A_i$ et les opérations y sont définies composante par composante, plus précisément si f est une opération et si $\mathfrak{A} = \underset{i \in I}{\pi} \mathfrak{A}_i$, pour $a_j = (a_j^i)_{i \in I} \in A$

$$f_{\mathfrak{A}}(a_1, \ldots, a_n) = (f_{\mathfrak{A}_i}(a_1^i, \ldots, a_n^i))_{i \in I}$$

Une relation d'aquivalence Γ sur A est une *congruence* sur $\mathfrak{A}$ si, pour chaque opération f d'arité n

$(\forall i\ [1, n]\ a_i\ \Gamma\ a'_i) \wedge f(a_1, \ldots, a_n)$ défini $\wedge f(a_1, \ldots, a'_n)$ défini $\Rightarrow$

$f(a_1, \ldots, a_n)\ \Gamma\ f(a'_1, \ldots, a'_n)$

Les classes d'équivalence $\bar{\Gamma}(a)$ forment le support d'une algèbre $\mathfrak{B} = \mathfrak{A}_{/\Gamma}$ où les opérations sont définies par

$$f_{\mathfrak{B}}(\bar{\Gamma}(a_1), \ldots, \bar{\Gamma}(a_n)) = \bar{\Gamma}(f_{\mathfrak{A}}(a_1, \ldots, a_n))$$

3.3.- ALGÈBRES LIBRES, ALGÈBRES INITIALES.

Une *algèbre partielle libre* sur X dans une classe $\underline{K}$ d'algèbres similaires est une algèbre $\mathfrak{A}$ telle que $X \subseteq A$ et que pour toute algèbre $\mathfrak{B}$ de $\underline{K}$, pour toute application h de X vers B , il existe un unique morphisme $\bar{h}$ de $\mathfrak{A}$ vers $\mathfrak{B}$ prolongeant h. Les résultats suivants sont

connus :

- pour tout X donné et pour toute classe $\underline{\underline{K}}$ d'algèbres partielles similaires, si $\mathcal{A}$ et $\mathcal{A}'$ sont deux algèbres libres sur X, alors $\mathcal{A}$ et $\mathcal{A}'$ sont isomorphes. Compte tenu de ce résultat nous noterons $\mathcal{P}(K, X)$ l'une de ces algèbres libres, elle est définie à un isomorphisme près, nous l'appellerons parfois l'algèbre libre sur X. (P est pris pour polynomes).

- si $\underline{\underline{K}}$ est une classe d'algèbres partielles similaires, stable par produit et par passage aux sous-algèbres et contenant une algèbre avec plus d'un élément alors $\underline{\underline{K}}$ admet pour tout X une algèbre libre sur X :

- en particulier $\underline{\underline{Alg}}(F)$ admet des algèbres libres pour tout X. Par exemple l'une d'elle est $\mathcal{P}(F, X)$ où $P(F, X)$ est formé des F-mots sur X ou F-termes à variables dans X, c'est à dire que $P(F, X)$ est le plus petit ensemble contenant X et stable par composition par les opérations de F. Nivat note cette F-algèbre (appelée F magma) $M(F, X)$ [NIV 75] et ADJ (Goguen, Thatcher, Wagner et Wright) la note $T_F(X)$ [GOG 77].

- si $X = \emptyset$, l'algèbre $\mathcal{P}(\underline{\underline{K}}, X)$ s'appelle algèbre initiale sur $\underline{\underline{K}}$; nous noterons parfois $\mathcal{P}(\underline{\underline{K}})$ au lieu de $\mathcal{P}(\underline{\underline{K}}, \emptyset)$; pour toute algèbre de $\underline{\underline{K}}$ il existe un unique morphisme $h_{\mathcal{A}}^{\underline{\underline{K}}}$ de $\mathcal{P}(K, \emptyset)$ vers $\mathcal{A}$ (notée souvent $h_{\mathcal{A}}$ s'il n'y a pas d'ambiguité sur $\underline{\underline{K}}$). Dans le cas où $\underline{\underline{K}} = \underline{\underline{Alg}}(F)$ Nivat note $M(F)$ le magma initial et ADJ note T_F, l'algèbre initiale.

3.4.- ALGÈBRES ENGENDRÉES ET ALGÈBRES PREMIÈRES.

Soit $\mathcal{A}$ une algèbre partielle, soit $\mathcal{S}(\mathcal{A})$ l'ensemble de tous les sous-ensembles de A qui déterminent une sous-algèbre de $\mathcal{A}$. $\mathcal{S}(\mathcal{A})$ est stable par intersection.

Soit $X \subset A$, la *sous algèbre engendrée* par X est la plus petite sous-algèbre dont le support contient X, son support est noté $[X]$. La sous-algèbre engendrée par $\emptyset$, notée $\mathbb{K}(\mathcal{A})$ est la plus petite sous-algèbre de $\mathcal{A}$, son support est $\emptyset$. Une algèbre partielle $\mathcal{A}$ est *première* si elle ne contient aucune sous-algèbre propre, autrement dit si $\mathcal{A} = \mathbb{K}(\mathcal{A})$ ou encore $\mathcal{S}(\mathcal{A}) = \{A\}$.

Proposition 3 :

L'image d'une algèbre première par un épimorphisme est une algèbre première.

Démonstration :

Soit $\varphi : \mathcal{A} \to \mathcal{B}$ un épimorphisme. Supposons que $\mathcal{B}$ n'est pas première mais que $\mathcal{A}$ l'est. Soit alors $C \in \mathcal{S}(\mathcal{A})$, $C \neq B$ alors $\varphi^{-1}(C) \neq A$ et $\varphi^{-1}(C) \in \mathcal{S}(\mathcal{A})$ ce qui est contradictoire □

Corollaire :

Si $\mathcal{A}$ est une algèbre première, si Θ est une congruence sur $\mathcal{A}$, l'algèbre $\mathcal{A}/\Theta$ est première. □

Remarque :

Les algèbres premières de même type forment une classe stable par passage aux sous-algèbres (si l'on peut dire, puisqu'il s'agit de l'identité) et par passage aux images épimorphes, mais elle n'est pas stable par

passage aux produits. En effet si p est premier, l'algèbre $< \mathbb{Z}/p\mathbb{Z} ; 0, 1, +, . >$ est une algèbre première ; l'algèbre $< \mathbb{Z}/p\mathbb{Z}\ 0, 1, +, . > \times < \mathbb{Z}/p\mathbb{Z} ; 0, 1, +, . >$ n'est pas une algèbre première puisqu'elle contient la sous algèbre propre de support $\mathbb{Z}/p\mathbb{Z} \times \{1\}$. □

Proposition 4 :

Si $\mathfrak{A}$ est une algèbre libre sur X dans une classe $\underline{\underline{K}}$ elle est engendrée par X . □

Corollaire :

Si $\mathfrak{A}$ est une algèbre initiale dans une classe $\underline{\underline{K}}$, elle est première. □

Proposition 5 :

Si $\mathfrak{A}$ et $\mathfrak{B}$ sont deux algèbres partielles premières, il y a au plus un morphisme de $\mathfrak{A}$ vers $\mathfrak{B}$; ce morphisme est surjectif. □

3.5.- FONCTIONS POLYNOMIALES.

Dans ce paragraphe, on donne l'interprétation des polynomes de degré n par des fonctions à n variables. Si $\mathfrak{A}$ est une algèbre partielle on peut munir l'ensemble $F^{(n)}(\mathfrak{A})$ des fonctions partielles de A^n vers A d'une structure d'algèbre en posant pour $p_1, \ldots, p_m$ dans $F^{(n)}(\mathfrak{A})$ et f d'arité m , $a = (a_1, \ldots, a_m)$.

Si $p_1(a), \ldots, p_m(a)$ sont défini il en est de même de $f(p_1, \ldots, p_m)(a)$ et $f(p_1, \ldots, p_m)(a) = f(p_1(a), \ldots, p_m(a))$

Parmi ces fonctions, il en existe n , notées $proj_1, \ldots, proj_n$ appelées les projections et définies par

$$proj_i(a_1, \ldots, a_n) = a_i$$

La sous-algèbre $\mathcal{P}^{(n)}(\mathfrak{A})$ engendrée par $\{proj_1, \ldots, proj_n\}$ est appelée l'algèbre des fonctions polynomiales à n variables sur $\mathfrak{A}$. Si $\mathfrak{A}$ est une algèbre totale, ces fonctions polynomiales sont des fonctions totales. Soit $X_n = \{x_1, \ldots, x_n\}$. Soit $\underline{\underline{K}}$ une classe contenant $\mathfrak{A}$ et $\mathcal{P}^{(n)}(\mathfrak{A})$, et une algèbre libre $\mathcal{P}(\underline{\underline{K}}, X_n)$; Remarquons que si $\mathfrak{A}$ est une algèbre totale il suffit que la classe K contenant $\mathfrak{A}$ soit stable par passage aux produits et aux sous-algèbres). Il existe un unique morphisme de $\mathcal{P}(\underline{\underline{K}}, X_n)$ vers $\mathcal{P}^{(n)}(\ )$ prolongeant l'application qui envoie x_i sur $proj_i$; on note $w_{\mathfrak{A}}$ l'image d'un élément w de $\mathcal{P}(K, X_n)$ ($w_{\mathfrak{A}}$ est indépendant de l'indice n)

3.6.- IDENTITÉS SUR LES ALGÈBRES.

Soit $X_\omega = \{x_1, \ldots, x_n, \ldots\}$ un ensemble infini dénombrable de variables indexées par les entiers. Une famille d'*identités* (resp d'*inégalités*) est une famille de couples d'éléments de $P(F, X_\omega)$ que l'on note $v = w$ (resp $v \sqsubseteq w$), si n est l'indice maximum des variables figurant dans $\{v,w\}$ on peut supposer que v $P(F, X_n)$ et w $P(F, X_n)$. Dans une famille d'inégalités une paire $\{v \sqsubseteq w, w \sqsubseteq v\}$ est une identité.

Une algèbre $\mathfrak{A}$ est un modèle d'une famille $\underline{\underline{E}}$ d'identités si pour tout $v = w$ de $\underline{\underline{E}}$ on a $v_{\mathfrak{A}} = w_{\mathfrak{A}}$. Une algèbre partielle $\mathfrak{A}$ est un *modèle* d'une famille $\underline{\underline{E}}$ d'inaglités si pour tout $v \sqsubseteq w$ de $\underline{\underline{E}}$ on a $v_{\mathfrak{A}} \sqsubseteq_{\mathfrak{A}} w_{\mathfrak{A}}$ où $\sqsubseteq_{\mathfrak{A}}$ représente l'ordre moins défini que sur $F^{(n)}(\mathfrak{A})$.

Si elle n'a pas un nom spécifique la classe constituée de toutes les algèbres qui sont des modèles d'une famille E d'identités est notée $\underline{\underline{Alg}}\ (F ; \underline{\underline{E}})$, celle des algèbres partielles qui sont modèles d'une famille d'inégalités, $\underline{\underline{E}}$ est notée $\underline{\underline{Alg\ p}}\ (F, \underline{\underline{E}})$.

Exemple 1 :

Soit $F = \{0, .\}$ où l'arité de 0 est 0 et l'arité de $.$ est 2. La classe des monoïdes est la classe Alg $(0, . ; 0.x_1 = x_1 , x_1.0 = x_1, x_1.(x_2.x_3) = (x_1.x_2).x_3)$ on la notera plus simplement MON □

Exemple 2 :

Soit $F = \{0, +, -, ., 1, ^{-1}\}$ où 0 et 1 sont d'arité 0 et $^{-1}$ d'arité 1 et $+$ et $.$ d'arité. La classe des corps est contenue dans celles des algèbres partielles qui sont modèles des inégalités :

$$0 + x_1 = x_1 \quad , \quad x_1 + - x_1 = 0 \quad , \quad x_1 + x_2 = x_2 + x_1$$

$$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3 \quad , \quad 1.x_1 = x_1.1 \quad , \quad x_1.1 = x_1$$

$$x_1 \cdot x_1^{-1} \sqsubseteq 1 \quad , \quad x_1^{-1} \cdot x_1 \sqsubseteq 1 \quad ,$$

$$x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$$

$$(x_1 + x_2) \cdot x_3 = x_1 \cdot x_3 \cdot x_3$$

(La condition supplémentaire pour être en corps est :
" $0, +, -, ., 1$ sont totales et $^{-1}$ a pour domaine de définition $A - \{0\}$)

4.- ALGÈBRES HÉTÉROGÈNES

4.1.- ALGÈBRES HÉTÉROGÈNES ET ALGÈBRES HÉTÉROGÈNES PARTIELLES

Une algèbre hétérogène est un couple $\mathfrak{A} = < A ; F >$ où

- $A = (s_{\mathfrak{A}})_{s \in S}$
- les $s_{\mathfrak{A}}$ sont des ensembles, les *sortes* de $\mathfrak{A}$ (S est l'ensemble des descripteurs de sortes).
- F est un ensemble d'opérations hétérogènes.

Nous allons préciser ce que nous entendons par opération hétérogène.

Le *profil* du symbole d'opération hétérogène f est un couple $(\sigma, t) \in S^* \times S$ noté

$$f : (s1 , \ldots, sn) \rightarrow t$$

ou $f : () \rightarrow t$ si σ est le mot vide S^*, c'est à dire si $n = 0$.

n est l'*arité* de f . L'opération hétérogène $f_{\mathfrak{A}}$ sur A qui correspond à f est une application de $s1 \times \ldots \times sn$ vers t . Si f est une fonction partielle, on dit qu'il s'agit d'une opération partielle.

Notations :

On essaie ici d'être cohérent avec les notations utilisées précédemment. L'algèbre est notée par une majuscule gothique, ici $\mathfrak{A}$; son *support*, c'est à dire l'ensemble $\{s_{\mathfrak{A}} \mid s \in S\}$ est noté par la majuscule latine correspondante, ici A , l'ensemble ayant pour sorte s est noté $s_{\mathfrak{A}}$. ADJ utilise la notation A_s , plus cohérente avec la pratique mathématique, mais moins avec la pratique informatique. Chaque opération de symbole f de l'algèbre $\mathfrak{A}$ est noté $f_{\mathfrak{A}}$; s'il n'y a pas d'ambiguité sur l'algèbre les indices $_{\mathfrak{A}}$ sont omis .

Si l'on considère toutes les algèbres ayant même ensemble de descripteurs de sortes et où $F_{\mathfrak{A}}$ est l'image d'une famille unique F de symboles de profils donnés, on notera Alg (F) la classe qu'elles constituent en supposant que l'ensemble S est implicitement connu, on dira qu'il s'agit d'algèbres *similaires*.

Une *algèbre hétérogène partielle* est un couple $\mathfrak{A} = < A ; F >$ où A est un S-uple $(s_{\mathfrak{A}})_{s \in S}$ de sortes et $F_{\mathfrak{A}}$ est une famille d'opérations partielles dont le profil est construit sur $S_{\mathfrak{A}}$.

Soit $\mathfrak{A} = < A ; F >$ une algèbre hétérogène partielle , soit $B \subset A$ c'est à dire que $s_{\mathfrak{B}} \subseteq s_{\mathfrak{A}}$ pour chaque $s \in S$; $\mathfrak{B} = < B ; F_{\mathfrak{B}} >$ est une *sous-algèbre* de $\mathfrak{A}$ si pour tout $(a_1, \ldots, a_n) \in s1_{\mathfrak{B}} \times \ldots \times sn_{\mathfrak{B}}$ et $f_{\mathfrak{A}}$

de profil $s1 \times \ldots \times sn \to s$, $f_{\mathcal{A}}(a_1, \ldots, a_n)$ défini implique que $f_{\mathcal{A}}(a_1, \ldots, a_n) \in s_{\mathcal{A}}$; on pose alors $f_{\mathcal{B}}(a_1, \ldots, a_n) = f_{\mathcal{A}}(a_1, \ldots, a_n)$

Un *morphisme* $\varphi : \mathcal{A} \to \mathcal{B}$ entre deux algèbres hétérogènes partielle similaires de sortes S et de symboles F est un ensemble $\varphi = \{\varphi_s / s \in S\}$ d'applications telles que, pour tout f de profil $(s1, \ldots, sn) \to s$ et tout $(a_1, \ldots, a_n) \in s1 \times \ldots \times sn$, $f_{\mathcal{A}}(a_1, \ldots, a_n)$ défini implique que $f_{\mathcal{B}}(\varphi s1(a_1), \ldots, \varphi sn(a_n))$ est défini et que

$$\varphi\ (f(a_1, \ldots, a_n)) = f_{\mathcal{B}}(\varphi_{s1}(a_1), \ldots, \varphi_{sn}(a_n))$$

Si φ est constitué d'applications surjectives, φ est appelé un *épimorphisme*. Si φ est constitué d'applications bijectives φ est appelé un *isomorphisme*.

4.2.- ALGÈBRES HÉTÉROGÈNES LIBRES, ALGÈBRES HÉTÉROGÈNES INITIALES.

Soit $X = (X_s)_{s \in S}$; une algèbre hétérogène partielle libre sur X dans une classe $\underline{\underline{K}}$ d'algèbres similaires (de sorte S) est une algèbre hétérogène partielle $\mathcal{A}$ telle que pour chaque $s \in S$, $X_s \subset s_{\mathcal{A}}$ et que pour toute algèbre hétérogène $\mathcal{B}$ de $\underline{\underline{K}}$, pour toute famille $(h_s)_{s \in S}$ d'application de X_s vers s il existe un unique morphisme $\overline{h}$ de $\mathcal{A}$ vers $\mathcal{B}$ prolongeant h dans le sens suivant : pour tout $x \in X_s$ $\overline{h}_s(x) = h_s(x)$.
Les résultats suivants sont faciles à prouver, on les trouve en partie dans l'article de Birkhoff et Lipson [BIR 70] .

- Pour tout X donné et toute classe $\underline{\underline{K}}$ d'algèbres hétérogènes partielles, si $\mathcal{A}$ et $\mathcal{A}'$ sont deux algèbres hétérogènes partielles libres sur X

alors $\mathcal{A}$ et $\mathcal{A}'$ sont isomorphes. Nous noterons $\mathcal{F}(\underline{\underline{K}}, X)$ l'algèbre libre hétérogène partielle (définie à un isomorphisme près). $\mathcal{F}(F, X)$ est l'algèbre libre de $\underline{\underline{Alg}}(F)$ sur X , ses éléments sont appelés des F-mots ou plus simplement des mots.

Exemple 3 :

Nous pouvons définir la complexité d'un F-mot en terme d'une unique morphisme vers une algèbre particulière. Considérons l'algèbre hétérogène $\mathcal{C}$ ou $s_{\mathcal{C}} = \mathbb{N}$ et pour chaque f d'arité n et de profil $s_1, \ldots, s_n \to s$ on a :

$$f(x_1, \ldots x_n) = 1 + \sum_{i=1}^{n} x_i$$

Soit $C^S : X_s \to \mathbb{N}$ tel que $C^S(x) = 0$ alors l'unique morphisme de $\mathcal{F}(F, X)$ vers $\mathcal{C}$ prolongement c est noté comp. comp(v) s'appelle la *complexité* de v son utilisation est fondamentale dans les démonstrations par récurrence. □

- si $\underline{\underline{K}}$ est une classe d'algèbres partielles hétérogènes similaires, stables par produit et passage aux sous-algèbres et contenant une algèbre avec plus d'un élément alors $\underline{\underline{K}}$ admet pour tout X une algèbre libre sur X .

. - si $X = \emptyset$ l'algèbre $\mathcal{F}(\underline{\underline{K}}, X)$ s'appelle l'algèbre initiale sur $\underline{\underline{K}}$; pour tout algèbre hétérogène $\mathcal{A}$ de $\underline{\underline{K}}$, il existe un unique morphisme $h^{\underline{\underline{K}}}_{\mathcal{A}}$ de $\mathcal{F}(K, \emptyset)$ vers $\mathcal{A}$. Dans le cas où $\underline{\underline{K}} = \underline{\underline{Alg}}(F)$ ADJ note T_F l'algèbre initiale, nous la noterons $\mathcal{F}(F)$.

4.3.- ALGÈBRES HÉTÉROGÈNES ENGENDRÉES ET ALGÈBRES HÉTÉROGÈNES PREMIÈRES.

Si $\mathcal{A}$ est une algèbre partielle, l'ensemble $\mathcal{S}(\mathcal{A})$ est l'ensemble de tous les S-uples d'ensembles qui déterminent une sous-algèbre hétérogène . $\mathcal{S}(\mathcal{A})$ est stable par intersection. Soit $X \subseteq A$ i.e $X = (X_s)_{s \in S}$ $X_s \subseteq s_{\mathcal{A}}$

la *sous-algèbre hétérogène engendrée par* X est la plus petite sous-algèbre $\mathcal{B}$ dont le support contient X i.e. $X_s \subseteq s_{\mathcal{B}}$, son support est noté $[X] = ([X_s])_{s \in S}$. La plus petite sous-algèbre hétérogène d'une algèbre hétérogène est notée $\mathbb{K}(\mathcal{A})$, elle est engendrée par $(\emptyset)_{s \in S}$. Une algèbre hétérogène est première, si elle ne contient aucune sous-algèbre première propre, autrement dit si $\mathcal{A} = \mathbb{K}(\mathcal{A})$ ou encore $\mathcal{T}(\mathcal{A}) = \{A\}$.
Les propositions du paragraphe 2.4 restent valables.

4.4.- ALGÈBRES TERMINALES.

Dans une classe $\underline{\underline{K}}$ d'algèbres, une algèbre $\mathcal{A}$ est terminale si pour toute algèbre $\mathcal{B}$ de $\underline{\underline{K}}$, il existe un unique morphisme de $\mathcal{B}$ vers $\mathcal{A}$.

Proposition 6 :

Si $\mathcal{A}$ et $\mathcal{B}$ sont deux algèbres terminales elles sont isomorphes.

Démonstration :

Le seul morphisme de $\mathcal{A}$ vers $\mathcal{A}$ est l'identité sur $\mathcal{A}$. Il existe un seul morphisme h de $\mathcal{A}$ vers $\mathcal{B}$ et un seul morphisme k de $\mathcal{B}$ vers $\mathcal{A}$. La composition koh est un morphisme de $\mathcal{A}$ vers $\mathcal{A}$, c'est l'identité ; donc h est un isomorphisme. □

On notera que ce raisonnement est exactement le même que celui que l'on fait pour montrer l'isomorphisme des algèbres initiales.

Dans les variétés, il existe des algèbres triviales, c'est à dire dont les supports sont réduits à un seul élément qui sont notamment produit vide d'algèbres ; ce sont les algèbres terminales car il est bien évident qu'il n'existe qu'un morphisme de toute autre algèbre vers elles.

L'étude des types abstraits amène à s'intéresser à des classes d'algèbres qui ne sont pas des variétés, qui ont des algèbres terminales non triviales et qui jouent un rôle fondamental.

4.5.- FONCTIONS POLYNOMIALES.

Si $\mathcal{A}$ est une algèbre hétérogène partielle, on peut munir d'une structure d'algèbre hétérogène, le S-uple $F^{(n)}(\mathcal{A}) = (F^{(n)}(\mathcal{A})_t)_{t \in S}$, où $n = (n_s)_{s \in S}$ et où $F^{(n)}(\mathcal{A})_t$ est l'ensemble des fonctions partielles de $\prod_{s \in S} s_{\mathcal{A}}^{n_s}$ vers $t_{\mathcal{A}}$. Cela se fait de la manière suivante : posons $p_1 \in F^{(n)}(\mathcal{A})_{s'1}, \ldots, p_m \in F^{(n)}(\mathcal{A})_{s'm}$, f de profil $(s'1, \ldots, s'm) \to s'$, $a = (a_s)_{s \in S}$ où $a_s \in s^{n_s}$; si $p_1(a), \ldots, p_m(a)$ son définis il en est de même de $f(p_1, \ldots, p_m)(a)$ et $f(p_1, \ldots, p_m)(a) = f_{\mathcal{A}}(p_1(a), \ldots, p_n(a))$.
Parmi ces fonctions, il en existe notées $proj_i^s$ $(1 \leq i \leq n_s)$, appelées les projections et définies par

$proj_i^s(a)$ = la i^e composante de a_s

La sous-algèbre $\mathcal{P}^{(n)}(\mathcal{A})$ engendrée par les projections est appelée *l'algèbre des fonctions polynomiales* à $n = (n_s)_{s \in S}$ variables sur $\mathcal{A}$.
Soit $X_n = (S^s_{n_s})_{s \in S}$ où $X^s_{n_s} = x^s_1, \ldots, x^s_{n_s}$. Soit $\underline{\underline{K}}$ une classe contenant $\mathcal{P}^{(n)}(\mathcal{A})$ et une algèbre libre $\mathcal{P}^{(n)}(\underline{\underline{K}}, X_n)$. Considérons l'homomorphisme de $\mathcal{P}^{(n)}(K, X_n)$ vers $\mathcal{P}^{(n)}(\mathcal{A})$ prolongeant les applications qui envoient x^s_i sur $proj^s_i$, on note $w_{\mathcal{A}}$ l'image d'un élément w de $s_{\mathcal{P}(\underline{\underline{K}}, X_n)}$.

4.6.- IDENTITÉS SUR LES ALGÈBRES HÉTÉROGÈNES.

Soit $X_\omega = (X^s_\omega)_{s \in S}$ où $X^s_\omega = \{x^s_1, \ldots, x^s_n, \ldots\}$ des ensembles infinis dénombrables de variables indexées par les descripteurs de sortes et les entiers. Une famille d'*identités* (resp. d'*inégalités*) est une famille de

couples d'éléments de $S_{\mathcal{P}}(F, X_w)$ que l'on note $v = w$ (resp $v \subseteq w$).

Si n est le maximum, pour l'ordre produit, des indices des variables figurant dans $\{v, w\}$ on peut supposer que v et w appartiennent à $S_{\mathcal{P}}(F, X_n)$.

Une algèbre hétérogène $\mathcal{A}$ est un *modèle* d'une famille $\underline{E}$ d'identités si pour tout $v = w$ de $\underline{E}$ on a $v_{\mathcal{A}} \equiv w_{\mathcal{A}}$. Une algèbre partielle hétérogène est un modèle d'une famille $\underline{E}$ d'inégalités si pour tout $v \subseteq w$ de E on a $v_{\mathcal{A}} \subseteq w_{\mathcal{A}}$.

Alg $(F ; E)$ est la classe des algèbres hétérogènes modèles d'une famille E d'identités, on l'appelle aussi une *variété* ; celles des algèbres hétérogènes partielles qui sont modèles d'une famille $\underline{E}$ d'inégalités est notée Algp $(F ; \underline{E})$.

Exemple 4 :

Dans le paragraphe 1.1, nous avons envisagé des algèbres qui sont modèles de la famille d'identités

GAU(CONS(x, v, y)) = x
DRO(CONS(x, v, y)) = y
TET(CONS(x, v, y)) = v
GAU(VID) = VID
DRO(VID) = VID □

Exemple 5 :

Dans le paragraphe 1.2 nous avons envisagé des algèbres partielles qui sont modèles de la famille d'inégalités.

EG(x, y) $\subseteq$ EG(y, x)
EG(G(x), G(y)) $\subseteq$ EG(x,y)
EG(D(x), D(y)) $\subseteq$ EG(x, y)
EG(D(x), G(y)) $\subseteq$ FAUX
EG(D(x), T) $\subseteq$ FAUX
EG(G(x), T) $\subseteq$ FAUX
Eg(T, T) $\subseteq$ VRAI □

5.- ALGÈBRES HOMOGÈNES PARAMÉTRÉES.

Les paragraphes 3 et 4 présentaient les algèbres homogènes et hétérogènes. Les premières n'ont qu'un seul ensemble sous-jacent, les secondes en ont plusieurs, mais aucun de ces ensembles ne joue un rôle prépondérant. Il apparaît à la lumière des exemples que nous avons vu qu'une sorte (parfois plusieurs) constitue le type d'intérêt et domine les autres qui sont des paramètres de la définition. Ainsi dans le type "les arbres binaires" la sorte Arb joue un rôle plus important que la sorte Alph ; de même dans le type : "un arbre binaire" la sorte Noe est la plus importante. Notons que les paramètres n'apparaissent pas sous forme de types, c'est à dire de classes d'algèbres, mais sous forme d'algèbres déjà complètement définies. Les algèbres paramétrées répondent au reproche fait aux algèbres hétérogènes d'être plate et de ne pas faire apparaître la structure de la spécification des objets. Un autre problème résolu par les algèbres paramétrées est le suivant: quand on s'intéresse à une représentation de type abstrait, on suppose que les types paramètres sont déjà représentés (ou le seront mais leur représentation n'intervient pas ici) autrement dit, on ne considère qu'une algèbre donnée pour chaque type et non une classe d'algèbres (voir la proposition 3 du chapitre 2 , paragraphe 2). Ce concept d'algèbres paramétrées recouvre des catégories d'objets mathématiques bien connus dont :

1) les K-espaces vectoriels ou espaces vectoriels sur un corps K, le paramètre est le corps K, si K est le corps $\mathbb{R}$ des réels, il s'agit des espaces vectoriels réels.

2) les groupes à opérateurs ou Ω-groupes.

3) les algèbres monadiques sur Alph (cf. chapitre 4, paragraphe 1.1) elles constituent un exemple assez typique et bien connu d'algèbres paramétrées intervenant dans la théorie des langages.

Soit $\mathcal{A}_1, \ldots, \mathcal{A}_n$ des algèbres homogènes (éventuellement elle-même paramétrées) ; *une algèbre homogène paramétrée par* $\mathcal{A}_1, \ldots, \mathcal{A}_n$ appelée aussi $\mathcal{A}_1 - \mathcal{A}_2 - \ldots - \mathcal{A}_n$ algèbre est un couple $\mathcal{A} = < A_0 ; F >$ où A_0 est un ensemble et où F est un ensemble d'opérations hétérogènes sur $A_0, A_1, \ldots, A_n$ dont le profil contient au moins une occurrence de la dénomination de sorte s_0 correspondant à A_0. Pour des raisons de commodité on supposera que si s_0 apparaît en partie gauche, elle apparaît en tête.

Un *morphisme d'algèbres paramétrées* de $\mathcal{A}$ vers $\mathcal{B}$ est une application h de A_0 vers B_0 telle que

- pour $f : (s_0, \ldots, s_0, s_{i1}, \ldots, s_{im}) \to s_0$ on a

$$h\,(f_{\mathcal{A}}(x_1, \ldots, x_n, y_1, \ldots, y_m) = f_{\mathcal{B}}(h(x_1), \ldots, h(x_n), y_1, \ldots, y_m)$$

- pour $f : (s_0, \ldots, s_0, s_{i1}, \ldots, s_{im}) \to t$ où $t \neq s_0$

$$f_{\mathcal{A}}(x_1, \ldots, x_n, y_1, \ldots, y_m) = f_{\mathcal{B}}(h(x_1), \ldots, h(x_1), y_1, \ldots, y_m)$$

On associe à toute algèbre paramétrée une algèbre hétérogène dont le support est $(A_0, A_1, \ldots, A_n)$ et à tout morphisme h d'algèbre paramétrée un morphisme d'algèbre hétérogène $(h, i_1, \ldots, i_n)$ où i_j est l'identité sur A_j. Les concepts tels que algèbres partielles, morphismes d'algèbres

partielles, algèbres premières, algèbres initiales, algèbres libres, identités sur les algèbres, se transposent sans difficultés aux algèbres paramétrées.

Proposition 7 :

La classe $\underline{\underline{\text{Alg}}}\,(F ; \mathcal{A}_1, \ldots, \mathcal{A}_n ; \underline{E})$ des F-algèbres paramétrées par $\mathcal{A}_1, \ldots, \mathcal{A}_n$ vérifiant une famille $\underline{E}$ d'identités, admet une algèbre libre sur X pour tout ensemble X

Démonstration :

Soit $F' = F \cup F(\mathcal{A}_n) \cup \ldots \cup F(\mathcal{A}_n)$ où $F(\mathcal{A}_i)$ est l'ensemble des opérations de $\mathcal{A}_i$:

Soit $\underline{E}' = \underline{E} \cup \underline{D}(\mathcal{A}_1) \cup \ldots \cup \underline{D}(\mathcal{A}_n)$ où $\underline{D}(\mathcal{A}_i)$ est l'ensemble des identités qui sont vraies dans $\mathcal{A}_i$:

La classe d'algèbres hétérogènes $\underline{\underline{\text{Alg}}}\,(F', \underline{E}')$ admet une algèbre libre, pour tout $\underline{X} = (X, \emptyset, \emptyset, \ldots, \emptyset)$ cette algèbre est paramétrée par $\mathcal{A}_1, \ldots, \mathcal{A}_n$. C'est une algèbre libre de $\underline{\underline{\text{Alg}}}\,(F ; \mathcal{A}_1, \ldots, \mathcal{A}_n)$; en effet si $\mathcal{A}$ est une algèbre quelconque de $\underline{\underline{\text{Alg}}}\,(F ; \mathcal{A}_1, \ldots, \mathcal{A}_n)$ c'est une algèbre de $\underline{\underline{\text{Alg}}}\,(F, \underline{E}')$ donc il existe une morphisme unique de l'algèbre libre vers $\mathcal{A}$ prolongeant X.

6.- DEUX_EXEMPLES : LES_FILES_ET_LES_ENSEMBLES.

Les deux types abstraits que nous allons étudier contribueront avec le type arbres binaires à illustrer le chapitre suivant :

6.1.- LES FILES.

Nous allons considérer les files notées File, sur un alphabet Alph. Les opérations sont :

FILEVIDE : () → File,
AJ : (File, Alph) → File,
OT : (File) → File,
FR : (File) → Alph U {INDEF} ,
CONCAT : (File, File) → File ;

Elles vérifient les identités

(OT1) OT(FILEVIDE) = FILEVIDE,
(OT2) OT(AJ(FILEVIDE, a)) = FILEVIDE ,
(OT3) OT(AJ(AJ(f) a), b)) = AJ(OT(AJ(f, a)), b) ,
(FR1) FR(FILEVIDE) = INDEF ,
(FR2) FR(AJ(FILEVIDE, a)) = a ,
(FR3) FR(AJ(AJ(f, a), b)) = FR (AJ(f, a))
(CONCAT1) CONCAT (f, FILEVIDE) = f
(CONCAT2) CONCAT (f, AJ(f', a)) = AJ(CONCAT (f, f'), a) .

Ce type abstrait a des différences profondes avec le type abstrait arbres binaires, que nous avons étudié au paragraphe 1 ; dans les arbres binaires GAU et DRO jouaient des rôles antagonistes par rapport à CONS . Ici OT n'est pas l'antagoniste de AJ : on n'"ote" pas le dernier élément que l'on a "ajouté" ; la spécification de OT est récursive ainsi le second

membre de la troisième équation contient OT ; on retrouve des propriétés semblables pour la spécification de FR. CONCAT , quant à lui permet de construire une file à partir de deux autres files, mais chaque file peut s'exprimer uniquement grâce à AJ , si bien que dans CONCAT (f, f') on peut faire disparaître totalement l'opération CONCAT, c'est ce que nous appellerons une opération secondaire. Une telle opération n'existe pas dans les arbres binaires.

La classe FILE [Alph] de modèles proposés ici est constitué des algèbres partielles premières finies munies des opérations suivantes :

SU : (Place) → Place,

PREM : () → Place

VA : (Place) → Alph,

EG : (Place, Place) → Bool

et vérifiant les inéquations suivantes :

EG (PREM, PREM) ⊑ VRAI
EG (PREM, S U (x)) ⊑ FAUX
EG (SU(x), PREM) ⊑ FAUX
EG (SU(x), SU(y)) ⊑ EG(x, y)

sur la classe FILE [Alph] on définit les opérations suivantes :

$\mathfrak{Filevide}$ est l'algèbre telle que Place = ∅

$\mathfrak{Fr}(\mathfrak{a}) = VA_{\mathfrak{a}}(PREM_{\mathfrak{a}})$

Les opérations $\mathfrak{at}$, $\mathfrak{aj}$, $\mathfrak{concat}$ sont données par la figure 7.

Notations :

Dans la suite nous utiliserons parfois les conventions d'écriture suivantes qui facilitent la compréhension des formules :

FILEVIDE sera noté ∧
AJ(f, a) sera noté f ∘ a
CONCAT(f, f') sera noté f * f'
INDEF sera noté ?

ainsi

CONCAT(AJ(f, FR(FILEVIDE, a)), OT(AJ(AJ(FILEVIDE, a), b)))

sera noté

[f ⊙ FR (∧ ⊙ a)] * OT(∧⊙a⊙b)

	Nouvelle valeur de Place	PREM	SU		VA		EG	
			général	exception	général	exception	général	exception
$\mathcal{C}_1(\mathfrak{L})$	$Place_{\mathfrak{L}} - \{PREM_{\mathfrak{L}}\}$	$SU_{\alpha}(PREM_{\mathfrak{L}})$	SU_{α}		VA_{α}		EG_{α}	
$\mathcal{A}_{\mathfrak{f}}(\mathfrak{L},a)$	$Place_{\mathfrak{L}}$ U α où $\alpha \notin Place_{\mathfrak{L}}$ par exemple $\alpha = Place_{\mathfrak{L}}$	$PREM_{\mathfrak{L}}$	$S\ U_{\mathfrak{L}}$	si S U (x) est indéfini alors SU(x) =	$VA_{\mathfrak{L}}$	$VA(\alpha) = a$	$EG_{\mathfrak{L}}$	$EG(\alpha,\alpha)$=VRAI si $x \neq \alpha$ $EG(\alpha, x)$ $= EG(x,\alpha)$ = FAUX
$\mathcal{C}_{oncat}(\mathfrak{L},\mathfrak{L}')$	$Place_{\mathfrak{L}} \bullet Place_{\mathfrak{L}'}$	$PREM_{\mathfrak{L}}$	$SU_{\mathfrak{L}}$ ou $SU_{\mathfrak{L}'}$	si $SU_{\mathfrak{L}}(x)$ est indéfini $SU(x) = PREM_{\mathfrak{L}'}$	$VA_{\mathfrak{L}}$ ou $VA_{\mathfrak{L}'}$		$EG_{\mathfrak{L}}$ ou $EG_{\mathfrak{L}'}$	si $x \in Place$ et $y \in Place$ $EG(x, y) =$ $EG(y, x)$ FAUX

Figure 7 : Définition des constructions $\mathcal{C}_t$, $\mathcal{A}_f$, $\mathcal{C}_{oncat}$

6.2.- LES ENSEMBLES.

Pour les ensembles nous proposons ici la description suivante :

VIDE : () → Ens ,
AJ : (Ens, Alph) → Ens,
PR : (Ens, Alph) → Bool ;

Elles vérifient les identités

PR(AJ(e, a), b) = SI EG (A, b) ALORS VRAI SINON PR(a, b)

PR(VIDE, a) = FAUX

Trois familles de modèles vont être présentées.

La première est constituée d'algèbres munies essentiellement d'une opération totale $\in$: (Alph) → Bool. On remarque que dans ces modèles aucune sorte, autrement dit aucune partie de l'objet, n'est cachée de l'extérieur. On n'a ajouté aucune nouvelle catégorie d'objet et il n'y a ici rien qui tiennent le lieu de la sorte Noe dans les arbres ou de la sorte Place dans les files. On définit simplement une opération.

. $\mathcal{V}ide$ est définie ainsi $\in_{Vide} \equiv$ FAUX

. $\mathcal{O}bj(\mathcal{A}, A)$ est définie ainsi : $A \in_{\mathcal{O}bj(\mathcal{A}, A)} = VRAI$

si $B \neq A$ $B \in_{\mathcal{O}bj(\mathcal{A}, A)} = B \in_{\mathcal{A}}$

. $\mathcal{R}(\mathcal{A}, A) = A \in_{\mathcal{A}}$

La seconde est constitué des mêmes objets que les files , mais

. $\mathcal{V}ide$ est l'algèbre telle que Place = $\emptyset$

. $\mathcal{O}bj(\mathcal{A}, a)$ est l'algèbre $\mathcal{B}$ telle que $Place_{\mathcal{B}} = Place_{\mathcal{A}} \cup \{Place_{\mathcal{A}}\}$

notons encore α le nouvel élément

$PREM_{\mathcal{B}} = \alpha$

si $x \in Place_{\mathcal{A}}$ alors $SU_{\mathcal{B}}(x) = SU_{\mathcal{A}}(x)$ et $VA_{\mathcal{B}}(x) = VA_{\mathcal{A}}(x)$

sinon $SU_{\mathcal{B}}(x) = PREM_{\mathcal{A}}$ et $VA_{\mathcal{B}}(x) = a$

si $x, y \in Place_{\mathcal{A}}$ alors $EQ_{\mathcal{B}}(x, y) = EQ_{\mathcal{A}}(x, y)$

si $x \in Place_{\mathcal{A}}$ alors $EQ_{\mathcal{B}}(x, \alpha) = FAUX = EQ_{\mathcal{B}}(\alpha, x)$

enfin $EQ_{\mathcal{B}}(\alpha, \alpha) = VRAI$

. $\mathcal{R}(\mathcal{A}, a) = (\exists x \in Place_{\mathcal{A}})\ (VA_{\mathcal{A}}(x) = a)$

La troisième est moins orthodoxe, elle suppose que Alph contient un élement distingué que nous noterons 0 si Alph était l'ensemble des entiers, ce serait zéro par exemple. Dans ces modèles on définit une sorte Place ; il y a les trois opérations du modèle précédent et une opération OE : () → Bool qui note si 0 appartient ou non à l'ensemble :

. $\mathcal{V}ide$ est l'algèbre telle que Place = $\emptyset$ et OE = FAUX

. Pour définir $\mathcal{O}bj(\mathcal{A}, a)$ deux cas sont à envisager si $a \neq 0$ alors $\mathcal{O}bj(\mathcal{A}, a)$ est défini comme dans la représentation précédente et

$OE_{\mathcal{O}bj(\mathcal{A}, a)} = OE_{\mathcal{A}}$.

$\mathcal{O}bj(\mathcal{A}, 0)$ est l'algèbre $\mathcal{B}$ telle que $Place_{\mathcal{B}} = Place_{\mathcal{A}}$, toutes les opérations sont les mêmes sur $\mathcal{A}$ et $\mathcal{B}$ sauf éventuellement $OE_{\mathcal{B}}$ qui prend la valeur VRAI .

. si $a \neq 0$ $\mathcal{R}(\mathcal{A}, a) = (\exists x \in Place_{\mathcal{A}})\ VA_{\mathcal{A}}(x) = a$

$\mathcal{R}(\mathcal{A}, 0) = OE_{\mathcal{A}}$

Une quatrième famille représente les ensembles par des "listes sans répétitions". Là encore on a une représentation en liste avec les opérations :

SU : (Place) → Place,
PREM : ( ) → Place,
VA : (Place) → Alph,
EG : (Place, Place) → Bool

et vérifiant les mêmes équations que les files

. Vide est toujours l'algèbre telle que Place = ∅

. $\mathcal{A}_j$ ($\mathcal{A}$, a) dépend de l'existence d'une place x telle que $VA_{\mathcal{A}}(x) = a$

si $(\exists\, x \in \text{Place}\ )\ (VA_{\mathcal{A}}(x) = a)$ alors $\mathcal{A}_j(\mathcal{A}, a) = \mathcal{A}$

sinon $\mathcal{A}_j(\mathcal{A}, a)$ est définie comme dans le deuxième modèle.

. $\mathcal{P}_r(\mathcal{A}, a) = (\exists\, x \in \text{Place}_{\mathcal{A}})\ (VA_{\mathcal{A}}(x) = a)$

sur le thème de la troisième famille de représentation, on pourrait construire d'autres représentations, en particularisant non pas un élément dans Alph mais deux, trois, ... n etc... D'autre part, on pourrait développer d'autres représentations à base d'arbres binaires de recherches [AHO 74] . Nous ne les aborderons pas ici.

Notation :

Dans la suite, nous adopterons la convention d'écriture suivante :

VIDE sera noté ∅

AJ(e, a) sera noté e + a .

CHAPITRE II

étude algébrique de la représentation d'un type abstrait

CHAPITRE II

ETUDE ALGÈBRIQUE DE LA REPRÉSENTATION D'UN TYPE ABSTRAIT.

Ce chapitre comporte essentiellement deux parties :
dans la première nous présentons les divers aspects algébriques des types abstraits et de leurs représentations : réécriture, algèbres de types, représentation par des algèbres.
Dans la deuxième partie nous présentons une représentation canonique d'un type abstrait.

Convention :

Dans ce chapitre, nous conviendrons de noter Ti la sorte associée au type abstrait que l'on défini ou type d'intérêt et Ext_1, ..., Ext_m les types qui interviennent dans la définition ou types paramètres.

1.- TYPES ABSTRAITS ET RÉÉCRITURES.

1.1.- SYSTÈMES DE RÉÉCRITURE [HUE 77] [RAO 78]

On peut considérer les types abstraits comme des *systèmes de réécriture*, chaque identité est orientée, c'est un couple (non pas une paire) $g \longrightarrow d$ appelé règle de réécriture.

Nous présentons ici les concepts de la réécriture assez informellement, en particulier le concept d'occurrence d'un sous terme est considéré sous son aspect intuitif (Huet [HUE 77] donne un traitement plus complet).

Une *substitution* est une famille $\sigma = \{x_1 := t_1, \ldots, x_{ni} = t_n\}$ de couples variables-termes. L'application de la substitution au terme t est le terme σt obtenu en substituant chaque occurrence de x_i par t_i.

Un terme t *se réécrit* en un terme t' si
- il existe un sous terme s de t
- il existe une substitution σ
- il existe une règle $g \to d$

tels que $\sigma g = s$ et t' est le terme obtenu en remplaçant dans t, s par σd.

On écrit alors $t \to t'$. La fermeture transitive et réflexive de $\to$ est notée $\overset{*}{\to}$.

Un terme t est *irréductible* si $t \overset{*}{\to} t'$ implique $t = t'$ autrement dit t ne peut pas se réécrire en un terme.

Un système est *confluent* ou a la *propriété de Church Rosser* si $t \overset{*}{\to} t_1$, $t \overset{*}{\to} t_2$ implique qu'il existe t' tels que $t_1 \overset{*}{\to} t'$ et $t_2 \overset{*}{\to} t'$ (figure 1)

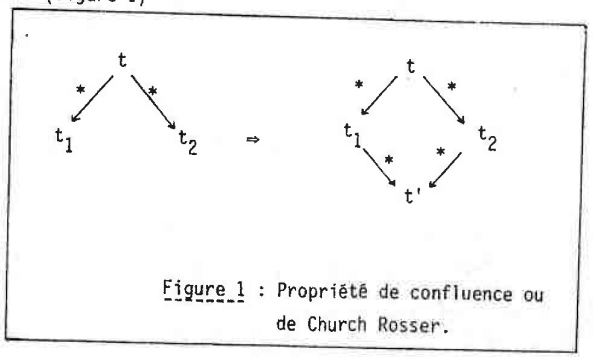


Figure 1 : Propriété de confluence ou de Church Rosser.

Si le système de réécriture est confluent, chaque terme se réécrit en au plus un terme irréductible $\mu[t]$ que l'on appelle sa *forme normale*. La fonction μ est en général une fonction partielle.

Un système de réécriture est noethérien ou a la propriété de *terminaison finie* si pour chaque terme t, il n'existe aucune suite infinie de termes tels que $t_0 \to t_1 \to \ldots \quad t_i \to t_{i+1} \to \ldots$.

Un système de réécriture est *complet*, si il est confluent et noethérien. En cas de confusion avec d'autres concepts de complétude nous dirons aussi confluent-noethérien. Musser [MUS 78] propose le terme *convergent*.

Si un système est complet, à chaque terme, on peut associer une forme normale ; donc la fonction μ est totale.

1.2.- CONFLUENCE DANS LE CAS DES TYPES ABSTRAITS.

L'algorithme de Knuth et Bendix [KNB 70][HUE 77] teste la confluence d'un système de réécriture ; il procède ainsi : g et g' sont supposés avoir des variables distinctes, un terme g est superposable à g' s'il existe un sous-terme s de g' (non réduit à une variable) et une substitution σ tels que $\sigma g = \sigma s$. Le terme $\sigma g'$ est appelé le résultat de la superposition.

Un système de réécriture est *localement confluent* si $t \to t_1$, $t \to t_2$ implique qu'il existe t' tels que $t_1 \overset{*}{\to} t'$ et $t_2 \overset{*}{\to} t'$

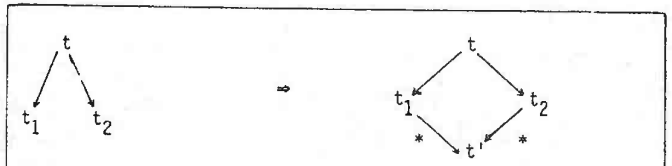


Figure 2 : Propriété de confluence locale.

Knuth et Bendix montrent qu'il suffit pour tester la confluence d'un système noethérien de tester la confluence locale sur les membres gauches $\sigma g'$ de règles qui résultent d'une superposition plus précisément dans le cas où la substitution σ est la "plus générale", appelée plus général unifieur (cf chapitre 6).

D'expérience, nous avons constaté le phénomène suivant :

Dans un type abstrait, les **membres** gauches des règles de réécritures ne sont pas en général superposables.

Voici une explication de ce phénomène . Les opérations de profil $\ldots \rightarrow$ Ti peuvent être partitionnées en deux familles.

- les *générateurs* ce sont ceux qui interviennent dans les formes normales

- les *opérations secondaires*, ce sont celles qui n'interviennent pas dans les formes normales ; certaines tendent à diminuer la longueur des formes normales. Nous les appellerons *destructeurs*. Précisément un destructeur f est de profil $(Ti, Ext_{i_1}, \ldots, Ext_{i_n}) \rightarrow Ti$ et

$$| \mu(f(x, y_1, \ldots, y_n))| < | \mu(x)|$$

pour toute expression x . $|\ |$ est la longueur.

Un *sélecteur* est une opération de profil $\ldots \rightarrow Ext_i$.

La plupart des règles d'un type abstrait ont la forme $H(t_1, \ldots, t_n) \rightarrow$ où $t_1, \ldots, t_n$ sont des termes qui ne sont construit qu'avec des générateurs et H est une opération secondaire. Ainsi s'il y a superposition d'un membre gauche $H'(t'_1, \ldots, t'_n)$ sur $H(t_1, \ldots, t_n)$, elle ne peut pas se faire sur un sous terme strict, car ces sous-termes ne contiennent pas d'opérations secondaires.

Ainsi $H' = H$ et il existe une substitution σ telle que $\sigma t_1 = \sigma t'_1, \ldots, \sigma t_n = \sigma t'_n$. Alors le terme $H(\sigma t_1, \ldots, \sigma t_n)$ se réécrit de deux façons σt et $\sigma t'$. Cette ambiguïté est souvent gênante dans une spécification : c'est pourquoi on l'évite le plus possible, ce qui revient à éviter tout cas de superposition . On pourrait même songer à un système automatique qui détecte les superpositions, les signale au spécifieur et éventuellement lui dit s'il y a confluence locale.

Exemple 1 :

Reprenons l'exemple des File du paragraphe 5.1 supposons qu'un spécifieur écrive les règles.

(CONCAT1) CONCAT1(f, FILEVIDE) = f

puis

(CONCAT2') CONCAT (f, AJ(f', a)) =
= CONCAT(AJ(f, FR(AJ(f', a))), OT(AJ(f', a)))

puis découvre une autre règle

(CONCAT2) CONCAT(f, AJ(f', a)) = AJ(CONCAT(f, f'), a)

La deuxième et la troisième règles se superposent de façon triviale, la règle (CONCAT2') un peu compliquée peut être supprimée. Une raison apparaîtra dans l'exemple 2.
Notons en passant, que ces deux règles ne se ramènent pas si simplement l'une à l'autre, puisque la preuve de l'équivalence entre les deux règles ne se fait pas par simple réécriture, mais nécessite une récurrence. □

1.3.- TERMINAISON DANS LE CAS DES TYPES ABSTRAITS.

La terminaison finie, qui est délicate à prouver dans les systèmes algébriques les plus généraux, est souvent facile à tester dans les types abstraits "bien construits" ; pour cela on utilise un critère proposé par Musser [MUS 78] .

Définition 1 :

Soient F et H deux opérations, une règle $H(t_1, \ldots, t_n) \rightarrow t$ *réduit* F s'il existe $i \in [1...n]$ et des termes $s_1, \ldots, s_p$ tels que $t_i = F(s_1, \ldots, s_p)$ et toute occurrence de H dans t à la forme $H(\ _1, \ldots, \ _n)$ où $\ _i$ est l'un des s_j pour un $j \in [1..p]$ (figure 3).

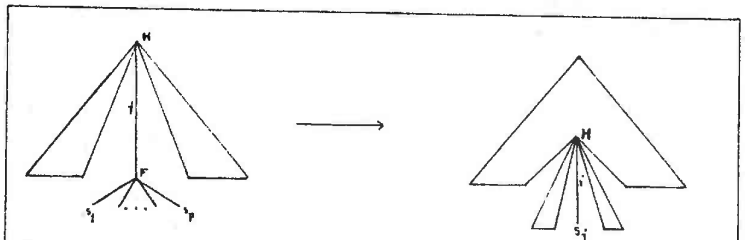


Figure 3 : schéma de réduction de F

Remarque 1 :

si les règles obéissent aux conditions énoncés aux alinéas précédents F est un constructeur, puisqu'il a une occurrence à l'intérieur du terme de gauche.

Remarque 2 :

si t ne contient aucune occurrence de H alors de façon évidente la règle réduit F .

Exemple 2 : la règle

CONCAT(f, AJ(f', a)) → AJ(CONCAT(f, f'), a)

réduit AJ avec i = 2 et j = 1

La règle

OT(AJ(AJ(f, a), b)) → AJ(OT(AJ(f, a)), b)

réduit AJ avec i = 1 et j = 1

La règle

CONCAT(f, AJ(f', a)) → CONCAT(AJ(f, FR(AJ(f', a))), OT(AJ(f', a)))

ne réduit pas AJ □

Exemple 3 :

Soit F une opération, la propriété d'associativité

F(a, F(b, c)) → F(F(a, b), c)

réduit F avec i = 2 et j = 1, 2 , dans le membre droit F apparaît dans F(a, b) et F(..., c) □

Théorème (Musser) :

Si un système S a la propriété de terminaison fine, si aucune règle de S ne contient d'occurrence de H , si R a la forme $H(t_1, \ldots, t_n) \rightarrow t$ et si R réduit F pour un F au moins, alors S U {R} a la propriété de terminaison finie.

Principe de la démonstration :

Sans nuire à la généralité on peut supposer que la règle R réduit F et a la forme $H(F(s_1, \ldots, s_p), t_2, \ldots, t_n)$. Supposons qu'il existe une chaîne infinie de réécriture dans S U {R} .
Cette chaîne utilise une infinité de fois la règle R .
Posons $\vec{x}$ pour $\vec{R} \circ \vec{S}^*$ et intéressons-nous à cette relation.

Par le lèmme de König, on peut déduire de cette chaîne, une chaîne infinie où seuls ne sont réécrits (par $\vec{x}$) , que des termes dont un des sous termes sera utilisé dans la réécriture $\vec{x}$ suivante.

Considérons la suite des radicaux c'est à dire des sous termes de la forme $R_k = H(F(\ldots, u_{i(k)}, \ldots), \ldots)$ qui vont intervenir dans une réécriture . Si $u_{i(k)}$ est le terme tel que $R'_{k+1} = H(u_{i(<)}, \ldots)$ dans la réécriture de R_k et tel que R'_{k+1} produira R_{k+1} . On remarque que

$u_{i(k)} \overset{*}{\underset{S}{\rightarrow}} F(\ldots, u_{i(k+1)}, \ldots)$ et qu'ainsi

$F(\ldots, u_{i(k)}, \ldots) \overset{*}{\underset{S}{\rightarrow}} F(\ldots, F(\ldots, u_{i(k+1)}, \ldots), \ldots)$.

On déduit alors une chaîne infinie dans S , d'où la contradiction □

Ainsi , pour qu'un système de réécriture associé à un type abstrait, ait la propriété de terminaison finie, il suffit qu'il soit construit par adjonctions successives de règles décrivant une opération non encore utilisée en partie droite auparavant et réduisant au moins une autre opération F déjà définie.

Exemple 4 :

Le type abstrait File du paragraphe 6.1 du chapitre 1, obéit au critère de Musser. Par contre, le type abstrait décrit par les axiomes (écrits sous forme simplifiée) :

$OT(\wedge) \rightarrow \wedge$
$OT(\wedge.a) \rightarrow \wedge$
$OT(f.a.b) \rightarrow OT(f.a).b$
$FR(\wedge) \rightarrow ?$
$FR(\wedge.a) \rightarrow a$
$FR(f.a.b) \rightarrow FR(f.a)$
$f * \wedge \rightarrow f$
$f * (f'.a) \rightarrow f \quad FR(f'.a) * OT(f'.a)$

ne vérifie pas le critère de Musser. Un raisonnement sur la complexité du deuxième opérande montre que tout terme contenant * se réduit en un terme ne contenant pas * .

Comparons les deux systèmes sur la réduction $f * (\wedge.a.b)$ pour le premier on a :

$f * (\wedge.a.b) \xrightarrow{CONCAT2} [f *(\wedge.a)] .b \xrightarrow{CONCAT2} (f*\wedge).a.b$

$\xrightarrow{CONCAT1} f.a.b$

pour le second on a

$f *(\wedge.a.b) \xrightarrow{CONCAT2'} [f. FR(\wedge.a.b)]* OT(\wedge.a.b)$

$\xrightarrow{FR3} [f.FR(\wedge.a)] * OT(\wedge.a.b) \xrightarrow{FR2} (f.a)* OT(\wedge.a.b)$

$\xrightarrow{OT3} (f.a)* [OT(\wedge.a).b] \xrightarrow{OT2} (f.a) * (\wedge.b)$

$\xrightarrow{CONCAT2'} [(f.a) . FR(\wedge.b)]* OT(\wedge.b)$

$\xrightarrow{FR2} (f.a.b)* OT(\wedge.b) \xrightarrow{OT2} (f.a.b)* \wedge$

$\xrightarrow{CONCAT1} f.a.b.$ □

Cet exemple montre que le critère de Musser est parfois insuffisant même dans les systèmes de réécriture associés aux spécifications algébriques de types abstraits. Parmi les diverses autres solutions proposées pour tester la terminaison, aucune ne semble satisfaisante.

1) l'ordre de Knuth et Bendix est trop complexe et trop ad hoc

2) l'ordre de Plaisted [PLA 78] bien qu'assez naturellement déduit d'un ordre sur les symboles fonctionnels n'est pas utilisable ici.

3) les méthodes fondées sur des interprétations par des fonctions de variables entières supposent une certaine intuition pour découvrir les bonnes fonctions ; en ce qui concerne l'exemple ci-dessus , elles ne sont pas évidentes. De toute façon il semble difficile d'en déduire une méthode automatisable [HUE 77] [LAN 77] .

2.- Types abstraits et algèbres de type.

Guttag et Horning [GUT 78] donnent la définition suivante :

Définition 2 :

Un système axiomatique de type abstrait est *suffisamment complet* si les formes normales des termes de profil → Ext_i existent et ne sont constituées que d'opérations internes à Ext_i □

A une description de type abstrait est associée une classe d'algèbres notées $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_m]$ appelée algèbre de type Ti et définie ainsi :

1) les sortes sont Ti, $Ext_1, \ldots, Ext_m$.

2) les opérations sont celles qui apparaissent dans la partie "fonctionnalité" ainsi que les termes de profil : Ext_i , construits uniquement avec les opérations de type Ext_i , correspondant aux éléments de types déjà définis.

3) les algèbres sont premières

4) elles sont modèles du système d'équations apparaissant dans la partie axiomatique (notée AX)

Proposition 1 :

si AX est suffisamment complet et si $\mathfrak{A}$ est une algèbre première $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_n]$ alors Ext_i est une image de termes du type Ext_i

Démonstration :

Puisque l'algèbre est première, Ext_i ne contient que des images de termes de profil :()→ Ext_i de même plus précisément des formes normales de ces termes et d'après la complétude suffisante ces formes normales sont de termes du type Ext_i □

Définition 3 :

Toute algèbre de $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_m]$ est un modèle du type abstrait □

C'est en termes d'algèbres ce qu'énonce Guttag. D'après le corollaire de la proposition 4 (§ 2.4) l'algèbre initiale de la variété des modèles de AX est première, donc c'est un modèle du type abstrait. Pour ADJ [i.e. Goguen, Thatcher, Wagner et Wright] , c'est le seul. En revanche, en considérant la classe $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_m]$, nous formalisons le point de vue de Guttag et Horning. Broy, Dosch, Portsch, Pepper et Wirsing [BOR 79] montrent qu'une telle classe admet une algèbre terminale (cf proposition 2). On peut admettre que cette algèbre terminale est le modèle le plus économique dans un certain sens.

Exemple 5 :

Le type abstrait Ensemble tel qu'il a été présenté au paragraphe 5.2 du chapitre 1 , possède une famille assez riche d'algèbres de type. Le premier modèle (les ensembles classiques) est l'algèbre terminale, le second modèle (représentation en listes) est l'algèbre initiale. Le troisième et ceux qui lui sont apparentés et le quatrième , sont deux algèbres de type parmi d'autres. Notons que l'algèbre terminale vérifie outre les équations portant sur PR , deux équations

$$e + x + x = e + x$$
$$e + x + y = e + y + x$$

Le troisième modèle vérifie

$$e + x + 0 = e + 0 + x$$
$$e + 0 + 0 = e + 0$$

Le quatrième modèle vérifie une infinité d'équations

$$e + + \sum_{i=1}^{n} x_i + x = e + x + \sum_{i=1}^{n} x_i$$

Quant à l'algèbre initiale, comme dans tous les autres cas, elle ne vérifie aucune équation autre que celles qui sont déduites de l'axiomatique du type abstrait □

Soit F l'ensemble des opérations du type abstrait. Supposons que les types $\underline{\underline{EXT}}_i$ n'ont qu'une algèbre de type à un isomorphisme près.

$\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_n]$ est une classe d'algèbres paramétrées par les algèbres initiales des $\underline{\underline{EXT}}_i$. L'algèbre $\mathcal{T}$ telle que $Ti_{\mathcal{T}} = \{ \mu(t)/t$ terme sous variable$\}$ et $f_{\mathcal{T}}(t_1, \ldots, t_m) = \mu(f(t_1, \ldots, t_n))$ est l'algèbre initiale de $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_m]$. L'algèbre terminale est décrite par la proposition suivante.

Proposition 2 :

On définit sur $\mathcal{T}$ (plus précisément sur $Ti_{\mathcal{T}}$) la congruence $\textcircled{H}$ telle que : $u \textcircled{H} v$ ssi pour tout $i \in [1..\ n]$, pour tout terme $\delta : Ti \rightarrow Ext_i$ (à une variable) $\mu(\delta(u)) = \mu(\delta(v))$

$\mathcal{T}/_{\textcircled{H}}$ est l'algèbre terminale de $T\ [Ext_1, \ldots, Ext_m]$

Démonstration :

Considérons pour simplifier une opération $f : Ti\ \ Ext_i \rightarrow Ti$. Montrons que $u \textcircled{H} v \Rightarrow f(u, a) \textcircled{H} f(v, a)$. Par définition $f(u, a) = \mu(f(u, a))$. Donc si $\delta : Ti \rightarrow Ext_i$ n'a qu'une variable

$\mu(\delta(f(u, a))) = \mu(\delta(\mu(f(u, a))))$

= (d'après la confluence) $\mu(\delta(f(u, a)))$

or $\delta(f(x, a)) : Ti \rightarrow Ext_i$, ainsi d'après $u \textcircled{H} v$

$= \mu(\delta(f(v, a)) = \mu(\delta(f(v, a)))$

d'où le résultat.

$\mathcal{T}/_{\textcircled{H}}$ est, par conséquent, une algèbre première qui vérifie $\underline{AX}$. $\mathcal{T}/_{\textcircled{H}}$ est terminale, en effet toute algèbre de $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_m]$ est

est isomorphe à $\mathcal{T}/_{\textcircled{H}'}$ pour une congruence $\textcircled{H}'$ car les morphismes sont surjectifs, il reste à prouver que $\textcircled{H}$ est la congruence la plus fine sur $\mathcal{T}$, autrement dit que $u \textcircled{H}' v \Rightarrow u \textcircled{H} v$; puisque $\textcircled{H}'$ est une congruence $u \textcircled{H}' v \Rightarrow$ (pour tout δ: $Ti() \rightarrow Ext_i$ $\delta(u) \textcircled{H}' \delta(v)$ or $\textcircled{H}'$ est l'unique congruence possible dans les algèbres de type $\underline{\underline{EXT}}_i$, c'est l'égalité des formes normales donc pour tout $\delta : Ti \rightarrow Ext_i$, $\mu(\delta(u)) = (u(v))$ c'est précisément $u \textcircled{H} v$ □

Définition 4 :

Un type abstrait à la propriété de *complète discrimination* si $u \in Ti$, $v \in Ti$ alors $\mu(u) \neq \mu(v)$ implique qu'il existe $\delta : Ti \rightarrow Ext_i$ (à une seule variable) tel que $\mu(\delta(u)) \neq \mu(\delta(v))$ □

Proposition 3 :

Si un type abstrait à la propriété de complète discrimination, alors $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_m]$ ne contient qu'une algèbre à un isomorphisme près

Démonstration :

D'après la propriété de complète discrimination, la congruence $\textcircled{H}$ est l'égalité, donc $\mathcal{T}$ est isomorphe à $\mathcal{T}/_{\textcircled{H}}$; ainsi pour toute algèbre $\mathcal{B}$ de $\underline{\underline{TI}}\ [Ext_1, \ldots, Ext_m]$, il existe un unique morphisme surjectif de $\mathcal{T}$ vers $\mathcal{B}$ et un unique morphisme surjectif de $\mathcal{B}$ vers $\mathcal{T}$, ce sont des isomorphismes. □

Exemple 6 :

Le type abstrait Arb [Alph] a la propriété de complète discrimination, l'algèbre AAF [Alph, Bool] du théorème est donc à un isomorphisme près la seule algèbre de type Arb [Alph] □

Exemple 7 :

Le type File a la propriété de complète discrimination

Exemple 8 :

Le type Ens n'a pas la propriété de complète discrimination. Par exemple AJOUT(AJOUT(VIDE, a), b) ne peut être discriminé de AJOUT(AJOUT(VIDE, Comme on l'a vu dans l'exemple 5 ENS [Alph] contient d'autres algèbres que l'algèbre initiale □

3.- ALGÈBRES HÉTÉROGÈNES ET PARAMÉTRÉES ET REPRÉSENTATION ALGÉBRIQUE D'UN TYPE ABSTRAIT.

Rappelons le problème de la représentaion, la spécification algébrique d'un type abstrait définit une classe d'algèbres appelées algèbres de type. La démarche à suivre est la suivante : on cherche à représenter l'une de ces algèbres, ou, ce qui revient au même si l'on prend le problème dans l'autre sens, on tient une représentation pour correcte si elle modélise une algèbre de type ; il s'agit d'associer à chaque point de la sorte Ti de l'algèbre de type choisie un objet mathématique dont on décrit la structure interne, cela conduit à spécifier les relations entre divers constituants internes à la strucutre, par exemple à expliciter des "opérations" algébriques appelées

aussi accès, entre ces points (au chapitre 5, un autre formalisme est proposé) ; Nous allons décrire, ce qu'on peut dégager des exemples.

3.1.- DESCRIPTIONS DES ALGÈBRES FILLES.

Ce sont des algèbres premières homogènes paramétrées par des algèbres de type prises dans EXT1 ... EXTn parmi les ensembles on distingue :

- d'une part le support de l'algèbre proprement dite. Nous le noterons Sup. Il est toujours fini ; comme l'algèbre est première, il est constitué de points qui peuvent être représentés par un terme nullaire. Dans les arbres binaires ce support est Noe. A noter que ce support peut être vide (cas de l'arbre vide) ou pour certaines représentations être inexistant (cas de la représentation des ensembles par la fonction d'appartenance).

- d'autre part, les algèbres externes ; en général on ne s'intéresse ni à leurs opérations internes ni à la manière dont elles sont représentées.

Méthodologiquement , il s'avère intéressant de classer les accès c'est à dire les opérations de ces algèbres en trois familles suivant leur profil :

- les fonctions de profil $Sup^k \rightarrow Sup$; ces fonctions relient entre eux les points du support Sup ; elles permettent de cheminer dans algèbre. Ces fonctions sont en général partiellement définies

- les fonctions de profil $(Exti_1, \ldots, Exti_n) \rightarrow Sup$ ces fonctions associent aux valeurs externes des valeurs du support de l'algèbre ; ces fonctions servent à accéder directement aux points du support. Dans les arbres c'est la fonction T : () → Noe .

- les fonctions de profil (Sup) → Exti

- plus rarement les fonctions $(Exti_1, \ldots, Exti_m) \rightarrow Extj$. En général, cela arrive quand Sup est vide et que l'algèbre est réduite à ces

opérations. Pour les ensembles, il s'agit de la fonction $\in$: (Alph) → Bool .

Enfin, les algèbres vérifient des axiomes destinés à caractériser celles qui sont candidates à représenter des objets de Ti .

3.2.- LES CONSTRUCTIONS COMME REPRÉSENTATION DES OPÉRATIONS

Suivant leur profil , les opérations du type abstrait se représentent d'une manière ou d'une autre :

- une opération du profil $Ti^k \times Exti_1 \times \ldots \times Exti_m \rightarrow Ti$, est représentée par une méthode construction d'un nouvelle algèbre à l'aide de k algèbres données ; il s'agit véritablement de constructions au sens mathématique classique du terme, comme le sont les produits ou les quotients dans les théories classiques des algèbres, un certain nombre de *définitions* de nouveaux accès en fonction des accès précisent cette construction.

- une opération de profil $Ti \times Exti_1 \times \ldots \times Exti_m \rightarrow Extj$ est représentée par une sélection, c'est la description de l'extraction d'une valeur de l'algèbre en se servant des paramètres appartenant respectivement à $Exti_1 \times \ldots \times Exti_m$

- une opération de profil $Ti^k \times Exti_1 \times \ldots \times Exti_m \rightarrow Extj$ est représentée par une sélection plus complexe car on extrait une valeur externe à partir de plusieurs algèbres, en général cette valeur est booléenne et le résultat est une comparaison, une égalité par exemple.

Exemple 9 :

Dans les arbres binaires, les constructions *Cons* et *Gau* sont du premier type, la sélection *Et* est du second type. Dans les ensembles *Obj* est du premier type, *Pr* du second □

3.3.- ELÉMENTS D'UNE MÉTHODOLOGIE.

Un formalisme de représentation, devrait servir à systématiser autant que possible la construction de la représentation. Comme le proposent beaucoup d'auteurs, il est intéressant de rester le plus longtemps possible au niveau des types abstraits (au niveau statique disent certains) car c'est là qu'en l'état actuel, les preuves sont les plus faciles à automatiser notamment grâce aux concepts de réécritures.

En particulier Guttag, Horowitz et Musser [GUT 78] proposent la méthode suivante ; ils appellent implantation une fonction allant d'un type abstrait dans un autre type abstrait. On peut prouver formellement la correction de cette fonction par les techniques de l'algèbre. Très sommairement, il suffit de prouver à l'aide des règles du type abstrait d'arrivée que l'implantation conserve les règles du type abstrait de départ et en particulier celle concernant les sélections. Le système DTVS (Data Types Vérification System) [MUS 77] et sa nouvelle version dans AFFIRM [MUS 79] sont des logiciels qui permettent de conduire automatiquement cette preuve . OBJ de Goguen et Tardo est un système apparenté [GOG 79] . Actuellement Fernand Reinig, à Nancy, développe les premières bases d'un système de réécriture dont nous essaierons de tirer un profit semblable. Darlington [DAR 78] , Jones [JON 79] Gaudel et Terrine [GAU 78a][GAU 78b] et Remy [REM 79] étudient comment déduire l'implantation en réduisant au maximum les hypothèses et les choix arbitraires . Ils utilisent et étendent des méthodes inspirées de Burstall et Darlington [BUR 77] . Procédant du même principe, il faut citer un article de Standish qui propose quelques réflexions à propos de l'implantation systématique des files sous forme de listes [STA 78] .

A la lumière des alinéas précédents, il apparaît que, pour représenter un type abstrait dans un langage de programmation on peut procéder comme suit ; tout d'abord, on l'implante efficacement dans un autre type abstrait dont la représentation par des algèbres filles est aisée à définir ; ensuite on définit et formalise la représentation. Construire systématiquement ces deux transitions reste un problème qui devra être résolu. Voici deux suggestions.

1) les concepts de la réécriture et l'étude des algèbres terminales doivent permettre de systématiser le construction des implantations. En particulier la recherche d'une famille, si possible finie, d'identités permettant de caractériser l'algèbre terminale (ou peut être une autre algèbre) doit aider à découvrir des propriétés utiles à la construction de l'implantation et du type d'arrivée.

2) l'étude de la structure du type abstrait doit permettre, un peu à la manière du paragraphe 4, de systématiser la représentation. Les opération peuvent alors être présentées, en s'inspirant de la figure 7 du chapitre 1er, par un cas "général" et des "exceptions", cela fournit un canevas pour leur construction (voir aussi la figure 2 du chapitre 4).

4.- UNE_REPRÉSENTATION_CANONIQUE.

Dans ce paragraphe , on va donner une réponse à la question suivante : existe-t-il pour chaque type abstrait qui forme un système confluent, noethérien et suffisamment complet, une classe d'algèbres filles paramétrées qui en soit un modèle ? On va répondre affirmativement à la question en proposant une représentation canonique de l'algèbre de type initiale. On a déjà vu que l'algèbre initiale n'est peut être par la meilleure représentation, on verra de plus par l'exemple des Listecirculaires que la représentation à laquelle on aboutit n'est même pas optimale en nombre d'opérations élémentaires parmi les représentations de l'algèbre initiale. Par contre, elle présente un intérêt théorique et pratique puisqu'on sait ainsi qu'il est toujours possible de construire une représentation d'un type abstrait à condition qu'il soit confluent, noethérien et suffisamment complet. La représentation que nous proposons est apparentée à l'implantation directe de Guttag, Horowitz et Musser [GUT 78] .

Tout d'abord, on s'appuie sur la description de l'algèbre initiale par les formes normales , telle qu'elle a été décrite au paragraphe 2. On

procède en deux temps :
on implante les termes plus précisément les formes normales, dans un type abstrait canonique : les ramifications, et ensuite on représente ce type canonique par les petites algèbres. Pour l'exposé, nous décrirons d'abord les ramifications puis comment les termes et les réécritures s'expriment par les ramifications. Finalement les résultats sont synthétisés dans l'énoncé du théorème fondamental.

4.1.- RAPPELS SUR LES RAMIFICATIONS [PAI 68] [PAI 77]

Une ramification sur Alph est une suite d'arborescences orientées étiquetées par Alph, elles forment l'ensemble Alph. On peut munir Alph d'une structure algèbrique dite de binoïde (cf. figure 4).

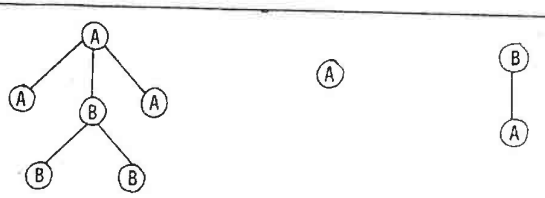
- une loi de composition interne, i.e. une opération binaire est notée + , elle est associative et possède un élément neutre noté $\wedge$. La loi + s'interprète comme la concaténation de deux suites de ramifications et $\wedge$ est la ramification vide i.e. la suite vide d'arbres.

- une opération x fait correspondre, à un élément a de Alph et une ramification, une ramification, x s'interprète comme la construction d'une arborescence étiqutée à partir d'une ramification en enracinant l'étiquette au-dessus de la ramification.

Afin de décrire les binoïdes de ramifications comme une algèbre d'un type abstrait ayant la propriété de complète discrimination, nous ajoutons trois opérations (dont les noms font des références évidentes à la généologie) :
la souche SOU, la descendance DES et la fratrie FRA . (cf. figure 5)

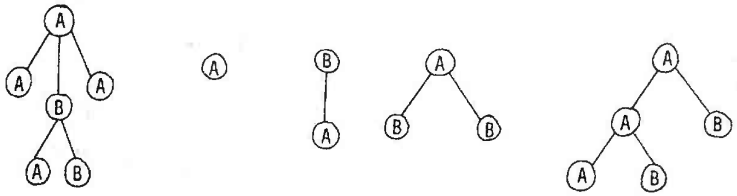
SOU : Binoïde → Alph
DES : Binoïde → Binoïde
FRA : Binoïde → Binoïde

La souche associe à une ramification l'étiquette de la racine de la première arborescence.

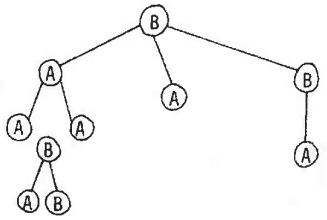


Une ramification r

Une ramification s



Une ramification r + s

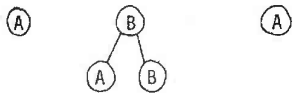


Une ramification B x r

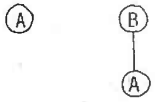
Figure 4 : Quelques ramifications.

La descendance est la ramification obtenue à partir de la première arborescence en enlevant sa racine.

La fratrie est la suite d'arborescences obtenue en enlevant la première arborescence.



La descendance de r (cf. figure 4)



La fratrie de r (cf. figure 4)

Figure 5 : La descendance et la fratrie d'ure ramification.

Type Binoïde [Alph]

fonctionnalité

+ : (Binoïde, Binoïde) → Binoïde
x : (Alph, Binoïde) → Binoïde
∧ : → Binoïde
SOU : Binoïde → Alph U {?}
DES : Binoïde → Binoïde
FRA : Binoïde → Binoïde

Axiomatique

SOU(∧) = ?
SOU(a x r)+ s) = a
DES(∧) = ∧
DES((a x r) + s) = r
FRA(∧) = ∧
FRA((a x r)+ s) = s

Type Ram [Alph]

Fonctionnalité

$\wedge$: () → Ram,

-*-+-: (Alph, Ram, Ram) → Ram ;

SOU : (Ram) → Alph U {?} ,

DES, FRA : (Ram) → Ram ;

$=_{Ram}$: (Ram, Ram) → Bool ;

\+ : (Ram, Ram) → Ram

x : (Alph, Ram) → Ram

Axiomatique

SOU () = ?

SOU(A x r + t) = A

DES($\wedge$) = $\wedge$

DES(A x r + t) = r

FRA($\wedge$) = $\wedge$

FRA(A x r + t) = t

($\wedge$ $=_{Ram}$ $\wedge$) = VRAI

($\wedge$ $=_{Ram}$ a x r + s) = FAUX

(a x r + s $=_{Ram}$ $\wedge$) = FAUX

(a x r + t $=_{Ram}$ b x u + v) = (a $=_{Alph}$ b) ET (r $=_{Ram}$ u) ET (t $=_{Ram}$ v)

$\wedge$ + r = r

(A x r + t) + v = A x r + (t + v)

A x r = A x r + $\wedge$

Type Ramvar [Alph, Var]

Fonctionnalité

$\wedge$: () → Ramvar ;

- *-+- : (Alph, Ramvar, Ramvar) → Ramvar ,

R : (Var) → Ramvar

SOU : (Ramvar) → Alph U {?}

DES, FRA : (Ramvar) → Ramvar

Figure 6 (début)

Axiomatique

SOU($\wedge$) = ?

SOU(R(x)) = ?

SOU(A x r + s) = A

DES($\wedge$) = $\wedge$

DES(R(x)) = $\wedge$

DES(A x r + s) = r

FRA($\wedge$) = $\wedge$

FRA(R(x)) = $\wedge$

FRA(A = r + s) = s

Type Sub [Ram, Var]

Fonctionnalité

IN : () → Sub

AFF : (Var, Ram, Sub) → Sub

VAL : (Var, Sub) → Ram U {$\perp$}

Axiomatique

VAL(x, AFF(y, r, s)) = *si* x = y *alors* r *sinon* VAL(x, s)

VAL(x, IN) = $\perp$

Figure 6 : Des types abstraits. (fin)

Remarquons la présence d'une opération de profil (Binoïde, Alph, Binoïde) → Binoïde qui à (r, a, t) fait correspondre a x r + t . Elle confère à Alph une structure d'algèbre de la variété ARB [Alph] . Plus précisément

Lemme 1 :

(Alph ; □ x □ + □, DES, FRA, SOU, $\wedge$) forme une algèbre initiale de ARB [Alph] .

4.2.- DESCRIPTION DE TERMES COMME DES RAMIFICATIONS ET RÉÉCRITURES.

On sait représenter l'algèbre initiale de ARB [Alph] par des algèbres filles. Pour décrire complètement la représentation canonique, il suffit de décrire les termes sur F par des ramifications sur l'alphabet F , et les réécritures par des fonctions sur les ramifications qui utilisent l'opération a x r + t . Les termes sans variables sont des éléments de Ram (cf. figure 6), les termes contenant des variables sont des éléments de Ramvar (cf. figure 6). Les substitutions sont des éléments de Sub.

Sur ces types abstraits nous définissons trois fonctions (primitives récursives de ramifications (voir [QUE 69])

FILTRE : (Ram, Ramvar) → Sub U {!}

qui , étant donnée une ramification r et une ramification avec variable t fournit une substitution σ telle que $t\sigma = r$.

EVAL (Ramvar, s) → Ram

qui, étant donnée une ramification avec variables, fournit la ramification obtenue par substitution, grâce à s , des variables.

REECRIRE (Ram, Ramvar, Ramvar) → Ram

qui réécrit une ramification r en se servant d'une règle g → d . Toutes les sous-ramifications disjointes que l'on rencontre depuis la souche et qui peuvent être identifiées à g sont remplacées. Cette restriction n'a pas d'importance, car le système, étant confluent, l'ordre des réécritures, n'a pas d'importance.

Les fonctions sont primitives récursives de ramifications, leurs calculs ne font intervenir que les opérations ∧ et -x-+- et se transcrivent aisément par des compositions d'opérations Cons dans AAF [Alph] .

La réécriture pour une liste de règle est donnée par

REEC : (Ram, Liste (Couple(Ramvar, Ramvar)) (pour le type couple voir chapitre 4 paragraphe 2)

REEC (r, VIDE) = r

REEC (r, AJ(e, c)) = REEC(REECRIRE(r, P1(c), P2(c)), e)

La forme normale pour un e donné est la fonction

FN : (Ram, Liste (Couple(Romvar, Romvar)) → Ram

FN(r, e) = SI REEC (r, e) = r ALORS r
SINON FN(REEC(r, e), e)

```
FILTRE : (Ram, Ramvar) → Sub U {!}
      FILTRE(r, t) = FIL(r, t, IN)
FIL : (Ramvar, Sub U ! ) → Sub U !
FIL(∧, ∧, s) = s
FIL(a x r + t, ∧, s) = !
FIL(∧, b x u + v, s) = !
FIL(r, R(x), s) =
      SI s = ! ALORS !
      SINON SI VAL(x, s) = ⊥ ALORS AFF(x, r, s)
            SINON SI VAL(x, s) =Ram r ALORS s
                  SINON !

EVAL(Ramvar, Sub) → Ram U {⊥}
      EVAL(∧, s) = ∧
      EVAL(R(x), s) = VAL(x, s)
      EVAL(a x r + t, s) =
         SI EVAL(r, s) = ⊥ OU EVAL(t, s) = ⊥ ALORS ⊥
         SINON a x EVAL(r, s) + EVAL(t, s)
REECRIRE(Ram, Ramvar, Ramvar) → Ram
      REECRIRE (∧, g, d) = ∧
      REECRIRE (a x r + t, g, d) =
         SI FILTRE (a x r + t, g) ≠ !
            ALORS EVAL(d, FILTRE(a x r + t, g))
            SINON a x REECRIRE(r, g, d) + REECRIRE(t, g, d)
```

Figure 7 : Trois fonctions qui servent pour la réécriture.

4.3.- LE THÉORÈME DE LA REPRÉSENTATION CANONIQUE.

Enoncé :

Pour chaque type abstrait spécifié par un système de réécriture convergent (c'est à dire confluent - noethérien), et suffisamment complet, il existe une représentation de l'algèbre initiale $\mathcal{T}$ dans une classe d'algèbres ; les algèbres sont des arbres binaires et les constructions traduisent la réécriture.

Démonstration :

Cela découle du fait que $\mathcal{T}$ décrite au paragraphe 2 est une algèbre initiale , du lemme 1 (paragraphe 4.1) et des lemmes 2 et 3.

Lemme 2 :

Si F est l'ensemble des symboles fonctionnels, tout terme est représenté par un élément unique de l'algèbre initiale de Ram [F] élément unique de Ram [F]

Démonstration :

Posons β l'application définie récursivement ainsi

si $ar(f) = 0$ alors $\beta(f) = f \times \wedge$

si $ar(f) = n > 0$ alors

$$\beta(f(t_1, \ldots, t_n)) = f \times (\beta(t_1) + \ldots + (t_n))$$

est injective □

Lemme 3 :

Toute réécriture peut être décrite à partir des opérations CONS et VIDE (i.e. $\square \times \square + \square$ et $\wedge$)

Démonstration :

elle découle immédiatement d'un examen attentif des descriptions du paragraphe 4.2 □

Ainsi puisque chaque terme sous sorme normale peut être représenté par une ramification, qu'il existe une interprétation des ramifications comme des arbres binaires et puisque toute réécriture peut être exprimée par $\square \times \square + \square$ et $\wedge$ la représentation du chapitre 1 paragraphe 2 satisfait le théorème □

4..4- L'EXEMPLE DES LISTES CIRCULAIRES.

On va montrer sur un exemple que la construction proposée n'est parfois pas efficace. Considérons le type abstrait Listecirculaire.

Type Listecirculaire [Alph]

Fonctionnalité

LVIDE :() → Listecirculaire
AJ : (Listecirculaire, Alph) → Listecirculaire,
RDT :(Listecirculaire) → Listecirculaire
TET :(Listecirculaire) → Alph U {INDEF}

Axiomatique

ROT(LVIDE) = LVIDE
ROT(AJ(LVIDE, a)) = AJ(LVIDE, a)
ROT(AJ(AJ(a, b), a) = AJ(ROT(AJ(x, a)), b)
TET(x, b) = b
TET(LVIDE) = INDEF

La figure 8 représente les différentes représentations obtenues quand on applique ROT sur la Listecirculaire AJ(AJ(AJ(LVIDE, c), b), a) . On remarque qu'il a fallu quatre états pour une représentation que l'on peut penser faire beaucoup plus simplement, par des algèbres premières. On définit deux opérations nullaires

T, Q : () → Noe

une opération unaire

S : (Noe) → Noe .

S est partout définie et vérifie

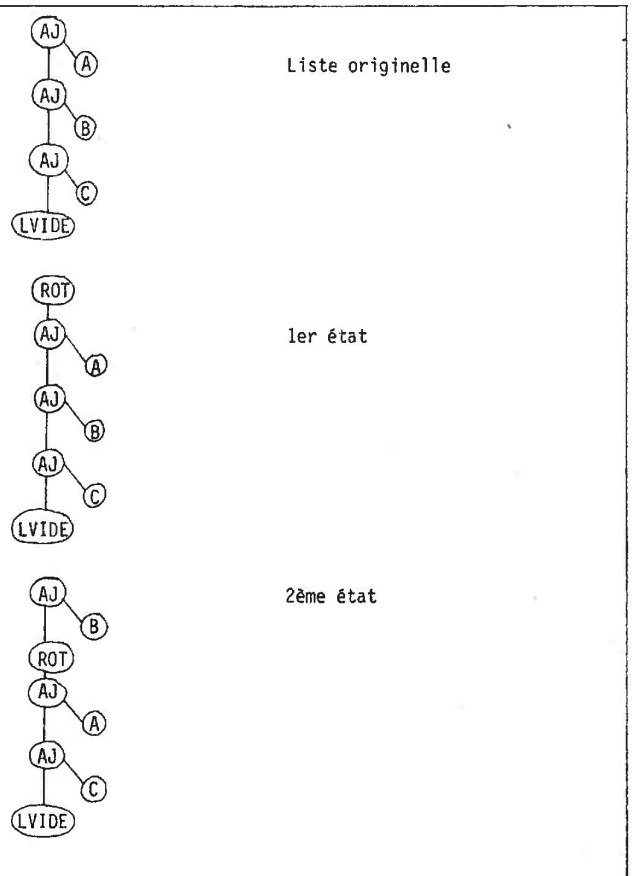
$S(Q) = R$

Il est facile de construire $\mathcal{R}ot(\mathfrak{L}) = \mathfrak{B}$ ainsi (figure 15)

$R_{\mathfrak{B}} = S_{\mathfrak{L}}(R_{\mathfrak{L}})$

$S_{\mathfrak{B}} = S_{\mathfrak{L}}$

$Q_{\mathfrak{B}} = S_{\mathfrak{L}}(Q_{\mathfrak{L}})$ qui est par conséquent $R_{\mathfrak{L}}$



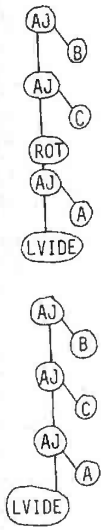


Figure 8 : L'opération Rot sur la représentation canonique des listes circulaires.

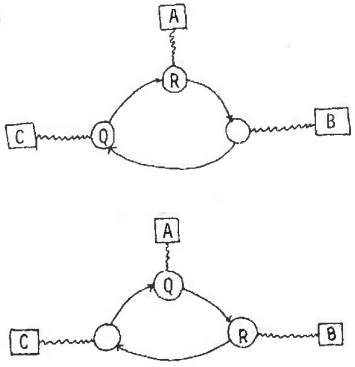


Figure 9 : L'opération $\mathcal{R}ot$ dans une représentation plus naturelle des listes circulaires

Il semble que la deuxième "représentation" est celle qui est la plus naturelle pour qui cherche à s'imaginer une liste circulaire. On peut s'étonner que le type abstrait suggère la première : cela provient du fait que la spécification algèbrique, autrement dit la réécriture, privilégie la représentation par des arbres, au détriment des autres. Il s'agit d'ailleurs d'un problème plus général courant dans les bases de données : il est difficile de spécifier simplement et fidèlement une structure comportant des circularités et réciproquement d'implanter efficacement et sans erreurs la spécification d'une telle structure.

CHAPITRE III

outils algèbriques pour le calcul sur les parties d'une algèbre de type

CHAPITRE III

OUTILS ALGÉBRIQUES POUR LE CALCUL

SUR LES PARTIES D'UNE ALGEBRE DE TYPE

I. - INTRODUCTION.

Les types abstraits que nous avons étudiés, comportaient uniquement des opérations déterministes, autrement dit des opérations qui à un uple de valeurs font correspondre au plus une valeur. Dans le cas des types abstraits comportant des opérations indéterministes, une opération appliquée à un uple unique peut rendre une multitude de résultats. Ainsi, puisqu'une opération peut produire un ensemble de résultats et du fait que les opérations se composent, il faut étudier comment se comportent les structures algébriques et les identités, quand on passe des opérations définies sur les uples d'éléments aux opérations définies sur les uples d'ensembles. Ce genre de types abstraits, avec opérations indéterministes, intervient particulièrement dans les bases de données , où les opérations ont, en général, la forme "un des..." ou "tous les ..." . L'objectif de ce chapitre est de caractériser les axiomes licites dans un tel type abstrait. Cette caractérisation se fait à l'aide des propriétés de *linéarité* (ie au plus une occurrence de chaque variable dans chaque membre) et de *régularité* (ie. mêmes ensembles de variables dans chaque membre). Ces propriétés qui s'énoncent bien dans le cas des algèbres

homogènes (cf par exemple [BLE 73]) demandent d'être approfondies dans le cas des algèbres hétérogènes.

On peut désirer faire sur les algèbres hétérogènes, une théorie similaire à la théorie des langages sur monoïde libre, c'est-à-dire pouvoir définir des ensembles algébriques, plus petit point fixe d'un système d'équations. Pour cela, il faut être capable d'étendre les opérations à l'ensemble des parties, et s'assurer que ces extensions vérifient les mêmes identités comme c'est le cas dans les monoïdes. Nous esquissons au paragraphe 6 une telle théorie.

On peut, ainsi qu'il vient d'être dit, caractériser certains ensembles algébriquement ; cette caractérisation par équation est plus maniable qu'un prédicat, car, en particulier, elle permet par des calculs simples à mettre en oeuvre, d'obtenir éventuellement d'autres équations , plus explicites ou plus maniables. Citons à titre d'exemple le fait suivant :

Quand on représente un type abstrait dans un autre type abstrait, l'image de la fonction de représentation n'est pas, en général, tout le type représentant. Guttag appelle "invariant de représentation" le prédicat caractérisant cette image ; il est souhaitable de remplacer celui-ci par une équation.

Enfin, signalons que cette étude a trait aux opérations dont l'interprétation est faite par "appel par nom" ; c'est-à-dire qu'on évalue une opération composée avant d'évaluer ses paramètres.

2.- Etude d'exemples.

Avant de passer à des résultats théoriques, nous allons citer quelques exemples et étudier la manière de "calculer" sur des ensembles de parties.

2.1.- Les groupes.

Considérons les groupes définis comme un type abstrait

fonctionnalité

$e : \longrightarrow$ Groupe ,

$^{-1} :$ Groupe $\longrightarrow$ Groupe , (noté en postfixé)

$. :$ Groupe x Groupe $\longrightarrow$ Groupe, (noté en infixé)

axiomatique

$(x \,.\, y) \,.\, z = x \,.\, (y.z)$,

$x \,.\, e = x$,

$e \,.\, x = x$,

$x \,.\, x^{-1} = e$,

$x^{-1} \,.\, x = e$,

soit $\mathcal{G}$ un groupe. Posons

$H \bar{.} K = \{h.k \mid h \in H \wedge k \in K\}$

$\bar{e} = \{e\}$

$H^{\bar{-1}} = \{h^{-1} \mid h \in H\}$

On sait que $< 2^G ; \bar{.} , \bar{^{-1}} , \bar{e} >$ et $< 2^G - \{\emptyset\} ; \bar{.} , \bar{^{-1}} , \bar{e} >$ (si $G \neq e$) ne sont pas des groupes. Certes $\bar{.}$ est associative et $\bar{e}$ est bien élément neutre de $\bar{e}$ ', mais on sait que si H est le support d'un sous-groupe de $\mathcal{G}$ non trivial alors

$$H \bar{.} H^{\bar{-1}} = H \neq \{e\} = \bar{e}$$

en effet, l'équation $x \,.\, x^{-1} = e$ contient deux occurrences de x à gauche et ainsi elle n'est plus vérifiée sur l'ensemble des parties. Par contre, les trois premières équations s'étendent à l'ensemble des parties et font de

$< 2^G ; \bar{.} , \bar{e} >$ un monoïde.

2.2.- LES ALGÈBRES LINÉAIRES.

Dans [LES 78] a été étudié un type abstrait (ou plus précisément une variété d'algèbres homogènes) qui intervient dans l'étude des langages et qui est appelé type des algèbres linéaires, car celles-ci sont naturellement liées aux langages dit "linéaires". Le mot "linéaire" n'est pas fortuit, car ces langages sont engendrés par des systèmes d'équations formées de mots qui contiennent au plus une occurrence de variables. Cependant, l'assimilation d'une définition à l'autre n'est pas complète ; dans un cas une occurrence de chaque variable au plus, dans l'autre une occurrence parmi toutes les variables au plus.

Les algèbres linéaires ont été choisies afin de montrer un exemple très simple, qui n'est pas celui des monoïdes, sur lequel notre théorie peut s'appliquer.

Soit Alph un ensemble de ν symboles et soit $\overline{Alph}$ un ensemble de ν symboles associés aux symboles de Alph .

1) La fonctionnalité est constitué d'un symbole.

VID :()→ Lin

de ν symboles appartenant à Alph

A :(Lin)→ Lin

(Nous noterons souvent ces opérations a. x)

et ν symboles appartenant à $\overline{Alph}$

$\bar{A}$:(Lin) → Lin

(Nous noterons souvent ces opérations x. a)

2)L'axiomatique est constituée de deux familles d'identités :

- famille des identités assurant l'équivalence des opérations A et $\bar{A}$ sur VID . On a une famille de ν identités qui s'écrivent sous la première forme

$$A(VID) = \bar{A}\ (VID)$$

et sous la deuxième forme

$$A.VID = VID.A$$

- famille des identités assurant une forme d'associativité sur les applications à droite et à gauche.

On a 2ν identités qui s'écrivent sous la première forme

$$A(\bar{B}\ (x)) = \bar{B}(A\ (x))$$

et sous la deuxième forme

$$A.(x.\ B) = (A.\ x).\ B$$

On note LIN [Alph] la variété d'algèbre ainsi définie .

Si Alph = {A, B} alors $\overline{Alph}$ = {$\bar{A}$, $\bar{B}$} , on obtient une description sous forme de type abstrait ainsi :

fonctionnalité

VID :()→ Lin ,

A, B, $\bar{A}$, $\bar{B}$: (Lin) ⟶ Lin .

axiomatique

$A(VID) = \bar{A}\ (VID)$,

$B(VID) = \bar{B}\ (VID)$,

$A(\bar{A}(x)) = \bar{A}\ (A(x))$,

$A(\bar{B}(x)) = \bar{B}\ (A(x))$,

$B(\bar{A}(x)) = \bar{A}\ (B(x))$,

$B(\bar{B}(x)) = \bar{B}\ (B(x))$.

On remarquera que la première famille d'identités ne comporte pas de variables tandis que la seconde contient une seule occurrence de la même variable dans chaque membre, ainsi les identités sont-elles linéaires et régulières et ainsi l'extension d'une algèbre à l'ensemble de ses parties est encore une algèbre linéaire.

D'après les équations, on s'aperçoit que l'ordre d'application des opérations est indifférent, une notation $A_1.A_2 \ldots A_n.\ x.\ B_1 \ldots B_m$ en rend bien compte (cela correspond d'ailleurs à un lemme de forme normale).

2.3.- La conditionnelle.

Une conditionnelle sur un ensemble est une fonction

SI - ALORS - SINON - : (Bool, Base, Base) → Base qui vérifie les équations

SI VRAI ALORS x SINON y = x

SI FAUX ALORS x SINON y = y

On remarque que si l'on étend la conditionnelle sur les parties de l'ensemble par

SI b ALORS X SINON Y = {SI b ALORS x SINON y | x ∈ X, y ∈ Y}

les équations ne sont valables que si X et Y ne sont pas vides. En effet pour Y vide on a d'une part

SI VRAI ALORS X SINON Y = X

et d'autre part

SI VRAI ALORS X SINON Y = {x | x ∈ X y ∈ ∅ } = ∅

ce qui est contradictoire.

2.4.- Les arbres binaires.

On montrera que si une algèbre $\mathcal{A}$ vérifie les équations sur les arbres binaires, il en est de même de l'algèbre dont le support est $2^{Arb} - \{\emptyset\}$, $2^{Alph} - \{\emptyset\}$. Afin d'écrire plus simplement les

équations, posons :

CONS (x, a, y) = < x, a, y > , VID = Ψ

Ainsi on peut définir par une équation a point fixe, l'ensemble

X = << X, u, X > , v, < X , w, X >> ∪ {Ψ}

où u, v, w sont des éléments de Alph. Si l'on pose Y = GAU(X) et Z = DRO(X) , on constate que

Y = << Y, v, Z > , u, < Y, v, Z >> ∪ {Ψ}

Z = << Y, v, Z > , w, < Y, v, Z >> ∪ {Ψ}

en effet, puisque les équations restent vraies dans $2^{Arb} - \{\emptyset\}$, $2^{Val} - \{\emptyset\}$

Y = GAU(X) = GAU(≪ X, u, X > , v, < X, w, X >> ∪ {Ψ})

= ≪X, u, X > ∪ {Ψ} = ≪ GAU(X), TET(X), DRO(X) > , u, < GAU(X), TET(X), DRO(X) ≫ ∪ {Ψ}

= << Y, v, Z > , u, < Y, v, Z >> ∪ {Ψ}

3.- Equations linéaires et équations linéaires et régulières.

Les groupes , les algèbres linéaires et les conditionnelles permettent d'examiner trois familles d'équations. D'autres exemples sont proposés au paragraphe suivant et chapitre 3.

La première famille est celle des équations *linéaires* dont le paradigme est l'équation de la conditionnelle

SI VRAI ALORS x SINON g = x .

Elle ne contient qu'une occurrence au plus de chaque variable dans chaque membre (ici x et y seulement à gauche et x seulement à droite)

La deuxième famille est celle des équations *linéaires et régulières* représentée par l'équation

$$A(\bar{B}(x)) = \bar{B}(A(x))$$

Ces équations sont linéaires et l'ensemble des variables qui apparaissent à gauche (ici {x}) est identique à celui des variables qui apparaissent à droite.

La troisième famille est constituée des autres équations. Leur modèle est l'équation des groupes.

$$x \cdot x^{-1} = e$$

Les premières équations s'étendent par passage à l'ensemble des parties non vides, les secondes s'étendent par passage à l'ensemble des parties y compris la vide, les troisièmes ne s'étendent pas en général.

4.- LES_ENSEMBLES.

Voici à nouveau l'exemple du type abstrait : ensembles .
Nous l'avons déjà étudié au paragraphe 6 du chapitre 1 . Il servira de référence au cours de l'étude qui suit.
Rappelons comment il est défini :

type Ens [Alph]

fonctionnalité

VIDE ⟶ Ens ;

AJ : (Ens, Alph) ⟶ Ens ;

PR : (Ens, Alph) ⟶ Bool .

axiomatique

PR (VIDE, x) = FAUX

PR (AJ(m, x), y) = SI EQ(x, y) ALORS VRAI SINON PR(m, y)

soit $\mathcal{F}$ une algèbre de ENS [Alph, Bool] . On montrera dans la suite du chapitre que $\mathcal{F}'$ est un modèle des mêmes équations ; $\mathcal{F}'$ est caractérisée par $Ens_{\mathcal{F}'} = 2^{Ens_{\mathcal{F}}} - \{\emptyset\}$ $Alph_{\mathcal{F}'} = Alph_{\mathcal{F}}$ et $Bool_{\mathcal{F}'} = 2^{Bool_{\mathcal{F}}} - \{\emptyset\}$

$VIDE_{\mathcal{F}'} = \{VIDE_{\mathcal{F}}\}$

$AJ_{\mathcal{F}'}(mm, x) = \{AJ_{\mathcal{F}}(m, x) \ / \ m \in mm\}$ †

$VRAI_{\mathcal{F}'} = \{VRAI_{\mathcal{F}}\}$

$FAUX_{\mathcal{F}'} = \{FAUX_{\mathcal{F}}\}$

$SI\ bb\ ALORS\ xx\ SINON\ yy_{\mathcal{F}'} = \{SI\ b\ ALORS\ x\ SINON\ y_{\mathcal{F}} \mid b \in bb,\ x \in xx\ \ y \in yy\}$

Dans l'algèbre initiale (ou une autre) considérons l'ensemble algébrique $\ell\ell$ des éléments non vide E qui ne contiennent qu'un élément donné de Alph disons A . On en déduit aisément que $\ell\ell$ est solution de l'équation

$$\ell\ell = AJ_{\mathcal{F}'}(VIDE_{\mathcal{F}'}, A) \cup AJ_{\mathcal{F}'}(\ell\ell, A)$$

autrement dit que tout élément de $\ell\ell$ contient A , en effet utilisons l'équation et les identités. Auparavant , convenons de supprimer les indices $\mathcal{F}$ et $\mathcal{F}'$ aux opérations.

D'après l'équation

PR($\ell\ell$, A) = PR [AJ({VIDE} , A) U AJ($\ell\ell$, A), A]

par la distributivité de U

= PR(AJ ({VID} , A), A) U PR(AJ($\ell\ell$, A), A)

par l'équation de PR

= SI EQ (A, A) ALORS VRAI SINON PR ({VID} , A)
U PR (AJ($\ell\ell$, A), A)

† Nous conviendrons de noter par des lettres doublées les ensembles d'"ensembles".

or évidemment EQ(A, A) = VRAI donc

= {VRAI} U PR (AJ(ℓℓ , A), A)

par un calcul similaire ou précédent on montre

= {VRAI} U {VRAI} = {VRAI}

En fait, les équations se comportent bien vis à vis de la sorte Ens la suite de l'exemple montre qu'il n'en est pas de même vis à vis de la so Alph.

Supposons maintenant qu'on introduise une opération indéterministe EXT :(Ens)⟶ Alph . Calculons

PR (EXT(e), e) où e = AJ(AJ(VIDE, A), B)

Si on fait la preuve en examinant toutes les valeurs possibles de EXT(e) on obtient VRAI . Mais si tout d'abord on réduit PR(EXT(e), e)

SI EXT(e) = B ALORS VRAI SINON PR(EXT(e), AJ(VIDE, A))

et si l'on admet que EXT(e) n'a pas partout la même valeur en vertu de l'indéterminisme et de l'appel par nom on peut obtenir VRAI ou FAUX . Cela se rattache aux ℂ-extensions car EXT peut s'interpréter comme une opération déterministe Ens $\longrightarrow 2^{Alph}$.

5.- PRÉSENTATION INFORMELLE DES RÉSULTATS SUR LES EXTENSIONS.

Tout d'abord nous définirons les ℂT-extensions qui sont des extens à l'ensemble des parties non vides des opérations d'algèbres ; ces extensi ne se font pas sur toutes les sortes, mais seulement sur quelques unes qui forment l'ensemble T . Les ℙT-extensions sont des extensions à l'ensemb des parties y compris la partie vide. Avant d'énoncer ces définitions, il faut vérifier quelques conditions, appelées stabilité, qui permettent à ces définitions d'avoir un sens. (On aurait pu,sous l'hypothèse que T et T' sont disjoints , définir des ℂT- ℙT'-extensions ; cela aurait alourdi

inutilement les définitions sans que des exemples en justifient l'introduction).

Plus loin, nous examinons sous quelles conditions des identités vraies dans une algèbre $\mathfrak{A}$ s'étendent à sa ℂT-extension et à sa ℙT-extension. Ces conditions sont :

- une identité sur $\mathfrak{A}$ est vérifiée sur ℂT $\mathfrak{A}$, si elle est T-linéaire, c'est-à-dire si toute variable de sorte dans T apparait au plus une fois dans chaque membre.

- une identité sur $\mathfrak{A}$ est vérifiée sur ℙ T $\mathfrak{A}$ si elle est T-linéaire et T-régulière, c'est-à-dire si toute variable qui apparait dans un membre apparait une et seule fois dans l'autre.

Ensuite, nous définissons le concept de système d'équations. Cela nous amène à la notion de plus petit point fixe ou de solution.

6.- ℂ-EXTENSION D'UNE ALGÈBRE ET ℙ-EXTENSION .

6.1.- ENSEMBLES STABLES DE SORTES.

Soit $\mathfrak{A}$ une F-algèbre hétérogène dont l'ensemble des descripteurs de sortes est S .

Définition 1 :

Une partie T de S est dite F-stable (ou stable s'il n'y a pas d'ambiguité sur F) si pour toute opération f ∈ F de profil

$$So_1, \ldots, So_n \longrightarrow So$$

So ∈ T implique que pour tout i ∈ [1..n] $So_i \in T$ □

Autrement dit les éléments de So dans T ne sont obtenus qu'à partir d'éléments de So dans T et d'une opération f .

Exemple 1 :

Si F est l'ensemble des opérations {CONS, GAU, DRO, TET} de ARB seuls les ensembles {Arb, Val} et ∅ sont stables □

Exemple 2 :

Dans ENS les ensembles ∅ , {Alph} , {Ens, Bool, Ent} sont stables

Exemple 3 :

Dans les algèbres homogènes où il n'y a qu'une seule sorte S = {S_0}, S et ∅ sont stables. □

6.2.- $\mathbb{P}$-EXTENSION D'UNE ALGÈBRE.

Définition 2 :

Soit $\mathfrak{A}$ une algèbre avec S comme ensemble de descripteurs de sortes et F comme ensemble d'opérations, soit T un ensemble tel que S - T soit F - stable. On appelle $\mathbb{P}$T-extension une algèbre notée $\mathbb{P}T\mathfrak{A}$ telle que l'ensemble des sortes soit

$$So_{\mathbb{P}T\mathfrak{A}} = So_{\mathfrak{A}} \quad \text{si} \quad So \notin T$$

$$So_{\mathbb{P}T\mathfrak{A}} = 2^{So_{\mathfrak{A}}} \quad \text{si} \quad So \in T$$

et l'ensemble des opérations définies ainsi
soit $f : So_1, \ldots, So_n \longrightarrow So$ une opération
alors $f_{\mathbb{P}T\mathfrak{A}}(X_1, \ldots, X_n) = \{f_{\mathfrak{A}}(x_1, \ldots, x_n) / So_i \notin T \Rightarrow x_i = X_i$ et $So_i \in T \Rightarrow x_i \in X_i\}$ □

Puisque S - T est stable , on remarque que si $f : S_{o_1}, \ldots, So_n \rightarrow So$ avec $So \notin T$ alors $f_{\mathbb{P}T\mathfrak{A}}(x_1, \ldots, x_n) \in So_{\mathfrak{A}}$, ce qui montre que la définition est correcte vis à vis des sortes de S - T .

Exemple 4 :

Dans le paragraphe 4 , $\mathfrak{F}'$ est une $\mathbb{P}$ {Ens, Bool} -extension de $\mathfrak{F}$ en effet S - {Ens, Bool} = {Alph} est stable.

Remarque 1 :

Si T = ∅ , la $\mathbb{P}$T-extension obtenue est une fausse extension puisqu'aucune sorte n'est étendue. Dorénavant le lecteur pourra s'imaginer T non vide.

Remarque 2 :

On remarque que $f(X_1, \ldots, X_n) = \emptyset$ si et seulement si il existe $i \in \{1 \ldots n\}$ tel que $s_i \in T$ et $X_i = \emptyset$

Cette remarque permet d'introduire la définition 3

6.3.- $\mathbb{C}$-EXTENSION D'UNE ALGÈBRE.

Définition 3:

Si S - T est stable, une $\mathbb{C}$T-extension est une algèbre hétérogène dont les sortes sont

$$So_{\mathbb{C}T} = So \quad \text{si} \quad So \notin T$$

$$So_{\mathbb{C}T} = 2^{So} - \{\emptyset\} \quad \text{si} \quad So \in T$$

et où f est défini comme dans les $\mathbb{P}$T-extensions, compte-tenu de la remarque ci-dessus cela est bien correct.

Remarque 3 : conventions de notation :

Quand w est un mot de $P(F, X_\omega)$ on le note parfois aussi $w(x_1, \ldots$ où $x_1, \ldots, x_n$ appartiennent à $\bigcup_{s \in S} X^s_\omega$. Dans la suite, nous convien-drons que S est l'ensemble des sortes, T le sous-ensemble de S sur lequ se fait l'extension et que tout mot w sera noté $w(x_1, \ldots, x_m ; y_1, \ldots, y_p$ avec $x_1, \ldots, x_m$ tous différents appartenant à $\bigcup_{s \in T} X^s_\omega$ et $y_1, \ldots, y_p$ tous différents appartenant à $\bigcup_{s \notin T} X^s_\omega$.

7.- PROLONGEMENT DES IDENTITÉS ET VARIÉTÉS $\mathfrak{C}$-STABLES.

Le problème qui se pose est le suivant. Si $\mathcal{E}$ est une famille d'identi Si $\mathcal{A}$ une algèbre hétérogène totale modèle de $\mathcal{E}$ dans quel cas $\mathfrak{C}\, T\, \mathcal{A}$ est-elle un modèle de $\mathcal{E}$? Dans le paragraphe suivant on se posera la même question pour les $\mathbb{P}\, T$ extensions.

Pour cela nous avons besoin de définir la linéarité dans le cas des algèbres hétérogènes.

Définition 4 :

Soit $\mathcal{B}(F, X_\omega)$ une algèbre hétérogène libre, considérons l'algèbre $\mathcal{T} = < N, F_{\mathcal{T}} >$ définie ainsi $N = (\mathbb{N})_{s \in S}$ et si $f : So_1, \ldots So_n \longrightarrow So$ alors

$$f_{\mathcal{T}}(p_1, \ldots, p_n) = \sum_{i=1}^{n} p_n$$

par conséquent si $g : \longrightarrow So$ alors $g_{\mathcal{T}} = 0$. Les applications $\sigma^s_i \; X^s_\omega \longrightarrow \mathbb{N}$ telle que $\sigma^s_i(x^s_i) = 1$ et $\sigma^s_i(x^s_j) = 0$ si $i \neq j$ se

prolongent en un unique morphisme $oc : \mathcal{B}(F, X_\omega) \to \mathcal{T}$. $oc^s_i(w)$ s'appelle le *nombre d'occurrence* de x^s_i dans w .

Définition 5 :

Un mot w de $\mathcal{B}(F, X_\omega)$ est dit T-linéaire si pour tout $i \in \mathbb{N}$ et tout $s \in T$ $oc^s_i(w) \leq 1$ autrement dit chaque variable ce X^s , où $s \in T$, a au plus une occurrence dans w . □

Définition 6 :

Une identité $v = w$ est T-linéaire si v et w sont T-linéaires. □

Exemple 5 : cas du type abstrait Ens [Alph]

Considérons l'identité

PR(AJ (m, x), y) = SI EQ (x, y) ALORS VRAI SINON PR(m, y) ,

elle est T-linéaire, si T ne contient pas Alph , en effet y apparaît deux fois dans le membre de gauche □

Le *support* d'un mot est l'ensemble des variables efficaces de ce mot, c'est-à-dire l'ensemble des variables qui apparaissent dans la décomposition de ce mot en ses opérations élémentaires. Cet exposé s'attachant à définir le plus grand nombre de concepts possibles en termes d'algèbres, nous allons donner une définition formelle du support comme une interprétation dans une algèbre particulière composée d'ensemble de variables.

Définition 7 :

Considérons l'algèbre $\mathcal{B} = < B, F_{\mathcal{B}} >$ définie ainsi

- pour chaque $So \in S$, $So_{\mathcal{B}}$ est l'ensemble des parties finies de $\bigcup_{s \in S} X^s_\omega$. Ainsi toutes les sortes sont identiques et composées de tous

les ensembles finis de variables

- pour chaque f , $f(v_1,\ldots, v_n) = \bigcup_{1 \leqslant i \leqslant n} v_i$.

Les applications $\gamma^S : X^S_\omega \longrightarrow s_{\mathcal{B}}$ définies par $\gamma^S(x^S_i) = \{x^S_i\}$ se prolongent en un unique morphisme $\Gamma : \mathcal{P}(F, X_\omega) \longrightarrow \mathcal{B}$.

$\Gamma^S(w)$ est le *support* de w ; on le note plus simplement $\Gamma(w)$ □

Remarque 4 : Conventions de notation (suite).

Si l'on écrit $w(x_1, \ldots, x_m ; y_1, \ldots, y_p)$ on suppose que $\Gamma(w) \subseteq \{x_1, \ldots, x_m ; y_1, \ldots, y_p\}$

Lemme 1 :

$x^S_i \in \Gamma(w)$ implique $oc^S_i(w) \geqslant 1$

Démonstration :

Elle se fait par récurrence sur la complexité de w

si $comp(w) = 0$ alors $w = x^S_j$ et $\Gamma(w) = \gamma^S(x^S_j) = \{x^S_j\}$ donc

$x^S_i \in \Gamma(w) \Longleftrightarrow i = j \Leftrightarrow \sigma^S_i(x_j) = 1 \Longleftrightarrow oc^S_i(w) = 1 \Rightarrow oc^S_i(w) \geqslant 1$

si $comp(w) > 0$. alors $w = f(w_1, \ldots, w_n)$ alors $\Gamma(w) = \bigcup_{1 \leqslant k \leqslant n} \Gamma(w_k)$

donc $x^S_i \in \Gamma(w) \Longleftrightarrow (\exists k \in [1..n])\ x^S_i \in \Gamma(w_k) \Rightarrow (\exists k \in [1..n])\ oc^S_i(w_k) \geqslant 1$

or $oc^S_i(w) = oc^S_i(f(w_1, \ldots, w_n)) = f_{T\zeta}(oc_i(w), \ldots, oc^S_i(w_n))$

$$= \sum_{k=1}^{n} oc^S_i(w_k) \geqslant oc^S_i(w_k)$$

donc

$x^S_i \in \Gamma^S(w) \Rightarrow oc^S_i(w) \geqslant 1$

Lemme 2 :

si $w \in P(F, X_w)$ et $w = f(w_1, \ldots, w_n)$ et si w est T-linéaire alors $k \in \{1..n\}$ et $h \in \{1..n\}$ et $k \neq h$ impliquent

$$\Gamma(w_k) \cap \Gamma(w_h) \cap \bigcup_{t \in T} X^t_\omega = \emptyset$$

Démonstration :

Si $comp(w) = 0$ alors w n'est pas de la forme $f(w_1 \ldots, w_n)$, le lemme est donc démontré.

Si $comp(w) > 0$ supposons $k \in \{1..n\}$ et $h \in \{1..n\}$ et $h \neq k$ et $\Gamma(w_k) \cap \Gamma(w_h) \ni x^S_i$ pour $s \in T$. D'après le lemme 1, $oc^S_i(w_k) \geqslant 1$ et $oc^S_i(w_h) \geqslant 1$ donc

$$oc^S_i(w) = \sum_{j=1}^{n} oc^S_i(w_j) \geqslant oc^S_i(w_k) + oc^S_i(w_h) \geqslant 2$$

donc w n'est pas T-linéaire □

Le lemme suivant énonce les propriétés d'une opération, liées à la F-stabilité

Lemme 3 :

si S' est F-stable, si $s' \in S'$ et $w \in s'_{P(F,X_\omega)}$ alors

- $\Gamma(w) \subseteq \bigcup_{s \in S'} X^S_\omega$

- $oc^S_i(w) \geqslant 1 \Rightarrow s \in S'$

Démonstration :

Si $Comp(w) = 0$ alors $w = x^{S'}_i$ et $\Gamma(w) = \{x^{S'}_i\} \subseteq \bigcup_{s \in S'} X^S_\omega$

$oc^S_i(w) \geqslant 1 \Rightarrow s = s' \Rightarrow s \in S'$

Si $Comp(w) > 0$ alors il existe $f : (s1, \ldots, sn) \longrightarrow s'$

$w = f(w_1, \ldots, w_n)$ et donc $\Gamma(w) = \bigcup_{j=1}^{n} \Gamma(w_j)$ d'après la stabilité

s1, ..., sn appartiennent à S' et d'après l'hypothèse de récurrence

$\Gamma(w_j) \subseteq \bigcup_{s \in S'} X_\omega^s$ d'où $\Gamma(w) \subseteq \bigcup_{s \in S'} X_\omega^s$

$oc_i^s(w) = \sum_{j=1}^{n} oc_i^s(w_j)$ donc $oc_i^s(w) \geqslant 1$ implique qu'il existe

$j \in [1..n]$ tel que $oc_i^s(w) \geqslant 1$ donc par récurrence $s \in S'$.

La proposition qui suit est fondamentale.

Proposition 1 :

si w est T-linéaire alors

$w_{\mathbb{C}T\mathfrak{L}}(K_1, \ldots, X_m ; y_1, \ldots, y_p) = \{w_{\mathfrak{L}}(x_1,\ldots,x_m ; y_1,\ldots,y_p)(x_i \in X_i) \text{pour } i \in [1..m]\}$

Démonstration :

si $comp(w) = 0$ et $w = x_i^s$ alors dans l'algèbre $\mathbb{C}T\mathfrak{L}$ on a

$w_{\mathbb{C}T\mathfrak{L}} = proj_i^s$ c'est à dire

$w_{\mathbb{C}T\mathfrak{L}}(X_1,\ldots, X_m ; y_1, \ldots, y_p) = X_j = \{proj_i^s(X_1,\ldots,X_m ; y_1,\ldots,y_p)$

$(x_i \in X_i \text{ pour } i \in [1\ldots m]\}$

(où $x_j = x_i^s = = proj_i^s(x_1, \ldots, x_m ; y_1, \ldots, y_p)$) . La deuxième égalité provient du fait qu'aucun X_i n'est vide .

si $comp(w) > 0$ $w = f(w_1, \ldots, w_n)$ où $f : (s1, \ldots, sn) \longrightarrow \underline{s}$

montrons que si z appartient au membre de gauche, il appartient au membre de droite et vice versa.

$z \in w_{\mathbb{C}T\mathfrak{L}}(X_1, \ldots, X_m ; y_1, \ldots, y_p)$

ssi $z \in f_{\mathbb{C}T\mathfrak{L}}(w_{\mathbb{C}T\mathfrak{L}1}(X_1, \ldots, X_m ; y_1, \ldots, y_p), \ldots, w_{\mathbb{C}T\mathfrak{L}n}(X_1,\ldots,X_m ; y_1,\ldots,y_p))$

ssi $\exists(z_1, \ldots, z_n), z = f_{\mathfrak{L}}(z_1, \ldots, z_n)$ et

$\forall i \in [1..n]$ $z_i \in w_{\mathbb{C}T\mathfrak{L}i}(X_1, \ldots, X_m ; y_1, \ldots y_p)$

par l'hypothèse de récurrence

ssi $\exists(z_1, \ldots, z_n)$ $z = f_{\mathfrak{L}}(z_1, \ldots, z_n)$ et $\forall_i \in [1,..r]$ $\exists(x_1, \ldots, x_m)$

$z_i = w_{\mathfrak{L}i}(x_1, \ldots, x_m ; y_1, \ldots, y_p)$ et $\forall_j \in [1..m]$ $x_j \in X_j$

puisque w est T-linéaire, d'après le lemme 2, si $k \neq h$ une variable x_j qui apparaît dans w_k n'apparaît pas dans w_k , autrement dit on peut intervertir $\forall i \in [1..n]$ et $\exists(x_1, \ldots, x_m)$ i.e. le choix des x_j est indépendant de la composante i de f . Donc la dernière assertion ci-dessus est équivalente à

$\exists(z_1, \ldots, z_n)$ $z = f_{\mathfrak{L}}(z_1, \ldots, z_n)$ et $\exists(x_1, \ldots, x_m)$ $\forall i \in [1..n]$

$z_i = w_{\mathfrak{L}i}(x_1, \ldots, x_m ; y_1, \ldots, y_p)$ et $\forall j \in [1..m$ $x_j \in X_j$

ssi $\exists(x_1,\ldots, x_m)$ $\exists(z_1,\ldots, z_n), z = f_{\mathfrak{L}}(z_1,\ldots, z_n)$ et $\forall i \in [1..n]$

$z_i = w_{\mathfrak{L}i}(x_1,\ldots, x_m ; y_1,\ldots y_p)$ et $\forall j \in [1.. m]$ $x_j \in X_j$

ssi $\exists(x_1,\ldots, x_m)$ $z = w_{\mathfrak{L}}(x_1,\ldots, x_m ; y_1,\ldots, y_p)$ et $\forall_j \in [1..m]$ $x_j \in X_j$

ssi $z \in \{w_{\mathfrak{L}}(x_1,\ldots, x_m ; y_1,\ldots, y_p) / \forall_j \in [1..m]$ $x_j \in X_j\}$ □

Commentaires :

la proposition 1 est une propriété de confluence [HUE 77] ou de Church Rosser [ROS 73] . Si w est T-linéaire la valeur de w dans $\mathbb{C}T\mathcal{A}$ est indépendante de la manière dont on la calcule, qu'on la calcule en "parallèle" point par point et qu'on fasse la réunion des résultats obtenus ou qu'on calcule sur les ensembles, on trouve le même résultat. Par conséquent, cette proposition peut être rapprochée du lemme 11 de Huet [HUE 77] . D'autre part comme le remarque Shafaat [SHA 74] dans le cas des algèbres homogènes, elle est à la base des résultats qui vont suivre sur les variétés

Théorème 1 :

si $v \equiv w$ est une identité T-linéaire, si $v_{\mathcal{A}} = w_{\mathcal{A}}$ alors $v_{\mathbb{C}T\mathcal{A}} = w_{\mathbb{C}T\mathcal{A}}$

Démonstration :

D'après la proposition 1

$v_{\mathbb{C}T\mathcal{A}}(X_1, \ldots, X_m ; y_1, \ldots, y_p) = \{v_{\mathcal{A}}(x_1, \ldots, x_m ; y_1, \ldots, y_p) | x_i \in X_i$ pour i

et

$w_{\mathbb{C}T\mathcal{A}}(X_1, \ldots, X_m ; y_1, \ldots, y_p) = \{w_{\mathcal{A}}(x_1, \ldots, x_m ; y_1, \ldots, y_p) | x_i \in X_i$ pour $i \in [1..m]\}$

et puisque pour chaque $x_1, \ldots, x_m$ on a

$v_{\mathcal{A}}(x_1, \ldots, x_m ; y_1, \ldots, y_p) = w_{\mathcal{A}}(x_1, \ldots, x_m ; y_1, \ldots, y_p)$

alors

$v_{\mathbb{C}T\mathcal{A}}(X_1, \ldots, X_m ; y_1, \ldots, y_p) = w_{\mathbb{C}T\mathcal{A}}(X_1, \ldots, X_m ; y_1, \ldots, y_p)$ □

Définition 8 :

Une variété $\underline{V}$ d'algèbres hétérogènes est $\mathbb{C}T$-stable si pour toute algèbre $\mathcal{A}$ de $\underline{V}$, $\mathbb{C}T\mathcal{A}$ est une algèbre de $\underline{V}$.

Corollaire :

Soit $\underline{E}$ une famille d'identités T-linéaires alors la variété des algèbres modèles de $\underline{E}$ est $\mathbb{C}T$-stable.

Démonstration :

Soit $\mathcal{A}$ un modèle de $\underline{E}$ alors pour toute identité $v \equiv w$ de $\underline{E}$ on a $v_{\mathcal{A}} = w_{\mathcal{A}}$ donc puisque l'identité est linéaire $v_{\mathbb{C}T\mathcal{A}} = w_{\mathbb{C}T\mathcal{A}}$ donc $\mathbb{C}T\mathcal{A}$ est un modèle de $\underline{E}$ □

8.- VARIÉTÉS $\mathbb{P}$-STABLES.

Nous étudions dans ce paragraphe le problème du prolongement des identités d'une algèbre, à la $\mathbb{P}$-extension.

Proposition 2 :

si v est T-linéaire et si $\Gamma(v) \cap \bigcup_{t \in T} X_\omega^t = \{x_1, \ldots, x_m\}$

alors $v_{\mathbb{P}T\mathcal{A}}(X_1, \ldots, X_m ; y_1, \ldots, y_p) = \{v_{\mathcal{A}}(x_1, \ldots, x_m ; y_1, \ldots, y_p) / x_i \in X_i$ pour $i \in [1..n]\}$

Démonstration :

si pour tout $i \in [1..m]$, $X_i \neq \emptyset$, alors le résultat est vrai, d'après la proposition 1. Si l'un des X_i est vide, le deuxième membre de l'égalité est vide . Il reste à montrer qu'alors $v_{IPT\mathcal{A}}(X_1,\ldots, X_m ; y_1,\ldots,$ est vide, en effet par récurrence :

si $v = x_i^s$ alors $m = 1$ et $X_1 = \emptyset$ ainsi $v_{IPT\mathcal{A}}(X_1 ; y_1,\ldots, y_p) = X_1$

si $comp(v) > 0$ alors $v = f(v_1,\ldots, v_n)$ et d'après la définition de Γ

$\Gamma(v) \cap \bigcup_{t\in T} X_\omega^t = \bigcup_{j=1}^{n} \Gamma(v_j) \cap \bigcup_{t\in T} X_\omega^t$. Soit j tel que

$x_i \in \Gamma(v_j)$ alors $v_j(x'_1,\ldots, x'_q ; y_1,\ldots, y_p)$ où

$\{x_1,\ldots, x'_q\} = \Gamma(v_j) \cap \bigcup_{t\in T} X_\omega^t$ alors $v_{j\,IPT\mathcal{A}}$ prend la valeur $\emptyset$

sur les valeurs $X'_1,\ldots, X'_q$ correspondant au vecteur $X_1,\ldots, X_m$ et ainsi

$f_{IPT\mathcal{A}}(\ldots, v_j(X'_1,\ldots, X'_q ; y_1,\ldots, y_p),\ldots) = \emptyset$ □

Nous définissons la régularité dans le cas des algèbres hétérogènes.

Définition 9 :

Une identité $v = w$ est T-régulière si

$$\Gamma(v) \cap \bigcup_{t\in T} X_\omega^t = \Gamma(w) \cap \bigcup_{t\in T} X_\omega^t$$

Exemple 6 : cas de la table des symboles. (cf. chapitre 4 paragraphe 4).

L'équation

LEAVEBLOCK (ADDID (symbtab, id, attlist)) = LEAVE (symbtab) est T-régulière si $T = \{symboltable, boolean\}$, mais n'est pas T-régulière si T contient Identifier ou Attributlist. □

Théorème 2 :

si $v = w$ est une identité T-linéaire et T-régulière, si $v_{\mathcal{A}} = w_{\mathcal{A}}$ alors $v_{IPT\mathcal{A}} = w_{IPT\mathcal{A}}$

Démonstration :

Puisque $\Gamma(v) \cap \bigcup_{t\in T} X_\omega^t = \Gamma(w) \cap \bigcup_{t\in T} X_\omega^t$ posons

$\Gamma(v) \cap \bigcup_{t\in T} X_\omega^t = \{x_1, \ldots, x_m\}$ alors d'après la proposition 2

$v_{IPT\mathcal{A}}(X_1,\ldots, X_m ; y_1, \ldots, y_p) = \{v_{\mathcal{A}}(x_1,\ldots, x_m ; y_1,\ldots, y_p)/x_i \in X_i$

pour $i \in [1..n]$)}

et de même pour w

$w_{IPT\mathcal{A}}(X_1,\ldots, X_m ; y_1,\ldots, y_p) = \{w_{\mathcal{A}}(x_1,\ldots, x_m ; y_1,\ldots, y_p)/x_i \in X_i$

pour $i \in [1..m]$)

ainsi puisque $v_{\mathcal{A}} = w_{\mathcal{A}}$ on a $v_{IPT\mathcal{A}} = w_{IPT\mathcal{A}}$ □

Définition 10 :

Une variété $\underline{V}$ est IPT-stable si pour toute algèbre $\mathcal{A}$ de $\underline{V}$, $IPT\mathcal{A}$ appartient à V .

Corollaire :

si $\underline{E}$ est une famille d'identités T-linéaires et T-régulières alors la variété des algèbres modèles de $\underline{E}$ est IPT-stable.

9.- ENSEMBLES ALGÉBRIQUES.

Dans ce paragraphe, nous nous placerons dans une variété CT-stable, notée $\underline{V}$. Soit X_n l'ensemble des variables

$$\bigcup_{s \in T} \{x_1^s, \ldots, x_{n_s}^s\} \text{ alors } n = (n_s)_{s \in S} \text{ où } n_s = 0 \text{ si } s \notin T .$$

Un système de n équations est une application E qui à chaque variable x_i^s de X_n fait correspondre un ensemble d'éléments noté E_i^s ou $E[x_i^s]$ inclus dans ${}_s\mathcal{F}(\underline{V}, X_n)$ i.e. dans la sorte s de l'algèbre libre sur X_n , dans la variété $\underline{V}$, souvent on notera le système

$$x_i^s = E_i^s \ (x_1^{s1}, \ldots, x_{n_{s1}}^{s1}, \ldots, x_j^{si}, \ldots)$$

Exemple 7 :

Dans la variété ARB [Alph] posons $Y = x_1^{Arb}$, $Z = x_2^{Arb}$ alors

$$Y = \langle\langle Y, v, Z \rangle, u, \langle Y, v, 2 \rangle\rangle$$

$$Z = \langle\langle Y, v, 2 \rangle, w, \langle Y, v, 2 \rangle\rangle$$

est un système de 2 équations □

Exemple 8 :

Dans la variété des algèbres {A, B} - linéaires, posons $x = x_1^{Lin}$ alors

$$x = A.A.x.B \ \cup \ A.A.B$$

est un système d'une équation. On notera que l'absence de parenthèse dans la notation des éléments de E met en évidence le fait qu'on se place dans $\mathcal{F}(\underline{LIN}\ [Alph\ ], X_1)$ et non pas dans $\mathcal{F}(\underline{ALG}[VID, Alph \cup \overline{Alph}], X_1)$. Il ne s'agit pas de l'équation

$$x = A(\bar{B}(A(x))) \cup A(A(B(VID))$$

ni de l'équation

$$x = A(A(\bar{B}(x))) \cup A(\bar{B}(\bar{A}(VID)) \qquad \square$$

Rappelons tout d'abord que si $\mathcal{A}$ est une algèbre et si $w \in {}_s\mathcal{F}(\underline{V}, X_n)$ on note $w_{\mathcal{A}}$ l'élément de $\mathcal{F}^{(n)}(\underline{V}, \mathcal{A})$ correspondant (cf. le paragraphe 4.5 du chapitre 1). L'application définit alors une application, notée aussi E_i^s, de $\prod_{s \in T} {}_s\mathcal{A}^{n_s}$ vers ${}_s\mathcal{A}$, ainsi

$$E_i^s(y) = \{ w_{\mathcal{A}}(y) \ / \ w \in E_i^s \}$$

Exemple 9 :

Si dans l'exemple 7 on pose $y = (VID, \langle VID, t, VID \rangle)$ on obtient :

$E[Y](y) = \{\langle\langle VID, v, \langle VID, t, VID\rangle\rangle, u, \langle VID, v, \langle VID, t, VID\rangle\rangle\rangle, VID\}$ □

Par réunion, on peut étendre E_i^s en une application de $\prod_{s \in T} (2^{S\mathcal{A}} - \emptyset)^{n_s}$ vers $2^S - \emptyset$ en posant

$$E_i^s(Y) = \{E_i^s(y) \ / \ y \in Y\}$$

On note $E(Y)$ le uple $(E_1^{s1}(Y), \ldots, E_{n_{s1}}^{s1}(Y), \ldots, E_j^i(Y), \ldots)$

Un point fixe ou solution de E est un uple Y d'ensembles tel que

$$Y = E(Y)$$

Proposition 3 :

E est une application U-continue de $\prod_{s \in T} (2^s - \emptyset)^{n_s}$ dans lui-même

Démonstration :

La définition de E_i^s par

$$E_i^s(Y) = \bigcup_{y \in Y} E_i^s(\{y\})$$

fait naturellement de E une fonction U-continue □

9.1.- SOLUTION DANS LE CAS DES VARIÉTÉS ℙ-STABLES .

Dans le cas où la variété est ℙ-stable, E s'étend de $\prod_{s \in T} (2^s)^{n_s}$ dans lui-même ; ainsi $E(\emptyset)$ a un sens et $\emptyset \subset E(\emptyset)$ en itérant cette application on obtient la suite $\emptyset \subset \ldots \subset E^i(\emptyset) \subset \ldots$ qui converge on a pour borne supérieure ou réunion le plus petit point fixe de l'équation $Y = E(Y)$.

Exemple 10 :

Dans le type Ens [Ent [Bool] , Bool] du paragraphe 2.4, l'équation

$\ell\ell$ = AJ(VID, SU.SU.ZERO) U AJ($\ell\ell$, SU.SU.ZERO)

peut être construit par itération à partir de $\emptyset$

$\ell\ell_0 = 0$, $\ell\ell_1 =$ { AJ(VID, SU.SU.ZERO)} ,

$\ell\ell_2 =$ { AJ(VID, SU.SU.ZERO), AJ(AJ(VID, SU.SU.ZERO), SU.SU.ZERO)}

etc... □

9.2.- SOLUTION D'UN SYSTÈME ℂ-ACCEPTABLE DANS LE CAS DES VARIÉTÉS ℂ-STABLES.

Puisque la variété est ℂ-stable, E ne s'étend pas forcément à $\emptyset$ ainsi la construction du plus petit point fixe ne peut débuter sur $\emptyset$. Mais nous allons imposer à E une propriété que vérifient la plupart des systèmes d'équations couramment utilisés.

Définition 11 :

Un système est ℂ-acceptable si pour chacune de ses variables x_i^s $E_i^s \cap \mathcal{P}(\underline{V})$ est non vide □ .

Autrement dit tout membre droit contient des constantes.

Notons B_i^s l'ensemble $E_i^s \cap \mathcal{P}(\underline{V})$ et B l'uple $(B_1^{11}, \ldots, B_j^{si}, \ldots)$ sa valeur dans $\mathcal{L}$ est notée $B_{\mathcal{L}}$.

Lemme 1 :

Pour tout uple X dans $\mathcal{L}^n$, $E_{\mathcal{L}}(X) \supseteq B_{\mathcal{L}}$

Démonstration :

elle découle directement de la définition de $B_{\mathcal{L}}$ □

Lemme 2 :

Si Y est un point fixe de $\mathcal{L}$, $Y \supseteq B_{\mathcal{L}}$.

Démonstration :

si Y est un point fixe $Y \supseteq E(Y)$ or d'après le lemme 1 , $E(Y) \supseteq B_{\mathcal{L}}$ □

On itère $E_{\mathcal{L}}$ sur $B_{\mathcal{L}}$ et on obtient la proposition suivante :

Proposition 4 :

$\bigcup_{n \geq 0} E^n_{\mathcal{L}} (B_{\mathcal{L}})$ est le plus petit point fixe de E

Démonstration :

Notons que, si Y est un point fixe, $\bigcup_{n \geq 0} E^n_{\mathcal{L}} (B_{\mathcal{L}}) \subseteq Y$ en effet

d'après le lemme 2, $B_{\mathcal{L}} \subseteq Y$ et donc par récurrence

si $E^n_{\mathcal{L}} (B_{\mathcal{L}}) \subseteq Y$ alors $E^{n+1}_{\mathcal{L}} (B_{\mathcal{L}}) \subseteq E(Y) \subseteq Y$

et donc la réunion des $E^n_{\mathcal{L}} (B_{\mathcal{L}})$ est incluse dans Y .

Réciproquement, par continuité de $E_{\mathcal{L}}$, on voit que

$$E_{\mathcal{L}} \left(\bigcup_{n \geq 0} E^n_{\mathcal{L}} (B_{\mathcal{L}}) \right) = \bigcup_{n \geq 1} E^n_{\mathcal{L}} (B_{\mathcal{L}})$$

or d'après le lemme 1

$E^n_{\mathcal{L}} (B_{\mathcal{L}}) \supseteq B_{\mathcal{L}}$ donc $\bigcup_{n \geq 1} E^n_{\mathcal{L}} (B_{\mathcal{L}}) = \bigcup_{n \geq 0} E^n_{\mathcal{L}} (B_{\mathcal{L}})$ □

10.- BIBLIOGRAPHIE.

Le calcul sur les parties d'un groupe, appelées complexes, remonte au début de ce siècle, il était reconnu que ces complexes ne formaient pas un groupe mais un monoïde [VDW 36] . Tous les travaux que nous allons citer portent sur les algèbres homogènes, aucune étude n'a été faite sur les algèbres hétérogènes. Le plus ancien article sur les possibilités de $\mathbb{P}$-extension est dû à Gautam en 57 [GAU 57] . Fuchs [FUC 65] pose le problème de l'extension des équations à des algèbres ordonnées. L'étude la plus complète est certainement celle de Bleichner, Schneider et Wilson [BLE 73] . Les $\mathbb{C}$-extensions

apparaissent comme le cas où l'algèbre à étendre est trivialement ordonnée, mais les techniques de démonstration ne sont pas fondamentalement différentes. Wand dans sa thèse [WAN 73] caractérise, indépendamment des autres travaux, les variétés $\mathbb{C}$-stables et $\mathbb{P}$-stables. Shafaat [SHA 74] quant à lui, expose une réciproque assez forte du corollaire du théorème 2 sous la forme suivante : "si une variété $\mathbb{P}$-stable est modèle d'un ensemble d'équations Σ , elle est modèle d'un sous-ensemble de Σ d'équations linéaires et régulières".

Huet [HUE 77] se place dans les systèmes de réécritures homogènes et par conséquent utilise une modélisation spécifique légèrement différente , il démontre des résultats de congruence des réécritures qui sont proches de la $\mathbb{C}$-stabilité (confluence de réécritures en parallèle) et font appel à la linéarité. Une règle de réécriture est , par essence, asymétrique, par conséquent la linéarité n'est nécessaire qu'à gauche dans le cas des réécritures parallèles.

CHAPITRE IV

exemples de types abstraits

CHAPITRE IV

EXEMPLES DE TYPES ABSTRAITS

I.- Mots.

Nous allons examiner des algèbres et des opérations dans ces algèbres, destinées à exprimer les objets que l'on appelle : mots, piles, listes, files etc.... Ces algèbres se classent en deux genres. Le premier correspond à des algèbres n'ayant que deux sortes : les mots et les booléens. Le second à des algèbres à trois sortes : les mots, l'alphabet, les booléens.

1.1.- ALGÈBRES À DEUX SORTES : MOTS ET BOOLÉENS [LES 78]

La figure 1 donne les opérations utilisées .

Algèbres monadiques :

Elles comportent les opérations ε , $Alph^9$ et EQ et vérifient

MONAD
$$\begin{cases} EQ(\varepsilon, \varepsilon) = VRAI \\ EQ(A.x, \varepsilon) = EQ(\varepsilon, A.x) = FAUX & \text{pour chaque } A \in Alph \\ EQ(A.x, B.y) = FAUX & \text{pour chaque } A, B \in Alph \text{ et } A \neq B \\ EQ(A.x, A.y) = EQ(x.y) & \text{pour chaque } A \in Alph \end{cases}$$

$< Alph^*, Bool ; \varepsilon, Alph^9 >$ vérifie MONAD.

Notation de l'opération	Commentaire	Notation de la famille d'opérations	Profil	Interprétation dans Alph*
ε	unique		()→Mot	le mot vide
A	pour chaque A ∈ Alph	Alph	()→Mot	A est le mot réduit à A
A.x	pour chaque A ∈ Alph	$Alph^g$	(Mot)→Mot	A.α est le mot constitué de A suivi des lettres de α
*	unique		(Mot,Mot)→Mot	concaténation
A(x, y)	pour chaque A ∈ Alph	$Alph^\diamond$	(Mot,Mot)→Mot	A(α , β) est A.(α*β)
x . A	"	$Alph^\lrcorner$	(Mot) → Mot	α.A est le mot constitué de α suivi de la lettre A
EQ	unique		(Mot,Mot)→Bool	égalité

Figure 1 : Quelques opérations sur les mots.

Le type abstrait construit avec les opérations ε, $Alph^g$ et EQ et l'axiomatique MONAD vérifie la propriété de complète discrimination, il n'y a par conséquent qu'une algèbre à un isomorphisme près. Les ensembles algébriques c'est à dire les langages associés sont les réguliers.

Monoïdes sur Alph

Ils comportent les opérations ε, Alph et * et vérifient

MONOIDE
$$\begin{cases} x\,(y*z) = (x*y)*z \\ x*\varepsilon = x \\ \varepsilon*x = x \\ EQ\,(\varepsilon,\varepsilon) = VRAI \\ EQ\,(A*x, B*y) = FAUX \quad si\ A \neq B \\ EQ\,(A*x, A*y) = EQ(x,y) \quad pour\ chaque\ A \in Alph \end{cases}$$

< Alph* , Bool ; ε U Alph, *, EQ > vérifie MONOIDE

Le type abstrait construit avec les opérations ε U Alph, * et l'axiomatique MONOIDE vérifie la propriété de complète discrimination. Les langages sont les C-langages.

Algèbres dyadiques de Greibach

Les algèbres dyadiques de Greibach comportent les opérations ε, $Alph^\diamond$ et EQ et vérifient

GRB
$$\begin{cases} A(\varepsilon, B(x,y)) = A(B(x,y),\varepsilon) \\ A(B(x,\varepsilon),y) = A(B(x,y),\varepsilon) \\ EQ(A(x,\varepsilon),A(y,\varepsilon)) = EQ(x,y) \\ EQ(\varepsilon,\varepsilon) = VRAI \\ EQ(A(x,y),B(z,t)) = FAUX \quad pour\ A \neq B \end{cases}$$

Remarquons a propos de ces équations que l'on peut montrer que le système de réécriture associée est confluent et noethérien, bien que l'on ne puisse appliquer le critère de Musser à la seconde. D'autre part on voit que si l'on avait remplacé $A(\varepsilon, B(x, y)) = A(B(x, y), \varepsilon)$ par $A(x, \varepsilon) = A(\varepsilon, x)$ on aurait perdu la propriété de terminaison finie (il suffit de poser $x = \varepsilon$) . Enfin la première équation aurait pu être remplacée par

$$A(\varepsilon, B(x, y)) = A(B(\varepsilon, x), y)$$

ou $$A(\varepsilon, B(x, y)) = A(B(x, \varepsilon), y)$$

La seconde ci-dessus donnant facilement par composition avec la deuxième de GRB celle déjà citée.

$< Alph^* , Bool ; \varepsilon, Alph^{\diamond}, EQ >$ est une algèbre dyadique de Greibach.

Les formes normales sont les

$$A_1(A_2(\dots (A_{n-1}(\varepsilon, \varepsilon), \varepsilon) \dots \varepsilon), \varepsilon)$$

Les langages associés sont les C-langages.

Algèbres linéaires

Les algèbres linéaires comportent les opérations ε, $Alph^g$, $Alph^d$ et EQ et vérifient

LIN
$$\begin{cases} (A.x)_{\circ} B = A. (x \circ B) \\ \varepsilon \circ A = A.\varepsilon \\ EQ(A.x, A.y) = EQ(x, y) \\ EQ(A.x, B.y) = FAUX \qquad \text{pour} \quad A \neq B \\ EQ(\varepsilon, \varepsilon) = VRAI \end{cases}$$

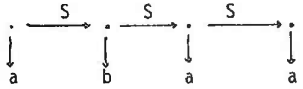
$< Alph^*, Bool ; \varepsilon, Alph^g \cup Alph^d, EQ >$ est une algèbre linéaire

Les formes normales sont les $A_1.A_2 \dots A_n.\varepsilon$

Les langages associés sont les langages linéaires.

Représentation :

Les représentations consistent pour chaque mot en une algèbre contenant un accès de succession et un accès de valeur, par exemple le mot abac est représenté par l'algèbre



Les représentations des opérations sont assez simples (voir par exemple la figure 2).

1.2.- ALGÈBRES À TROIS SORTES : MOTS, ALPHABET ET BOOLÉENS.

1.2.1.- SPÉCIFICATION DU TYPE LISTLIN

Nous ne décrivons qu'un type Listlin qui contient la plupart des opérations qu'on y étudie habituellement. Les constructeurs sont ε, . . Nous prenons tout de suite des notations infixées plus lisibles.

type Listlin [Alph]
fonctionnalité

ε : () → Listlin
· : (Alph, Listlin) → Listlin
TETE : (Listlin) → Alph U {INDEF}
RESTE : (Listlin) → Listlin
* : (Listlin, Listlin) → Listlin
$\circ$: (Listlin, Alph) → Listlin
QUEUE : (Listlin) → Alph U {INDEF}
DEBUT : (Listlin) → Listlin
RENV : (Listlin) → Listlin
ESTVIDE : (Listlin) → Bool

Axiomatique

{T1} TETE(ε) = INDEF
{T2} TETE(a.x) = a
{T3} RESTE(ε) = ε
{R2} RESTE(a.x) = x
{*1} ε * x = x
{*2} (a.x)* y = a. (x * y)
{⊙1} ε ⊙ a = a.ε
{⊙2} (a.x)⊙b = a.(x⊙ b)
{Q1} QUEUE(ε) = INDEF
{Q2} QUEUE(a.ε) = a
{Q3} QUEUE(a.(b.x)) = QUEUE(b.x)
{D1} DEBUT(ε) = ε
{D2} DEBUT(a.ε) = ε
{D3} DEBUT(a.(b.x)) = a. DEBUT(b.x)
{RV1} RENV (ε) = ε
{RV2} RENV (a.x) = RENV (x)⊙a
{V1} ESTVIDE(ε) = VRAI
{V2} ESTVIDE(a.x) = FAUX

1.2.2.- Propriétés du type Lislin

Voici quelques résultats concernant ce type abstrait.

Lemme 1 :

Le système de réécriture associé est noethérien

Démonstration :

Le critère de Musser s'applique. Par exemple dans la règle

(a.x) * y → a.(x * y)

* réduit . et n'apparaît pas auparavant en partie droite. Notons que le critère proposé par Plaisted s'applique aussi à condition de prendre un ordre

sur les symboles fonctionnels tel que . soit plus petit que tous les autres symboles à l'exception éventuellement de ε et que ⊙ soit plus petit que RENV, par exemple l'ordre dans lequel ces symboles sont introduits. □

Lemme 2 :

Le type abstrait est confluent

Démonstration :

Aucune superposition n'est possible □

Lemme 3 de forme normale :

Les formes normales sont ε ou a.ℓ

Démonstration :

Par récurrence sur la taille des termes de profil () → Listlin.

On voit tout de suite en examinant les règles que la forme normale d'un terme a une taille plus petite ou égale à ce terme.

- si taille (ℓ) = 1 alors ℓ = ε et le résultat est évident
- si taille (ℓ) > 1 , il suffit de procéder par cas suivant le premier symbole de ℓ .

Considérons par exemple le cas où ℓ = DEBUT(ℓ') .
Par récurrence sur la taille, ℓ' a une forme normale. Si la forme normale de ℓ' est ε ou a.ε , la forme normale de ℓ' est ε
Si la forme normale de ℓ' est a.(b. ℓ"), alors ℓ se réécrit DEBUT (a.(b.ℓ")) qui se réécrit a.DEBUT(b.ℓ")
or

taille(DEBUT(b.ℓ")) < taille(DEBUT(a.(b.ℓ"))

d'après la remarque sur la taille des formes normales

≤ taille(DEBUT(ℓ')) = taille (ℓ)

et par récurrence, DEBUT(b.ℓ") a une forme normale, disons ℓ'" , alors ℓ a une forme normale a. ℓ'" □

Lemme 4 :

Le type abstrait est suffisamment complet

Démonstration :

Les termes à considérer ont un profil $() \rightarrow$ Alph U {INDEF} , il suffit de considérer les termes de la forme TETE(ℓ) ou QUEUE(ℓ) et de raisonner sur la taille de ℓ en supposant ℓ sous forme normale .

1) si $\ell = \varepsilon$ alors TETE(ε) et QUEUE(ε) se réduisent à INDEF

2) si $\ell = a.x$ alors TETE(ℓ) se réécrit en a et pour QUEUE, il faut considérer deux cas

2a) $\ell = a.\varepsilon$ alors QUEUE se réécrit en a

2b) $\ell = a.(b.\ell')$ alors

QUEUE(ℓ) $\rightarrow$ QUEUE($b.\ell'$) qui par récurrence se réécrit en termes de Alph U {INDEF} . □

Voici quelques formules vraies sur ce type et que l'on peut démontrer par récurrence (ce sont des formules de la théorie Ind(Listlin) au sens de Burstall et Goguen [BUR 77]). Cette liste ne se veut évidemment pas exhaustive.

RESTE$((x \odot a) \odot b)$ = RESTE $(x \odot a)$

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

$x * (a.y) = (x \odot a) * y$

$x * (y \odot b) = (x * y) \odot b$

QUEUE$(x \odot b)$ = b

DEBUT$(x \odot b)$ = x

RENV(RENV(x)) = x

RENV$(x \odot a)$ = a.RENV(x)

RENV$(x * y)$ = RENV(y)* RENV(y)

Démontrons par exemple

Lemme 5 :

$x * (a.y) = (x \odot a) * y$

Démonstration :

Par récurrence sur x

si $x = \varepsilon$, cela se démontre par des méthodes habituelles de réécriture,

$$\varepsilon * (a.y) \xrightarrow{*1} a.y$$

et

$$(\varepsilon \odot a) * y \xrightarrow{\odot 1} (a.\varepsilon) * y \xrightarrow{*2} a.(\varepsilon * y) \xrightarrow{*1} a.y$$

si $x = b.z$

$$(b.z) * (a.y) \xrightarrow{*2} b.(z * (a.y))$$

et

$$((b.z) \odot a) * y \xrightarrow{\odot 2} (b.(z \odot a)) * y \xrightarrow{*2} b.((z \odot a) * y)$$

or par récurrence $z * (a.y) = (z \odot a) * y$ d'où le résultat □

Ce type de démonstration peut être automatisé en s'inspirant des travaux de Boyer et Moore [BOY 75] [BOY 77] .

Signalons une autre démonstration suggérée par MUSSER [MUS 78] et facilement automatisable. Elle consiste à utiliser la méthode proposée par Knuth et Bendix pour compléter un système qui n'est pas confluent. Elle suppose le type abstrait, ainsi que ses types paramètres, complètement définis par des systèmes de réécritures confluents, noethériens et suffisamment complets. On ajoute la propriété à démontrer comme une nouvelle règle et l'on tente de compléter le système ; pour chaque paire de termes, issus d'un même terme, obtenus par superposition et contredisant la confluence locale, on ajoute une nouvelle règle formée à partir de la paire de termes irréductibles déduits de ces termes.

Trois solutions sont possibles :

- en itérant le processus on engendre la règle VRAI → FAUX , alors la propriété est fausse
- en itérant le processus , on aboutit à un système confluent sans engendrer la règle VRAI → FAUX alors la propriété est vraie

- le processus peut être répété sans s'arrêter, alors la propriété ne peut être démontrée (du moins pas par cette méthode)

Pour cette équation, on introduit la règle

$$x * (a.y) \xrightarrow{?} (x \odot a) * y$$

elle se superpose avec la règle {*2} pour donner le terme $(a.x) * (b.y)$.

On obtient

$$(a.x) * (b.y) \xrightarrow{*2} a.(x*(b.y)) \xrightarrow{?} a.((x \odot b) * y)$$

et d'autre part

$$(a.x) * (b.y) \xrightarrow{?} ((a.x) \odot b) * y \xrightarrow{\odot 2} (a.(x \odot b)) * y \xrightarrow{*2} a.((x \odot b) * y)$$

La confluence locale n'est pas contredite.

Une deuxième superposition est possible avec la règle {*1} pour donner $\varepsilon * (a.y)$ qui fournit :

$$\varepsilon * (a.y) \xrightarrow{*1} a.y$$

et

$$\varepsilon * (a.y) \xrightarrow{?} (\varepsilon \quad a) * y \xrightarrow{*2} a.(\varepsilon * y) \xrightarrow{*1} a.y$$

La confluence locale n'est pas non plus contredite et il n'y a plus d'autres superpositions. La proposition est vraie.

Examinons ce que cela donne sur une proposition fausse

RESTE (DEBUT(x)) = x

En superposant avec la règle D2 on obtient

RESTE (DEBUT(a.ε)) → a.ε d'après la nouvelle règle

et

$$\text{RESTE (DEBUT(a.}\varepsilon\text{))} \xrightarrow{D2} \text{RESTE}(\varepsilon) \xrightarrow{R1} \varepsilon$$

et ainsi on introduit la nouvelle règle $a.\varepsilon \rightarrow \varepsilon$ qui se superpose avec {V2} pour donner

$$\text{ESTVIDE}(a.\varepsilon) \xrightarrow{V2} \text{FAUX}$$

et

$$\text{ESTVIDE}(a.\varepsilon) \rightarrow \text{ESTVIDE}(\varepsilon) \xrightarrow{V1} \text{VRAI}$$

D'où le résultat.

1.2.3.- UNE REPRÉSENTATION DU TYPE LISTE.

Voici un exemple de représentation des listes linéaires. Les algèbres filles possèdent deux sortes : Place, Alph et cinq accès

T : () → Place
Q : () → Place
S : (Place) → Place
P : (Place) → Place
V : (Place) → Alph

et vérifient

SoP(x) = x et PoS(x) = x

Les opérations Vid, . , ⊙, Reste, Début et Fenêtre sont données par la figure 2 et

$\text{Tête}(\mathfrak{a}) = V_{\mathfrak{a}}(T_{\mathfrak{a}})$ si $V_{\mathfrak{a}}(T_{\mathfrak{a}})$ est défini

et

$\text{Tête}(\mathfrak{a}) = \text{INDEF}$ si $V_{\mathfrak{a}}(T_{\mathfrak{a}})$ n'est pas défini

$\text{Queue}(\mathfrak{a}) = V_{\mathfrak{a}}(Q_{\mathfrak{a}})$ si $V_{\mathfrak{a}}(Q_{\mathfrak{a}})$ est défini

et

$\text{Queue}(\mathfrak{a}) = \text{INDEF}$ si $V_{\mathfrak{a}}(Q_{\mathfrak{a}})$ n'est pas défini

$\text{Estvide}(\mathfrak{a}) = \text{VRAI}$ si et seulement si $\text{Place}_{\mathfrak{a}} = \emptyset$

Cette représentation vérifie les équations, par exemple

Lemme 6 :

$$\text{Début}(a.(b.\mathfrak{a})) \sim a.\ \text{Début}(b.\mathfrak{a})$$

Démonstration :

$$\text{Place}_{\text{Début}(a.(b.\mathfrak{a}))} = \text{Place}_{a.(b.\mathfrak{a})} - \{ Q_{\text{Début}(a.(b.\mathfrak{a}))} \}$$

$$= \text{Place}_{b.\mathfrak{a}} \cup \{ \text{Place}_{b.a} \} - \{ P_{a.(b.\mathfrak{a})}(Q_{a.(b.\mathfrak{a})}) \},$$

$= Place_{b.\mathfrak{a}} \cup \{Place_{b.\mathfrak{a}}\} - \{P_{b.\mathfrak{a}}(Q_{b.\mathfrak{a}})\}$

$= Place_{Début(b.\mathfrak{a})} \cup \{Place_{b.\mathfrak{a}}\}$

et posant $\varphi(Place_{b.\mathfrak{a}}) = \varphi(Place_{Début(b.\mathfrak{a})})$

et $\varphi(x) = x$ pour les autres valeurs de x .

On définit un isomorphisme entre $Place_{Début(a.(b.\mathfrak{a}))}$ et

$Place_{a.Début(b.\mathfrak{a})} = Place_{Début(b.\mathfrak{a})} \cup \{Place_{Début(b.\mathfrak{a})}\}$

On remarque $Q_{Début(a.\mathfrak{A})} = Q_{a.Début(\mathfrak{A})}$ où l'on a posé $\mathfrak{B} = b.\mathfrak{a}$

en effet

$Q_{Début(a.\mathfrak{B})} = P_{a.\mathfrak{B}}(Q_{a.\mathfrak{B}})$

or $Q_{\mathfrak{B}}$ est défini

$= P_{\mathfrak{B}}(Q_{\mathfrak{B}}) = Q_{Début(\mathfrak{B})}$

$Q_{Début(\mathfrak{B})}$ est aussi défini

$= Q_{a.Début(\mathfrak{B})}$

$S_{Début(a.\mathfrak{B})}(x) = S_{a.\mathfrak{B}}(x) = S_{\mathfrak{B}}(x) = S_{Début(\mathfrak{B})}(x) = S_{a.Début(\mathfrak{B})}(x)$

si $x \neq Place_{b.\mathfrak{a}}$ autrement on a

$S_{Début(a.\mathfrak{A})}(x) = S_{a.Début(\mathfrak{B})}(\varphi(x))$

et ainsi de suite pour les autres accès □

ira-on	Place	T		Q		S		P		V	
		général	exception	général	exception	général	exception	général	exception	général	exception
id.	$\emptyset$	indéfini	/	indéfini	/	indéfini	/	indéfini	/	indéfini	/
	$Place_{\mathfrak{a}} \cup \{Place_{\mathfrak{A}}\}$	$Place_{\mathfrak{A}}$	/	$Q_{\mathfrak{A}}$	$Place_{\mathfrak{A}}$ si $Q_{\mathfrak{A}}$ indéfini	$S_{\mathfrak{A}}$	$S_{\mathfrak{A}}(*)=T_{\mathfrak{A}}$	$P_{\mathfrak{A}}$	$P_{\mathfrak{A}}(T_{\mathfrak{A}}) = *$	$V_{\mathfrak{A}}$	$V_{\mathfrak{B}}(*) = a$
te	$Place_{\mathfrak{a}} - \{T_{\mathfrak{x}}\}$	$S_{\mathfrak{A}}(T_{\mathfrak{A}})$	/	$Q_{\mathfrak{A}}$	indéfini si $T_{\mathfrak{a}} = Q_{\mathfrak{a}}$	$S_{\mathfrak{a}}$	/	$P_{\mathfrak{a}}$	/	$V_{\mathfrak{a}}$	/
ut	$Place_{\mathfrak{a}} - \{Q_{\mathfrak{A}}\}$	$T_{\mathfrak{A}}$	indéfini si $T_{\mathfrak{a}} = Q_{\mathfrak{a}}$	$P_{\mathfrak{A}}(Q_{\mathfrak{A}})$	/	$S_{\mathfrak{a}}$	/	$P_{\mathfrak{a}}$	/	$V_{\mathfrak{a}}$	/
D	$Place_{\mathfrak{a}} \cup \{Place_{\mathfrak{A}}\}$	T	$Place_{\mathfrak{A}}$ si $T_{\mathfrak{a}}$ est indéfini	$Place_{\mathfrak{A}}$	/	$S_{\mathfrak{a}}$	$S_{\mathfrak{B}}(Q_{\mathfrak{A}})=*$	$P_{\mathfrak{a}}$	$P_{\mathfrak{B}}(*)=T_{\mathfrak{A}}$	$V_{\mathfrak{a}}$	$V_{\mathfrak{B}}(*) = a$
	$Place_{\mathfrak{a}}$	$Q_{\mathfrak{A}}$	/	$Q_{\mathfrak{a}}$	/	$P_{\mathfrak{a}}$	/	$S_{\mathfrak{a}}$	/	$V_{\mathfrak{a}}$	/

Figure 2 : Représentation des opérations.

2.- Couples.

Type Couple (Alph)

fonctionnalité

JOINT : (Alph, Alph) → Couple ,

P1, P2: (Couple) → Alph,

axiomatique

P1(JOINT(x, y)) = x

P2(JOINT(x, y)) = y

Il y a peu de choses à dire sur ce type, sinon qu'il est de façon triviale suffisamment complet, puisque les seules expressions de type () → Couple sont de la forme JOINT(a, b) où a et b sont des formes normales de type () → Alph, alors trivialement

P1 (JOINT(a, b)) et P2 (JOINT(a, b)) se réduisent.

Une équation de la forme

JOINT(P1(c) , P2(c)) = c

que l'on pourrait s'attendre à trouver est inutile,car les seuls objets que l'on considère sont de la forme c = JOINT (a, b) et l'équation est alors de la forme c = JOINT (a, b) et l'équation est alors conséquence des deux autres. La représentation d'un couple est tellement évidente que nous n'en parlerons pas. Ce type sert dans le type de graphe qui suit.

3.- Graphes.

L'exemple qui suit est une spécification possible du type Graphe.

type Graphe [Noeud]

fonctionnalité

GRAPHEVIDE : () → Graphe,

AJNOEUD : (Graphe, Noeud) → Graphe,

AJARC : (Graphe, Couple(Noeud)) → Graphe,

PRNO : (Graphe, Noeud) → Bool

PRARC : (Graphe, Couple(Noeud)) → Bool ;

Axiomatique

PRNO(GRAPHEVIDE, n) = FAUX

PRNO(AJNOEUD(g, n'), n) =

SI n = n' ALORS VRAI

SINON PRNO(g, n)

PRNO(AJARC(g, a), n) =

SI n = P1(a) OU n=P2(a) ALORS VRAI

SINON PRNO(g, n)

PRARC(GRAPHEVIDE, a) = FAUX

PRARC(AJNOEUD(g, n), a) = PRARC (g, a)

PRARC(AJARC(g, a'), a) =

SI a = a' ALORS VRAI

SINON PRARC(g, a)

3.1.- Une première représentation.

Dans cette première représentation, une algèbre graphe $\mathcal{G}$ n'a pas de support mais seulement deux fonctions

$\in$: (Noeud) → Bool

ARC : (Noeud , Noeud) → Bool

avec un axiome

(ARC(a, b) ⇒ $\in$(a) ∧ $\in$ (b)) = VRAI

C'est à dire qu'un arc n'appartient au graphe que si ses deux sommets y appartiennent.

Graphevide est l'algèbre dans laquelle les deux opérations sont les fonctions identiques à FAUX. Elle vérifie trivialement l'équation

Ajnoeud($\mathcal{G}$, n) est l'algèbre $\mathcal{H}$ telle que

- si n' ≠ n alors $\in_{\mathcal{H}}(n') = \in_{\mathcal{G}}(n')$

et $\in_{\mathcal{H}}(n) = VRAI$

- $ARC_{\mathcal{H}}(n', n'') = ARC_{\mathcal{G}}(n', n'')$

$\mathcal{A}jarc(\mathcal{G}, a)$ est l'algèbre $\mathcal{H}$ telle que

- $\epsilon_{\mathcal{H}}$ coïncide avec $\epsilon_{\mathcal{G}}$ sauf en P1(a) et P2(a) où elle vaut VRAI
- si $b \neq a$ alors $ARC_{\mathcal{H}}(P1(b), P2(b)) = ARC_{\mathcal{G}}(P1(a), P2(b))$
  sinon $ARC_{\mathcal{H}}(P1(a), P2(a)) = VRAI$

	ϵ		ARC	
	Général	Exception	général	Exception
graphvide	FAUX		FAUX	
$\mathcal{A}jnoeud(\mathcal{G}, n)$	$\epsilon_{\mathcal{G}}$	$\epsilon(n) = VRAI$	$ARC_{\mathcal{G}}$	
$\mathcal{A}jarc(\mathcal{G}, a)$	$\epsilon_{\mathcal{G}}$	$\epsilon(P1(a))=VRAI$ $\epsilon(P2(a))=VRAI$	$ARC_{\mathcal{G}}$	$ARC(P1(a), P2(a))$ $= VRAI$

Figure 3 : Une première représentation du type Graphe.

$\mathcal{P}no(\mathcal{G}, n) = \epsilon_{\mathcal{G}}(n)$

$\mathcal{P}arc(\mathcal{G}, a) = ARC_{\mathcal{G}}(P1(a), P2(a))$

Vérifions l'une des équations

$\mathcal{P}no(\mathcal{A}jnoeud(\mathcal{G}, n'), n) =$ SI $n = n'$ ALORS VRAI
SINON $\mathcal{P}no(\mathcal{G}, n)$

en effet si $n = n'$ alors

$\mathcal{P}no(\mathcal{A}jnoeud(\mathcal{G}, n), n) =$
(d'après la définition de $\mathcal{P}no$)

$= \epsilon_{\mathcal{A}jnoeud(\mathcal{G}, n)}(n)$

(d'après la définition de $\mathcal{A}jnoeud$)

$=$ VRAI

si $n \neq n'$

$\mathcal{P}no(\mathcal{A}jnoeud(\mathcal{G}, n'), n) =$
(d'après la définition de $\mathcal{P}no$)

$\epsilon_{\mathcal{A}jnoeud(\mathcal{G}, n')}(n)$
$= \epsilon_{\mathcal{G}}(n)$

(d'après la définition de $\mathcal{P}no$)

$= \mathcal{P}no(\mathcal{G}, n)$

On peut montrer que, modulo l'isomorphisme, ces algèbres forment un modèle du type abstrait.

3.2.- UNE DEUXIÈME REPRÉSENTATION.

Dans cette deuxième représentation, une algèbre graphe $\mathcal{G}$ est constituée d'un support Sommet, d'une application

$ASS_{\mathcal{G}}$: (Noeud) → Sommet U {?}

et d'une application

$ADJ_{\mathcal{G}}$: (Sommet U {?}) → Ens (Noeud)

vérifiant les équations $ADJ(?) = VIDE$ et $[n \in ADJ(t) \Rightarrow ASS(n) \neq ?] = VRAI$

C'est une formalisation de la représentation d'un graphe par "liste d'adjacence" : ce sera le cas précisément si Ens(Noeud) est représenté par des listes (paragraphe 5.2 du chapitre 1, seconde représentation).

Les opérations graphevide, $\mathcal{A}jnoeud$ et $\mathcal{A}jarc$ sont données par la figure 4.

$\mathcal{P}no(\mathcal{G}, n) = [ASS_{\mathcal{G}}(n) \neq ?]$

$\mathcal{P}arc(\mathcal{G}, a) = PR(ADJ_{\mathcal{G}}(ASS_{\mathcal{G}}(P1(a))), P2(a))$

On peut montrer par exemple l'équation

$\mathcal{P}arc(ctgnoeud(\mathcal{G}, n), a) = \mathcal{P}arc(\mathcal{G}, a)$

en effet

$\mathcal{P}arc(ctgnoeud(\mathcal{G}, n), a) =$
(définition de $\mathcal{P}arc$)

$= PR(ADJ_{ctgnoeud(\mathcal{G}, n)}(ASS_{ctgnoeud(\mathcal{G}, n)}(P1(a))), P2(a))$
(définition de ADJ dans $ctgnoeud(\mathcal{G}, n)$)

$= PR(ADJ_{\mathcal{G}}(ASS_{\mathcal{G}}(P1(a)), P2(a))$
(définition de $\mathcal{P}arc$)

$= \mathcal{P}arc(\mathcal{G}, a)$

Ici aussi on montrera que ces algèbres forment modulo l'isomorphisme une algèbre de GRAPHE [Noeud].

		ASS		ADJ	
	Sommet	général	Particulier	général	Particulier
$\mathcal{G}raphe_{vide}$	$\emptyset$	?			
$ctgnoeud(\mathcal{G}, n)$	$Sommet_{\mathcal{G}} \cup \{s\}$ si $ASS_{\mathcal{G}}(n) \neq ?$ alors $s = ASS_{\mathcal{G}}(n)$	$ASS_{\mathcal{G}}$	$ASS(n) := s$	$ADJ_{\mathcal{G}}$	si $s \notin Sommet_{\mathcal{G}}$ $ADJ(s) = VIDE$ sinon $ADJ(s) = ADJ_{\mathcal{G}}(s)$
$ctarc(\mathcal{G}, JOINT(n, m))$	$Sommet \cup \{s,t\}$ si $ASS_{\mathcal{G}}(n) \neq ?$ alors $s = ASS_{\mathcal{G}}(n)$ sinon $s \notin Sommet_{\mathcal{G}}$ si $ASS_{\mathcal{G}}(m) \neq ?$ alors $t = ASS_{\mathcal{G}}(m)$ sinon $s \notin Sommet_{\mathcal{G}}$	$ASS_{\mathcal{G}}$	$ASS(n) = s$ $ASS(m) = t$	$ADJ_{\mathcal{G}}$	$ADJ(s) = AJ(ADJ_{\mathcal{G}}(n), t)$ $ADJ(t) = AJ(ADJ_{\mathcal{G}}(m), s)$

Figure 4 : Une deuxième représentation du type Graphe

4.- TABLES DES SYMBOLES.

Nous reprenons l'exemple de la table des symboles de Guttag, Horowitz et Musser [GUT 76] pour illustrer le chapitre 3 et seulement ce chapitre.

type Symboltable [Identifier, Attributlist]

syntaxe

INIT → Symboltable,
ENTERBLOCK(Symboltable) → Symboltable,
ADDID(Symboltable, Identifier, Attributelist) → Symboltable,
LEAVEBLOCK(Symboltable) → Symboltable,
ISINBLOCK (Symboltable, Identifier) → Boolean,
RETRIEVE(Symboltable, Indentifier) → Attributelist U {INDEFINED} .

sémantique

LEAVEBLOCK(INIT) = INIT,
LEAVEBLOCK(ENTERBLOCK(symbtab)) = symbtab,
LEAVEBLOCK(ADDID(symbtab, id, attlist)) = LEAVEBLOCK(symbtab),
ISINBLOCK(INIT, id) = FALSE,
ISINBLOCK(ENTERBLOCK(symbtab), id) = FALSE,
ISINBLOCK(ADDID(symbtab, id, attrlist), id1) = IF id = id1
THEN TRUE
ELSE ISINBLOCK(symbtab, id1),
RETRIEVE(INIT, id) = UNDEFINED,
RETRIEVE(ENTERBLOCK(symbtab), id) = RETRIEVE(symbtab, id),
RETRIEVE(ADDID(symbtab, id, attlist), id 1) =
IF id = id1
THEN attlist
ELSE RETRIEVE(symbtab, id1).

Considérons l'ensemble sstt des tables des symboles associées à des programmes où l'on a déclaré des identificateurs x avec l'attribut a

et des identificateurs y avec l'attribut b ; on a

sstt = ADDID(sstt, x, a) U ADDID(sstt, y, b)
U ENTERBLOCK(sstt) U ADDID(ADDID(INIT, x, a), y, b)

Il est facile de montrer que si &&vv = LEAVEBLOCK(sstt)

&&vv = &&vv U sstt U {INIT}

de même on peut montrer que

RETRIEVE(sstt, x) = a U RETRIEVE (sstt, x)

A l'aide de considérations sur les plus petits points fixes on pourrait montrer que

&&vv = sstt U {INIT}

RETRIEVE(sstt, x) = a

En revanche les calculs ci-dessus ne sont plus valables si l'on remplace x par {x, y} ainsi que vaut

RETRIEVE(ADDID(ADDID(INIT, x, a), y, b), {x, y}) ?

CHAPITRE V

calcul relationnel et représentation d'un type abstrait

CHAPITRE V

PRÉSENTATION GÉNÉRALE DU CALCUL RELATIONNEL ET SON APPLICATION A LA REPRÉSENTATION D'UN TYPE ABSTRAIT

1.- INTRODUCTION.

La démarche que nous avons proposée jusqu'à présent est la suivante :

- un formalisme équationnel décrit le type abstrait c'est à dire l'ensemble des objets auxquels on s'intéresse et les opérations, constructions et sélections qui permettent de passer des uns aux autres.

- un formalisme, qui n'est plus équationnel mais inéquationnel, (cf. le chapitre 1 paragraphe 1.2) décrit une représentation, c'est à dire une constitution interne possible des objets , cette description est faite en termes d'accès à l'intérieur de ces objets, les constructions apparaissent comme des constructions d'algèbres et les sélections comme des accès privilégiés.

La description des représentations en terme d'algèbres qui a le mérite, entre autres choses, d'être simple et claire et d'utiliser le même vocabulaire que la présentation algébrique des types abstraits offre cependant quelques inconvénients :

- Dans certains cas les accès intéressants ne sont pas univoques mais multivoques, autrement dit à partir d'un objet, on a accès à plusieurs autres (nous avons déjà évoqué des aspects similaires dans le chapitre 3 pour les opérations de niveau plus élévé).

- Pour affirmer certaines propriétés, il est intéressant de considérer non pas un accès univoque mais un accès multivoque, c'est le cas, par exemple, de la propriétés "tous les points d'un arbre sont atteints à partir de la racine" qui peut se traduire "la relation qui associe à la racine tous les points de l'arbre est incluse dans la relation, entre la racine et ses descendants, obtenue par itération de la relation " fil gauche ou droit". Il en est de même de toute propriété faisant appel à l'"inverse" d'une fonction non injective.

- Comme on l'a vu le système est un système d'inéquations car les fonctions sont partielles ; il n'est pas difficile alors de généraliser ces fonctions partielles vers des relations.

- Enfin, certains axiomes, que l'on veut écrire, contiennent des propriétés sur des sous-ensembles. Une sorte particulière a été introduite , grâce à laquelle les sous-ensembles et les accès 0-aires s'axiomatisent comme des relations.

1.1.- UN EXEMPLE PRÉLIMINAIRE.

Avant d'exposer le calcul relationnel, nous allons voir, sur un exemple, les différents éléments qu'il faut introduire. Supposons que l'on veuille décrire en utilisant des accès multivoques les "mots" ou "listes linéaires" tels que nous les avons vus au chapitre 3 paragraphe 1.

Tout d'abord nous faisons appel à une relation "successeur" notée S qui à une place fait correspondre une place, pour mettre en valeur son profil, nous la notons (Place) S (Place). Nous allons énoncer sous forme relationnelle le fait que "tout élément a au plus un successeur" en affirmant que la "composition de l'inverse de S avec S est incluse dans l'identité".

$$\rightarrow \text{(Place)}\ S^{-1}\ \text{(Place)}\ S\ \text{(Place)} \subset \text{(Place)}\ ID\ \text{(Place)}$$

Autrement dit, l'axiome que nous venons de donner si on l'interprête s'écrit :

$$x\ [\text{(Place)}\ S^{-1}\ \text{(Place)}\ S\ \text{(Place)}]\ y \text{ implique } x = y$$

c'est à dire

$$\forall z \quad x\ [\text{(Place)}\ S^{-1}\ \text{(Place)}\ S\ \text{(Place)}]\ z \text{ et } z\ [\text{(Place)}\ S\ \text{(Place)}]\ y \text{ implique } x = y$$

ou encore

$$\forall z \quad z\ [\text{(Place)}\ S\ \text{(Place)}]\ x \text{ et } z\ [\text{(Place)}\ S\ \text{(Place)}]\ y \text{ implique } x = y$$

ce qui correspond bien à ce que l'on voulait formaliser.

La relation S^{-1} est donc la relation prédécesseur.

Un autre accès qui intervient dans les listes linéaires est la tête d'une liste. Bien que cet accès soit 0-aires, nous le considérerons , pour unifier les concepts comme une relation T dont le but est Place et la source est une sorte particulière notée ( ) à un seul élément ; T est donc un sous-ensemble de Place dont il reste à spécifier qu'il contient (au plus) un élément (cf. 3.2.). Nous pouvons aussi énoncer que T n'a pas de prédécesseur :

$$\rightarrow \text{()}\ T\ \text{(Place)}\ S^{-1}\ \text{(Place)} \subseteq \text{()}\ VIDE\ \text{(Place)}$$

VIDE est la relation vide ; ainsi () VIDE (Place) est la partie vide de Place.

Les raisons qui font que les relations sont adaptées à l'étude des représentations de types abstraits ont déjà été dites. Mais après les quelques exemples proposés nous pouvons signaler encore un avantage du calcul relationnel : il n'utilise aucune instance de variables d'individus. Tout est dit en termes d'opérations de base : la composition, l'inversion , la réunion, l'intersection, les relations vides et pleines etc... si l'on a besoin de spécifier des individus, on utilise encore des relations dont la source est de type (). Ainsi les propriétés d'une relation pourront être prouvées par un calcul dans les algèbres relationnelles. Bien que manipulant des objets de type relation, c'est donc un calcul du premier ordre,tant qu'on ne manipule pas d'équations récursives [PAR 76] (voir aussi [BAC 78]).

1.3.- LES ALGÈBRES RELATIONNELLES, UNE SPÉCIFICATION DE TYPE ABSTRAIT.

Les algèbres relationnelles forment un type abstrait particulier qui constitue l'univers dans lequel on étudie la représentation des objets informatiques ; et l'étude qui suit est une étude formelle de ce type.
On notera que celui-ci n'est pas spécifié algébriquement. Cette vision est à rapprocher de celle de C. Pair et M.C. Gaudel qui étudient le type abstrait "structure d'information" comme l'univers dans lequel travaille un compilateur . [GAU 78] .

2.- LES ALGÈBRES RELATIONNELLES : PARTIE SYNTAXIQUE.

La description du langage des algèbres relationnelles proposée ici, se compose de trois niveaux :

- la *description des relations*, elles sont décrites par des termes construits à l'aide des opérations
- les *prédicats relationnels*, ils sont deux : l'égalité et l'inclusion
- les *assertions*, elles s'énoncent essentiellement comme une composition de prédicats. Dans le but de pouvoir les utiliser dans un système de preuve efficace, nous les mettrons sous forme de clauses de Horn [KOW 74]

2.1.- DESCRIPTION DES RELATIONS.

Ne serait-ce que pour considérer la sorte () qui sert à introduire les sous-ensembles ; les relations seront toutes typées avec une sorte source et une sorte but. Par analogie avec la première partie, nous écrirons les relations en majuscules et les sortes entre parenthèses et en minuscule avec une première lettre majuscule.

Exemple 1 : (Place) S (Place), () T (Place) □

Les relations constantes que l'on trouve dans toute algèbre relationnelle sont

- la relation pleine PLEINE souvent abrégée U .
- la relation vide VIDE souvent abrégée Ω .
- la relation identité ID, c'est aussi la relation d'égalité souvent abrégée E .

*X, *Y, *X1, *X2, *XA, X, Y, X1 etc... sont les notations des variables relations, (*T), (*S), (*T1), (*T2), (*Ta), (T), (S) etc... sont les notations des variables sortes.

Avec les relations constantes et les variables relations, avec la sorte () et les variables sortes , on construit des termes en se servant d'opérations : l'union, l'intersection, la composition, l'inversion et l'itération (figure 1). La dualité de certaines notations provient du fait que l'on veut faire coexister des notations traditionnelles et des notations susceptibles d'être utilisées comme données d'un programme informatique (certains signes n'existant pas sur les consoles habituellement utilisées).

NOTATIONS	CONCEPTS
()	sorte servant à construire les parties
(Place)	notation de sorte de nom place
(*T)	notation de variable sorte
(*Source) PLEINE (*But)	notation de la relation pleine
(*Source) VIDE (*But) Ω	notation de la relation vide
(*Source) ID (*But) E	notation de la relation identité
(*S) [A U B] (*T)	notation de l'union des relations A et B
(*S) [A ∩ B] (*T) (*S) [A & B] (*T)	notation de l'intersection des relations A et B
(*T1)A(*T2)B(*T3) A ; B	notation de la composition des relations A et B
(*T) A^{-1} (*S) (*T) A ? (*S)	notation de l'inversion de la relation (*S) A (*T)
(*T) A* (*T)	notation de l'itération de la relation (*T) A (*T)

Figure 1 : notations de l'algèbre relationnelle.

Exemple 2 :

(*T) [[A U B] & [PLEINE (*T1)* X] (*S) est un terme □

Très fréquemment une algèbre ne comporte qu'une seule sorte (disons T) en dehors de () ; dans ce cas on emploiera une écriture simplifiée pour les termes en supprimant toute occurrence de (T) dans les termes et en notant la composition ";" (T) A (T) B (T) sera remplacé par A ; B .

2.1.1.- RESTRICTION.

Voici quelques restrictions sur l'emploi des opérations et des constantes :

- la source de ID est égal à son but.
- A U B et A ∩ B (ou A & B) sont bien formés si et seulement si la source (resp le but) de A est égale à la source (resp le but) de B
- ($*T_0$) A ($*T_1$) B ($*T_2$) n'est bien formé que si le but ($*T_1$) de A est la source de B .
- (*T) A* (*T) n'est bien formés que si la source de A est égale à son but.

2.1.2.- PRIORITÉ ET PARENTHÈSAGE.

Afin de ne pas alourdir les expressions par de multiples parenthèses nous admettrons les règles suivantes :

- les parenthèsages d'expressions faisant intervenir plusieurs fois le même opérateur binaire associatif seront omis, c'est le cas de l'union, de l'intersection et de la composition.
- les parenthèsages d'expressions mixtes seront omis en tenant compte des priorités suivantes pour les opérateurs dans l'ordre décroissant : les opérations unaires, la composition, l'intersection et l'union.

2.1.3.- INUTILITÉ DE LA COMPLÉMENTATION.

Signalons que nous n'avons pas employé la complémentation, opération dont on peut se passer dans la formalisation que nous proposons et dont l'interprétation informatique est difficile.

2.2.- Ecriture d'un prédicat.

Un prédicat est de la forme

terme 1 = terme 2 pour l'égalité

ou

terme 1 ⊂ terme 2 (noté parfois terme 1 = C = terme 2)

pour l'inclusion. La seule restriction est que la source (resp le but) de terme 1 soit la même que la source (resp le but) de terme 2.

2.3.- Ecriture d'une assertion.

Une assertion sous la forme d'une clause de Horn est écrite ainsi

antécédent ⟶ séquent

antécédent est une suite éventuellement vide de prédicats et

séquent est un prédicat ou est vide.

3.- Les algèbres relationnelles : partie axiomatique.

L'interprétation des termes, des prédicats et des assertions est évidente et ne sera pas décrites complètement. Disons simplement qu'un terme peut être interprété comme une relation, cette relation fait éventuellement intervenir des variables. L'égalité et l'inclusion doivent être comprises dans leur sens classique.

Quant aux assertions, quatre cas sont à envisager :

- l'antécédent est composé des prédicats $p_1, \ldots, p_n$ et le séquent est s et alors s est la conséquence logique de la conjonction de $p_1, \ldots, p_n$.
- l'antécédent est composé des prédicats $p_1, \ldots, p_n$ et le séquent est vide, alors l'un des p_i est faux.
- l'antécédent est vide et le séquent est s , alors s est vrai
- si l'antécédent est vide et le séquent est vide, alors l'assertion est la contradiction, tout système où cette assertion peut être démontrée est contradictoire.

3.1.- Axiomes pour les algèbres relationnelles.

Tout d'abord *les relations de même source, même but forment des treillis distributifs* avec élément maximum PLEINE et minimum VIDE.

Axiomatique pour la composition et l'inversion.

Nous ne donnons pas ici une axiomatique minimum et cela pour deux raisons :

- une axiomatique minimum en nombre d'axiomes n'est probablement pas celle qui est minimum en longueur des démonstrations et ce point est crucial lors d'une automatisation.
- ayant abandonné la complémentation on ne peut utiliser l'axiomatique de Tardki qui nécessite l'utilisation de la complémentation pour la preuve de certaines propriétés qui ne font pas intervenir la complémentation. Je pense en fait que la construction d'une "bonne axiomatique" (c.a.d un système complet de réécriture) doit se faire en s'aidant de l'ordinateur (cf l'article de Knuth et Bendix [KNB 70] .

Voici quelques unes des propriétés.

La composition est associative

{T1} → (s) [X (t) Y] (u) Z (v) = (s) X (t) [Y (u) Z] (v)

La conversion est idempotente

{T2} → $(s)\ [X^{-1}]^{-1}\ (t) = (s)\ X\ (t)$

L'inversion est distributive par rapport à la composition

{T3} → $(s)[\ X\ (t)\ Y\ ]^{-1}\ (u) = (s)\ Y^{-1}\ (t)\ X^{-1}\ (u)$

La relation identité est élément neutre à droite pour la composition

{T4} → (s) X (t) ID(t) = (s) X (t)

L'axiome qui suit est un axiome très spécifique des algèbres relationnelles, il n'est pas équationnel, il régit les rapports entre la relation

vide, la composition , la conversion et l'intersection

{T5} (s) [[X (t) Y] ∩ Z] (u) = (s) VIDE (u)

→ (t) [[Y (u) Z^{-1}] ∩ X^{-1}] (s) = (t) VIDE (s)

La composition est distributive par rapport à l'union

{R1} → (s) X (t) [Y U Z] (u) = (s) [X (t) Y U X (t) Z] (u)

{R2} → (s) [Y U Z] (t) X (u) = (s) [Y (t) X U Z (t) X] (u)

L'inversion est distributive par rapport à l'union et à l'intersection

{R3} → (s) [X U Y] $^{-1}$ (t) = (s) [X^{-1} U Y^{-1}] (t)

{R4} → (s) [X ∩ Y]$^{-1}$ (t) = (s) [X^{-1} ∩ Y^{-1}] (t)

L'intersection, la composition et l'inversion sont reliées par la propriété suivante

{R5} → (s) [X (t) Y ∩ Z] (u) = (s) [X (t) {X^{-1} (s) Z ∩ Y} ∩ Z] (u)

De ces propriétés on peut déduire d'autres propriétés, que nous donnons sous forme simplifiée

{R5'} → X ; Y ∩ Z = (X ∩ Z ; Y^{-1}) ; Y ∩ Z

{R6} E ; X = X {R7} E^{-1} = E {R8} Ω^{-1} = Ω {R9} $\mathcal{U}^{-1}$ = $\mathcal{U}$

{R10} Ω ; X = X ; Ω = Ω

ou encore d'après [BAK 72] p. 170

{R11} X^{-1} ; X ⊂ E → X ; [Y ∩ T] = X ; Y ∩ X ; T

{R12} X ⊂ E → X^{-1} = X

{R13} X = (X ; $\mathcal{U}$ ∩ E) ; X {R13'} X = X ; ($\mathcal{U}$; X ∩ E)

{R14} X ; $\mathcal{U}$ = (X ; $\mathcal{U}$ ∩ E) ; {R14'} ; X = $\mathcal{U}$; ($\mathcal{U}$; X ∩ E)

{R15} X ; ∩ E = X ; X^{-1} ∩ E {R15'} ; X ∩ E = X^{-1} ; X ∩ E

3.2.- AUTRES AXIOMES.

Le produit de deux relations pleines est inclus dans une relation pleine

{U} → (s) PLEINE (t) (s) PLEINE (u) PLEINE (t) .

D'autre part () ne peut contenir plus d'un élément

{V} → () PLEINE () = () ID ()

De là nous déduisons par exemple le lemme suivant

Lemme 1 : → () X (t) X^{-1} () X (t) = () X (t)

Démonstration :

() X (t) = (d'après le résultat {R13} ci-dessus) = () [X(t)$\mathcal{U}$ ∩ E] () X (t)

= (d'après {R15} ci-dessus) = () [X (t) X^{-1} ∩ E] () X (t)

= (d'après {V}) = () X (t) X^{-1} () X (t) □

3.3.- AXIOMES D'ITÉRATIONS.

Nous en introduisons essentiellement quatre [CONW 71] .

Le premier "étoile-union" régit les rapports de l'union et de l'itération

{EU} → (s) [X U Y]* (s) = (s) [X*(s) Y]* (s) X* (s)

La seconde "étoile-produit" régit les rapports de l'itération et du produit

{EP} → (s) [X(t)Y]* (s) = (s) [ID U X (t) [Y (s) X]* (t) Y] (s)

La troisième "étoile-étoile" affirme l'idempotence de l'étoile

{EE} → (s) [X*] (s) = s X* (s)

La quatrième "étoile-inversion" affirme la commutativité des opérations

{EI} → (s) [X^{-1}]* (s) = (s) [X*]$^{-1}$ (s)

De {EU} et {EP} on déduit facilement

{EU'} (s) [X U Y]* (s) = (s) X* (s) [Y (s) X*] (s)

De même on montre

Lemme 2 : $X \subset Y \rightarrow X^* \subset Y^*$

Démonstration :

$X \subset Y \rightarrow Y = X \cup Y \rightarrow Y^* = [X \cup Y]^*$

$\rightarrow Y^* = [X^* Y]^* \quad X^* \rightarrow Y^* = X^* \cup X^* Y [X^* Y]^* X^*$

$\rightarrow X^* \subset Y^*$ □

On remarquera qu'il n'existe pas d'axiome "étoile-intersection" car il n'y a pas de rapports simples entre l'intersection et l'itération.

Le système d'axiomes proposé ici n'est certainement pas complet, car Conway a montré qu'il n'existe pas de système complet fini pour axiomatiser l'itération [CONW 71] et Lyndon a prouvé que le système relationnel proposé par Tarski est incomplet et qu'aucune famille finie d'axiomes ne peut axiomatiser les propriétés des relations [LYN 50] .

{T1}	$X ; Y ; Z = X ; (Y ; Z)$
{T2}	$X^{-1\,-1} = X$
{T3}	$X ; Y^{-1} = Y^{-1} ; X^{-1}$
{T4}	$X ; E = X$
{T5}	$X ; Y \cap Z = \Omega \rightarrow Y ; Z^{-1} \cap X^{-1} = \Omega$
{R1}	$X ; Y \cup Z = X ; Y \cup X ; Z$
{R2}	$Y \cup Z ; X = Y ; X \cup Z ; X$
{R3}	$X \cup Y^{-1} = X^{-1} \cup Y^{-1}$
{R4}	$X \cap Y^{-1} = X^{-1} \cup Y^{-1}$
{R5}	$X ; Y \cap Z = X ; [X^{-1} ; Z \cap Y] \cap Z$
{R5'}	$X ; Y \cap Z = [X \cap Z ; Y^{-1}] ; Y \cap Z$
{R6}	$E ; X = X$
{R7}	$E^{-1} = E$
{R8}	$\Omega^{-1} = \Omega$
{R9}	$\mathcal{U}^{-1} = \mathcal{U}$
{R10}	$\Omega = X ; \Omega$ {R10'} $\Omega = \Omega ; X$
{R11}	$X^{-1} ; X \subset E \rightarrow X ; [Y \cap T] = X ; Y \cap X ; T$
{R11'}	$X ; X^{-1} \subset E \rightarrow [Y \cap T] ; X = Y ; X \cap I ; X$
{R12}	$X \subset E \rightarrow X^{-1} = X$
{R13}	$X = (X ; \mathcal{U} \cap E) ; X$ {R13'} $X = X ; (\mathcal{U} ; X \cap E)$
{R14}	$X ; \mathcal{U} = (X ; \mathcal{U} \cap E) ; \mathcal{U}$ {R14'} $\mathcal{U} ; X = \mathcal{U} ; (\mathcal{U} ; X \cap E)$
{R15}	$X ; \mathcal{U} \cap E = X ; X^{-1} \cap E$ {R15'} $\mathcal{U} ; X \cap E = X^{-1} ; X \cap E$
{U}	$\mathcal{U} \subset \mathcal{U} ; \mathcal{U}$
{V}	$()\mathcal{U}() \subset () E ()$
{EU}	$[X \cup Y]^* = [X^* ; Y]^* ; X^*$
{EP}	$[X ; Y]^* = E \cup X ; [Y ; X]^* ; Y$
{EE}	$[X^*]^* = X^*$
{EU'}	$[X \cup Y]^* = X^* ; [Y ; X^*]$

Figure 2 :
Axiomes et résultats sur les algèbres relationnelles.

4.- ETUDE D'UN EXEMPLE LES LISTES LINÉAIRES.

4.1.- REPRÉSENTATION DES OBJETS.

En adjoignant aux symboles et aux axiomes des algèbres relationnelles un certain nombre de constantes et d'axiomes, on peut décrire une représentation des objets, d'un type abstrait. Nous allons formaliser les listes linéaires. Supposons que nous ayons deux sortes (Place) et (Alph) et quatre relations :

- () T (Place) la tête
- () D (Place) le dernier
- (Place) S (Place) la succession
- (Place) VA (Alph) les valeurs

Des axiomes possibles sont les suivants : (ils sont présentés d'abord en français ; puis en termes de relations).

La tête est unique

{LL1} → (Place) T^{-1} () T (Place) ⊂ (Place) ID (Place)

Toute place a au plus un successeur

{LL2} → (Place) S^{-1} (Place) S (Place) ⊂ (Place) ID (Place)

Le dernier est unique

{LL3} → (Place) D^{-1} () D (Place) ⊂ (Place) ID (Place).

Toute place a au plus un prédecesseur

{LL4} → (Place) S () S^{-1} (Place) ⊂ (Place) ID (Place).

Le premier n'a pas de précesseur

{LL5} → () T (Place) S^{-1} (Place) ⊂ () VIDE (Place)

Le dernier n'a pas de successeur

{LL6} → () D (Place) S (Place) ⊂ () VIDE (Place)

Le domaine de S est la liste et seulement la liste

{LL7} → () PLEINE (Place) S (Place) ⊂ () T (Place) S* (Place) S (Place)

La liste est constituée des points que l'on peut atteindre à partir du dernier par les prédecesseurs

{LL8} → () T (Place) S* (Place) = () D (Place) $[S^{-1}]$*(Place)

VA est fonctionnelle

{LL9} → (Alph) VA^{-1} (Place) VA (Alph) ⊂ (Alph) ID (Alph)

Le domaine de définition de VA contient la liste

{LL10} → () T (Place) S* (Place) ⊂ () (Alph) VA^{-1} (Place)

{LL1} T^{-1} ; T ⊂ E	{LL2} S^{-1} ; S ⊂ E
{LL3} D^{-1} ; D ⊂ E	{LL4} S ; S^{-1} ⊂ E
{LL5} T ; S^{-1} = Ω	{LL6} D ; S = Ω
{LL7} $\mathcal{U}$; S = (T ; S*) ; S	{LL8} T ; S* = D ; (S^{-1})*
{LL9} VA^{-1} ; VA ⊂ E	{LL10} T ; S* ⊂ $\mathcal{U}$; VA^{-1}

Figure 3 : Axiomes des listes.

4.2.- REPRÉSENTATION DE L'OPÉRATION Rest

Soit $\mathcal{L}$ une liste, on définit l'opération Rest($\mathcal{L}$) en donnant les relations T' , D', S' et VA' en fonction de T, D, S et VA . Nous essaierons de justifier ces choix en français.

La nouvelle tête est le successeur de l'ancienne tête

() T' (Place) = () T (Place) S (Place)

N'ont de successeurs dans la nouvelle liste que les éléments qui ont un prédécesseur dans l'ancienne

(Place) S' (Place) = (Place) S^{-1} (Place) S (Place) S (Place)

si D a un prédécesseur D' est D sinon D n'est pas défini

() D' (Place) = () D (Place) S^{-1} (Place) S (Place)

Les nouvelles valeurs ne sont attribuées qu'à des éléments qui avaient un prédécesseur dans l'ancienne liste

(Place) VA' (Alph) = (Place) S^{-1} (Place) S (Place) VA (Alph)

$S' = S^{-1} ; S ; S$
$T' = T ; S$
$D' = D ; S^{-1} ; S$
$VA' = S^{-1} ; S ; VA$

figure 4 : Définition de Rest

4.3.- PREUVE DE LA CORRECTION DE L'OPÉRATION Rest

Nous allons montrer que l'opération Rest transforme une liste en une liste autrement dit que les nouvelles relations vérifient les axiomes des listes linéaires.

Lemme 3 : $T'^{-1} ; T' \subset E$

Démonstration :

$T'^{-1} ; T'$ = (par définition) = $S^{-1} ; T^{-1} ; T ; S \subset$ {LL1} $S^{-1} ; E ; S \subset$ {LL2} $\subset E$ □

Lemme 4 : $S'^{-1} ; S' \subset E$

Démonstration :

$S'^{-1} ; S'$ = (par définition) $S^{-1} ; S^{-1} ; S ; S^{-1} ; S ; S \subset$ {LL4} $S^{-1} ; S^{-1} ; E ; S ; S \subset$ {LL2} $S^{-1} ; E ; S ; S \subset$ {LL2} E □

Lemme 5 : $D'^{-1} ; D' \subset E$

Démonstration :

$D'^{-1} ; D'$ = (par définition) $S^{-1} ; S ; D^{-1} ; D ; S^{-1} ; S \subset$ {LL3} $S^{-1} S ; S^{-1} ; S \subset$ {LL2} $E ; E = E$ □

Lemme 6 : $S' ; S'^{-1} \subset E$

Démonstration :

immédiat comme pour le lemme 4

Lemme 7 : $T' ; S'^{-1} = \Omega$

Démonstration :

$T' ; S'^{-1}$ = (par définition) $T ; S ; S^{-1} ; S^{-1} ; S \subset$ {LL4} $T ; E S^{-1} ; S = T ; S^{-1} ; S \subset$ {LL5} $\Omega ; S = \Omega$ □

d'où $T' ; S'^{-1} = \Omega$ car Ω est minimum

Lemme 8 : $D' ; S' = \Omega$

Démonstration :

$D' ; S'$ = (par définition) $D ; S^{-1} ; S ; S^{-1} ; S ; S \subset$ {LL2} $D ; E ; E ; S = D ; S$ {LL6} $= \Omega$ □

Lemme 9 : $S ; S^{-1} ; S = S$

Démonstration :

$S =$ {R13'} et {R15'} $= S ; (S^{-1} ; S \cap E) =$ {LL2} $= S ; S^{-1} ; S$ □

Lemme 10 : $T' ; S'^* = (T ; S^*) ; S$

Démonstration :

$T' ; (S')^* = T ; S ; (S^* ; S^{-1} ; S)^*$

= {EP} $T ; S ; (E \cup S^{-1} ; S ; (S ; S^{-1} ; S)^* ; S)$

= (lemme 9) $T ; S ; (E \cup S^{-1} ; S ; S^* ; S)$

= {R1} $T ; S \cup T ; S ; S^{-1} ; S ; S^* ; S$

= (lemme 9) $T ; S \cup T ; E ; S^* ; S$ = {T4} $T ; S \cup T ; S^* ; S$

= {R2} $(T ; S^*) ; S$ □

Lemme 11 : $T' ; (S')^* ; S' = \mathcal{U} ; S'$

Démonstration :

$T' ; S'^* ; S'$ = (lemme 9) = $T ; S^* ; S ; S'$

= {LL7} $\mathcal{U} ; S ; S'$ = {R14'} = $\mathcal{U} ; (\mathcal{U} ; S \cap E) ; S'$

= {R15'} $\mathcal{U} ; (S^{-1} ; S \cap E) ; S'$

= (par définition et {LL2}) $\mathcal{U} ; S^{-1} ; S ; S^{-1} ; S ; S$

= (lemme 9) $\mathcal{U} ; S^{-1} ; S ; S = \mathcal{U} ; S'$ □

Lemme 12 : $D' ; S'^{-1} = T' ; S'^*$

Démonstration :

$D' ; S'^{-1}$ = (par définition) $D ; S^{-1} ; S ; (S^{-1} ; S^{-1} ; S)^*$

= {EP} $D ; S^{-1} ; S ; (E \cup S^{-1} ; (S^{-1} ; S ; S^{-1})^* ; S^{-1} ; S)$

= (lemme 9 et {T3} et {R1}) $D ; S^{-1} ; S \cup D ; S^{-1} ; (S^{-1})^* ; S^{-1} ; S$

= {R2} $D ; (E \cup S^{-1} ; (S^{-1})^*) ; S^{-1} ; S$

= {EP} $D ; (S^{-1})^* ; S^{-1} ; S$ = {LL6} $D ; S \cup D ; (S^{-1})^* ; S^{-1} ; S$

= {R2} $D ; (E \cup (S^{-1})^* ; S^{-1}) ; S$ = {EP} $D ; (S^{-1})^* ; S$

= {LL8} $T ; S^* ; S$ = (lemme 10) = $T' ; S'^*$ □

Lemme 13 : $VA'^{-1} ; VA' \subset E$

Démonstration :

$VA'^{-1} ; VA'$ = (par définition) $VA^{-1} ; S ; S^{-1} ; S ; S^{-1} ; VA$

$\subset$ ({LL4} et {LL9}) E □

Lemme 14 : $T' ; S'^* = \mathcal{U} ; (VA')^{-1}$

Démonstration :

$T' ; S'^*$ = (lemme 10) $T ; S^* ; S$ = ({LL5} et lemme 9) $T ; S^{-1} ; S \cup T ; S^* ; S ; S^{-1} ; S$

= ({R1} et {R2}) $T ; (E \cup S^* ; S) ; S^{-1} ; S$

= {EP} $T ; S^* S^{-1} ; S$ = {LL10} $\mathcal{U} ; VA^{-1} ; S^{-1} ; S$

= (par définition) $\mathcal{U} ; VA'^{-1}$ □

Remarquons de $(VA')^{-1}$ $VA' \subset E$ est évident ; ainsi d'après les lemmes 3, 4, 5, 6, 7, 8, 11, 12, 13, 14 Rest transforme une liste linéaire en une liste linéaire.

4.4.- Représentation de l'opération $\mathcal{A}_j$

Soit $\mathcal{L}$ une liste linéaire pour définir $\mathcal{A}_j(A, \mathcal{L})$ on définit les nouvelles relations S' , T' et VA' et l'on caractérise D' .

La nouvelle tête est un élément H

→ () T'(Place) = () H (Place) .

La succession S' est la succession donnée par S et le fait que le successeur de H est T .

→ (Place S' (Place) = (Place) $[S \cup H^{-1}$ () T] (Place).

La valeur VA' est VA ou bien la valeur de H est A

→ (Place VA' (Alph) = (Place) $[VA \cup H^{-1}$ () A] (Alph)

Le dernier D' est caractérisé par les trois axiomes {LL3} , {LL6} et {LL8} déjà rencontrés.

Notons que pour que ces définitions aient un sens il faut stipuler quelques propriétés

H est constitué d'au plus un élément

→ (Place) H^{-1} (H (Place) ⊂ (Place) ID (Place)
et H contient au moins un élément

→ () H (Place) H^{-1} () = () ID ()

A est constitué d'au plus un élément et d'au moins un élément .

→ (Alph) A' () A (Alph) ⊂ (Alph) ID (Alph)

→ () A (Alph) A^{-1} () = () ID ()

VA n'attribue pas de valeur à H

→ () H (Place) VA (Alph) = () VIDE (Alph)

$T' = H$

$S' = S \cup H^{-1}$

$VA' = VA \cup H^{-1}$; A

D' ; $S' = \Omega$ $\quad D'^{-1}$; $D' \subset E$

T' ; $(S')^* = D'$; $(S'^{-1})^*$

figure 5 : Définition de t_j

H^{-1} ; $H \subset E$ $\quad H$; $H^{-1} = E$

A^{-1} ; $A \subset E$ $\quad A$; $A^{-1} = E$

H ; $VA = \Omega$

figure 6 : Propriétés de H et A

4.5.- PREUVE DE LA CORRECTION DE

Remarquons que T'^{-1} ; $T' \subset E$ et D'^{-1} ; $D' \subset E$ sont évidents ; le premier d'après les propriétés de H , le second puisqu'il figure dans les définitions.

Lemme 15 : H ; $S = \Omega$

Démonstration :

H ; $S = H$; $S \cap \mathcal{U}$; S = (par hypothèse) H ; $S \cap T$; S^* ; S

= {LL10} H ; $S \cup \mathcal{U}$; VA^{-1} ; S = {R11'} et {LL4} $(H \cap \mathcal{U} ; VA^{-1})$; S

= {R5'} $(H \cap (\mathcal{U} \cap H ; VA) ; VA^{-1})$; S

(Propriété de H)

= $(H \cap (\mathcal{U} \cap \Omega) ; VA^{-1})$; $S = \Omega$ □

Lemme 16 : S'^{-1} ; $S' \subset E$

Démonstration :

S'^{-1} ; $S' = (S^{-1} \cup T^{-1} ; H)$; $(S \cup H^{-1} ; T)$

= {R1} S^{-1} ; $S \cup T^{-1}$; H ; $S \cup S^{-1}$; H^{-1} ; $T \cup T^{-1}$; H ; H^{-1} ; T

= (lemme 15) S^{-1} ; $S \cup T^{-1}$; H ; H^{-1} ; T

(par hypothèse H ; $H^{-1} = E$)

= S^{-1} ; $S \cup T^{-1}$; T ({LL1} et {LL2}) E □

Lemme 17 : S' ; $S'^{-1} \subset E$

Démonstration :

S' ; $S'^{-1} = (S \cup H^{-1} ; T)$; $(S^{-1} \cup T^{-1} ; H)$

= {R1} S ; $S^{-1} \cup H^{-1}$; T ; $S^{-1} \cup S$; T^{-1} ; $H \cup H^{-1}$; T ; T^{-1} ; H

or T ; $S^{-1} = \Omega$ et T ; $T^{-1} \subset \mathcal{U} = E$

S ; $S^{-1} \cup H^{-1}$; $H \subset E$ □

Lemme 18 : H ; $S^{-1} = \Omega$

Démonstration :

H ; S^{-1} = ({R11'} et {LL2}) H ; $S^{-1} \cap \mathcal{U}$; S^{-1}

({R13'} et {R15'}) = H ; $S^{-1} \cap \mathcal{U}$; $(S$; $S^{-1} \cap E)$

= {LL4} H ; $S^{-1} \cap \mathcal{U}$; S ; S^{-1} = {LL7} H ; $S^{-1} \cap T$; S^* ; S ; S^{-1}

$\subset H$; $S^{-1} \cap T$; S^* ; S^{-1}

on continue comme au lemme 15

$\subset \Omega$ □

Lemme 19 : $H ; T^{-1} \subset \Omega$

Démonstration :

$H ; T^{-1} ; S \subset H ; (T ; S^*)^{-1} \subset$ {LL10} $H ; VA ; \mathcal{V}^{-1} = \Omega$ □

Lemme 20 : $T' ; S'^{-1} = \Omega$

Démonstration :

$T' ; S'^{-1}$ = (par définition) $H ; (S^{-1} \cup T^{-1} ; H)$ = {R1} $H ; S^{-1} \cup H ; T^{-1} ; H$

= (lemme 18 et lemme 19) Ω □

Lemme 21 : $VA'^{-1} ; VA' \subset E$

Démonstration :

$VA'^{-1} ; VA'$ = (par définition) $(VA \cup H^{-1} ; A)^{-1} ; (VA \cup H^{-1} ; A)$

$= VA^{-1} ; VA \cup VA^{-1} ; H^{-1} ; A \cup A^{-1} ; H ; VA \cup A^{-1} ; H ; H^{-1} ; A$

or $H ; VA = \Omega$ et $H ; H^{-1} = E$

$= VA^{-1} ; VA \cup A^{-1} ; A \subset$ ({LL9} et définition de $\mathcal{V}_j$) E □

Lemme 22 : $H ; S'^* = H \cup T ; S^*$

Démonstration :

$H ; S'^*$ = (par définition) $H ; (S \cup H^{-1} ; T)^*$

= {EU} $H ; (S^* ; H^{-1} ; T)^* ; S^*$

= {EP} $H ; (H^{-1} ; T \cup S^* ; S ; H^{-1} ; T)^* ; S^*$

= (lemme 18) $H ; (H^{-1} ; T)^* ; S^*$

= {EP} $H ; (E \cup H^{-1} ; (T ; H^{-1})^* ; T) ; S^*$

= (le lemme 19 donne $T ; H^{-1} = \Omega$ donc $(T ; H^{-1})^* = E$)

= $H ; (E \cup H^{-1} ; T) ; S^*$ = ({R1} et {T4}) $H ; S^* \cup H ; H^{-1} ; T ; S^*$

= ($H ; S^* = H \cup H ; S ; S^*$ d'après lemme 15 $H ; S^* = H$) $H \cup H ; H^{-1} ; T ; S^*$

= (propriété de H) $H \cup T ; S^*$ □

Lemme 23 : $T' ; S'^* = \mathcal{V} ; (VA')^{-1}$

Démonstration :

$T' ; S'^*$ = (lemme 22) $H \cup T ; S^* \subset$ {LL10} $H \cup \mathcal{V} ; VA^{-1}$

= (propriété de A) $A ; A^{-1} ; H \cup \mathcal{V} ; VA^{-1}$

= $(A ; A^{-1} \cap E) ; H \cup \mathcal{V} ; VA^{-1}$

= ({R15'} et {V}) $(\mathcal{V} ; A^{-1} \cap \mathcal{V}) ; H \cup \mathcal{V} ; VA^{-1}$

= $\mathcal{V} ; A^{-1} ; H \cup \mathcal{V} ; VA^{-1}$

= {R1} $\mathcal{V} ; (A^{-1} ; H \cup VA^{-1})$ = (par définition) $\mathcal{V} ; (VA')^{-1}$ □

La plupart des lemmes ainsi démontrés sont des propriétés des listes, les autres propriétés font partie de la définition de $\mathcal{V}_j$; ainsi $\mathcal{V}_j$ est correcte.

4.6.- La liste linéaire Vide

La liste Vide est la liste qui vérifie $T = \Omega$, il est facile d'en déduire qu'alors $D = \Omega$, $S = \Omega$, $VA = \Omega$.

4.7.- La représentation de Tête .

L'opération Tête fait correspondre à une liste $\mathcal{L}$ sa tête qui est un élément de Alph. La forme de cet élément a pour source ( ) et pour but (Alph), c'est l'élément () T (Place) VA (Alph) ; s'il existe, il est unique puisque $Tête(\mathcal{L})^{-1} ; Tête(\mathcal{L}) \subset E$ si () T (Place) VA (Alph) = () Ω (Alph) on pose Tête ($\mathcal{L}$) = INDEF.

4.8.- PREUVE DE L'ÉQUATION $\mathcal{R}este(\mathcal{A}lg(A,\mathcal{L})) = \mathcal{L}$

Pour montrer que ces représentations sont bien celles des listes, il suffit de montrer que les équations du type abstrait sont vérifiées. Dans la suite, la seule à laquelle nous nous intéresserons est l'équation $\mathcal{R}este(\mathcal{A}lg(A,\mathcal{L})) = \mathcal{L}$, nous allons la montrer de façon rigoureuse comme auparavant.
Notons T, S, D, VA les relations de $\mathcal{A}lg$, T' ; S', D', VA' les relations de $\mathcal{A}lg(A,\mathcal{L})$ et T", S", D", VA" les relations de $\mathcal{R}este(\mathcal{A}lg(A,\mathcal{L}))$.

Lemme 24 : T = T"

Démonstration :

$T" = T' ; S' = H ; (S \cup H^{-1} ; T) = H ; S \cup H ; H^{-1} ; T$

= (lemme 15) $H ; H^{-1} ; T$ = (propriété de H) T □

Lemme 25 : S = S"

Démonstration :

$S" = S'^{-1} ; S' ; S' = (S \cup H^{-1} ; T)^{-1} ; (S \cup H^{-1} ; T) ; (S \cup H^{-1} ; T)$

$= (S^{-1} ; S \cup S^{-1} ; H^{-1} ; T \cup T^{-1} ; H ; S \cup T^{-1} ; H^{-1} ; H ; T);(S \cup H^{-1};T$

= (lemme 15) $(S^{-1} ; S \cup T^{-1} ; T) ; (S \cup H^{-1} ; T)$

$= S^{-1} ; S ; S \cup S^{-1} ; S ; H^{-1} ; T \cup T^{-1} ; T ; S \cup T^{-1} ; T ; H^{-1} ; T$

= (lemme 18 et lemme 19) $S^{-1} ; S ; S \cup T^{-1} ; T ; S$

$= (S^{-1} ; S \cap T^{-1} ; T); S =$ (lemme 28) $[(T ; S*)^{-1}; (T ; S*) \cap E]; S =$

$\{\mathcal{V} ; T ; S* \cap E\} ; S$

{R11'} $\mathcal{V} ; T ; S* ; S \cap E = \mathcal{V}; \mathcal{V} ; S \cap E = \mathcal{V}; S \cap E = \{\mathcal{V} \cap E\} ; S$ □

Lemme 26 : $S^{-1} ; (S* ; T)^{-1} ; T \cap E = \Omega$

Démonstration :

$S^{-1} ; (T ; S)^{-1} ; T \cap E =$ {LL7} $S^{-1} ; \mathcal{V}; T \cap E$

= ({R5} et {T2}) S^{-1}; [S ; E ∩ $\mathcal{V}$; T] ∩ E = S^{-1} ; [S ∩ $\mathcal{V}$; T] ∩ E

= {R5'} S^{-1} ; [[$\mathcal{V}$ ∩ S ; T^{-1}] ; T ∩ S] ∩ E

= {LL5} S^{-1} ;[[$\mathcal{V}$ ∩ Ω] ; T ∩ S] ∩ E

= (diverses règles sur Ω) Ω □

Lemme 27 : $S^{-1} ; (S* ; T)^{-1} ; S* ; T ; S \cap E = S^{-1} ; S$

Démonstration :

$S^{-1} ; (S* ; T)^{-1} ; S* ; T ; S \cap E =$ {LL7} $S^{-1} ; \mathcal{V} ; \mathcal{V} ; S \cap E =$

$S^{-1} ; \mathcal{V} ; S \cap E$

= ({R5} et {T2}) S^{-1} ; [S ; E ∩ $\mathcal{V}$; S] ∩ E = S^{-1} ; [E ; S ∩ $\mathcal{V}$; S] ∩ E

= {R11'} S^{-1} ;[$\mathcal{V}$ ∩ E]; S ∩ E = S^{-1} ; E ; S ∩ E = S^{-1} ; S ∩ E

= {LL2} S^{-1} ; S □

Lemme 28 : $(T ; S*)^{-1} ; (T ; S*) \cap E = S^{-1} ; S \cup T^{-1} ; T$

Démonstration :

$(T ; S*)^{-1} ; (T ; S*) \cap E =$ {EP} $(T \cup T ; S* ; S)^{-1} (T \cup T ; S* ; S)$

$= T^{-1} ; T \cup T^{-1} ; T ; S* ; S \cup S^{-1} ; (T* ; S*)^{-1} T \cup S^{-1} ; (T ; S)^{-1} ;$

$(T ; S*) ; S^{-1}$

= (lemme 26 et lemme 27) $T^{-1} ; T \cup S^{-1} ; S$ □

Lemme 29 : VA = VA"

Démonstration :

VA" = (par définition) $S'^{-1} ; S' ; VA'$ = (lemme 16 et par définition)

$= [S^{-1} ; S \cup T^{-1} ; T] ; [VA \cup H^{-1} ; A]$

= (lemme 28 et {R1} $[(T ; S*)^{-1} ; (T ; S*) \cap E] ; VA \cup S^{-1} ; S ;$

$H^{-1} ; A \cup T^{-1} ; T ; H^{-1} ; A$

= (lemme 18 et lemme 19) $[(T ; S^*)^{-1} ; (T ; S^*) \cap E] ; VA$

= {R15'}$[\mathcal{D}; (T ; S^*) \cap E] ; VA$ = {LL10}$[\mathcal{D} ; \mathcal{D} ; VA^{-1} \cap E] ; VA$

= {U}$[\mathcal{D}; VA^{-1} \cap E] ; VA$ = ({R15} et {R15'}) $VA ; \mathcal{D} \cap E ; VA$

= {R13} VA □

4.9. Théorème de la représentation relationnelle des listes linéaires.

Enoncé : Les algèbres relationnelles et les opérations décrites dans les paragraphes 4.1 à 4.7 forment un modèle du type abstrait Liste [Alph]

Fonctionnalité

VIDE : () → Liste
AJ : (Alph, Liste) → Liste
TETE : (Liste) → Alph
RESTE : (Alph, Liste) → Liste

Axiomatique

TETE (VIDE) = INDEF
TETE (AJ (a, ℓ)) = a
RESTE (VIDE = VIDE
RESTE (AJ(a, ℓ)) = ℓ

Démonstration :

Les lemmes 3 à 23 démontrent le fait que les opérations transforment une liste en une liste, les lemmes 24 et 25 démontrent la quatrième équation. D'autres lemmes non donnés ici établissent les autres équations □

5.- Conclusion et bibliographie.

L'exemple des listes montre une représentation d'un type abstrait en termes d'algèbres relationnelles ; la relation S* et les relations qui en sont dérivées T ; S* et D ; S* ne sont pas fonctionnelle justifient ici l'introduction du calcul relationnel. Des exemples plus pertinents pourraient être tirés de l'étude de la représentation de types plus complexes, notamment ceux qui interviennent à l'occasion de la description de systèmes d'information. Cette approche nous a été inspirée des travaux de de Bakker de Roever, Hitchcock et Park [BAK 72] , [ROE 74] [HIT 72] tous dérivés des travaux de Tarski [TAR 40] . L'introduction du type () nous a paru une innovation intéressante pour la représentation des structures de données.

On ne peut pas parler de calcul relationnel et des structures de données sans parler des travaux de Codd [COD 70] . Nous citerons aussi Abrial [ABR 74][ABR 78] qui propose le calcul relationnel pour spécifier un problème informatique.

Les approches de Codd et Abrial n'utilisent pas le même formalisme que celles de de Roever, Hitchcock et Park mais en fait toutes poursuivent le même objectif : décrire un type abstrait à l'aide de relations. Le problème de la représentation dans ce cas nous a paru , pour l'instant, trop complexe pour pouvoir être exprimé directement en termes d'algèbres filles (relationnelles ou non), c'est pourquoi, dans ce chapitre, les relations ne sont apparues qu'au niveau des représentations.

CHAPITRE VI

automutisation des preuves

CHAPITRE VI

EXPÉRIENCES D'AUTOMATISATION DES PREUVES DANS LES SYSTÈMES ALGÉBRIQUES

I.- INTRODUCTION.

Comme il est apparu au chapitre précédent, les démonstrations dans les algèbres relationnelles sont particulièrement fastidieuses. On se trouve confronté au problème suivant :
chaque fois , il s'agit de prouver une égalité ou une inclusion, à partir des axiomes des algèbres relationnelles $\mathcal{R}$ et des axiomes du systèmes particulier $\mathcal{S}$ que l'on étudie,(les listes linéaires dans le chapitre 5). Confronté à ce problème, je l'ai attaqué avec les moyens dont je disposais à Nancy, c'est à dire essentiellement le compilateur Pascal.

Je l'ai abordé de façon classique, c'est à dire par des techniques de réfutation : car pour prouver D à partir de $\mathcal{R}$ et $\mathcal{S}$ il faut réfuter le système $\mathcal{R} + \mathcal{S} + \neg D$, c'est à dire montrer son inconsistance. Ainsi une réfutation de D est une suite $D_0, D_1, \ldots, D_n$ d'assertions où $D_0 = D$ et telles que D_i est une conséquence de $\mathcal{R}+\mathcal{S}+\neg D$ et D_n est la contradiction (vrai → faux par exemple).

En général, D_{i+1} est obtenu en inférant une assertion D_i à partir de $\mathcal{R}+\mathcal{S}+D \cup \{D_1, \ldots, D_i\}$ et de D_i . La règle d'inférence qui s'impose est la *résolution* elle est décrite au paragraphe 3. Elle utilise une algorithme d'unification qui est décrit au paragraphe 1. La procédure de choix de l'assertion à résoudre avec D_i dans $(\mathcal{R}+\mathcal{S}+D) \cup \{D_1, \ldots, D_i\}$, s'appelle

la *procédure de sélection*, si ce choix se fait dans $(\mathcal{R}+\mathcal{P}+ D)$ la sélection s'en trouve grandement facilitée puisqu'on opère toujours sur le même ensemble, on dit qu'on fait alors une *résolution linéaire*. Dans le cas général, la résolution linéaire n'est pas complète, c'est à dire qu'elle n'arrive pas à réfuter toutes les familles contradictoires d'axiomes, compte tenu de la spécificité de notre objet (études de propriétés algébriques) et sous l'influence de Prolog [COL 75] [ROU 75] [WAR 76] et Kowalski [KOW 74] , nous avons choisi un cas où cette réfutation est toujours possible, c'est celui des clauses de Horn ; ce sont des clauses de la forme $\neg p_1 \vee \ldots \vee \neg p_n \quad q$ où $p_1, \ldots, p_n, q$ sont des prédicats sans négation, nous les écrirons sous la forme $p_1, \ldots, p_n \rightarrow q$. Malgré cela, le fait que la recherche se fasse par essais et erreurs rend les preuves extrêmement longues. Aussi est-il apparu qu'il fallait "programmer" la preuve, autrement dit l'orienter pour aboutir à une solution sans retours en arrière (ou très peu). Dans le cas de l'égalité, une méthode de réécriture a paru efficace. L'algorithme de superposition de Knuth et Bendix (déjà cité au chapitre 3) fournit un des outils les plus efficaces pour améliorer la programmation de la preuve dans un système algébrique.

Avertissements :

Les algorithmes d'unification sont présentés ici en se servant des types abstraits et autant que possible de la spécification algébrique de ceux-ci. Il est certain que des recherches doivent être poursuivies pour rendre le développement encore plus systématique et rigoureux, au départ ce n'était pas mon objectif. Si l'on compare avec l'exposé de Huet, on notera que l'on obtient une plus grande clarté au niveau des algorithmes proprement dits.

Ce chapitre ne vise pas à présenter un produit parfaitement fini, mais seulement les résultats de travaux exploratoires sur les possibilités de démonstrations dans les systèmes algébriques et parmi eux les algèbres

relationnelles. C'est pourquoi on insiste essentiellement sur l'aspect expérimental. Ces recherches très liées à celles développées à propos des types abstraits seront poursuivis en collaboration avec Fernand Reinig.

2.- UNIFICATION.

2.1.- PRINCIPES GÉNÉRAUX.

Voici quelques définitions (voir le chapitre 5 de la thèse de Huet [HUE 76]). Soit un ensemble de variables, soit $P(F, X)$ l'ensemble des termes sur X . On suppose que F n'est pas constitué seulement de symboles monadiques et que tous les symboles sont homogènes.

Définition 1 :

Une substitution est une application de X vers $P(F, X)$ presque partout égale à l'identité (i.e. sauf sur un ensemble fini de variables) □

Définition 2 :

Puisque $P(F, X)$ est la F-algèbre libre, σ se prolonge en un morphisme encore noté σ de $P(F, X)$ vers $P(F, X)$ □

Définition 3 :

On définit un *ordre sur* $P(F, X)$ de la façon suivante , $e1 \leqslant e2$ s'il existe une substitution σ telle que $\sigma e1 = e2$

L'ordre s'étend par extensionalité à l'ensemble des substitutions en posant

$$\sigma \leqslant \tau \iff \forall e \in P(F, X) \quad \sigma e \leqslant \tau e$$

que l'on peut montrer être équivalent à

$$\sigma \leqslant \tau \iff \exists \rho \quad \tau = \rho\sigma$$

Un *automorphisme* de $P(F, X)$ étend une substitution de X dont le codomaine est inclus dans X

$e1 \equiv e2 \iff \exists\ \xi$ permutation $\xi\ e1 = e2$

on appelle aussi cela une *permutation des variables*.
et

$\sigma \equiv \sigma \iff \exists\ \xi$ permutation $\sigma' = \xi\sigma$

$\iff \forall\ e \in P(F, X) \quad \sigma e \equiv \sigma\ e'$

$< P(F, X) / \equiv\ ; \leqslant >$ est un inf demi-treillis bien-fondé c'est à dire que $\leqslant$ est un ordre et que tout couple de classes de termes (e1, e2) admet une borne inférieure et il n'y a pas de chaines infinies décroissantes. Ainsi tout ensemble majoré admet une borne supérieure. Unifier des classes de termes e1 et e2 c'est trouver le majorant des classes de [e1] et [e2] c'est à dire c'est trouver [e1] v [e2] , s'il existe , et aussi la substitution telle que $\sigma e1 = e1 \vee e2$ et $\sigma e2 = e1 \vee e2$, où $e1 \vee e2$ est un représentant de la classe . Cette substitution est appelée le *plus général unificateur de* e1 et e2 . En fait, seule la substitution suffit. Unifier e1 et e2 c'est aussi d'une certaine manière résoudre l'équation $e1 = e2$.

2.2.- LA PROCÉDURE UNIFIER UNE PREMIÈRE FORME DE L'UNIFICATION.

On va proposer un algorithme dans le cas d'opérations d'arité 1 et 2 puisque c'est ainsi qu'il a été codé.

La généralisation ne présenterait aucune autre difficulté que technique.


```
UNIFIER (r1, r2) =
    cas  r1 = s1 op t1  et  r2 = s2 op t2  où op ∈ {; , &, U}
         alors  SUP(UNIFIER(s1, s2), UNIFIER(t1, t2))
    cas  r1 = s1' et r2 = s2' alors  UNIFIER(s1, s2)
    cas  r1 ∈ C et r2 ∈ C  et r1 = r2 alors  IN
    cas  r1 ∈ Var et  r2  ∈ Var alors  AF(r1, r2)
    cas  r1 ∈ Var et  r2 ∉ Var
         alors si  r1  n'apparaît pas dans  r2
            alors  AF(r1, r2)
            sinon  ECHEC
    cas  r1 ∉ Var et  r2 ∈ Var
         alors si  r2  n'apparaît pas dans r1
            alors  AF(r2, r1)
            sinon  ECHEC
    autrement  ECHEC
```

Figure 1 : Algorithme UNIFIER

Les types utilisés dans cet algorithme sont :

- Terme, c'est l'ensemble des termes représentant les classes de termes. Puisqu'il s'agit de représentants quelconques de classes, on a le choix du nom des variables, aussi on choisit pour chaque terme un ensemble de variables différent de l'ensemble des variables de tout terme que l'on cherche à unifier avec lui. Anticipant sur la suite, on peut signaler que ce ne sont pas les termes qui possèdent leur propre ensemble de variables mais les assertions, cet ensemble est commun à plusieurs termes et disjoint de l'ensemble des variables de tout autre assertion.

- Sub, c'est l'ensemble des substitutions .

Les opérations sur ces types sont :

IN : () → Sub

qui est la substitution nulle part définie

AF : (Var, Terme) → Sub

où AF(x, t) est la substitution définie seulement en x où elle vaut t

SUP : (Sub U {ECHEC} , Sub U {ECHEC}) → Sub U {ECHEC}
est une fonction qui, à deux substitutions (différentes de ECHEC) fait correspondre la borne supérieure de ces substitutions si elle existe sinon elle prend la valeur ECHEC.

Var : (Terme) → Bool
est une opération postfixée qui prend la valeur VRAI si elle s'applique à un terme qui est une variable et FAUX sinon

2.3.- UNE PREMIÈRE VERSION DE UNIF.

Pour obtenir une forme plus efficace de l'algorithme , on cherche à obtenir une procédure où l'appel récursif que l'on rencontre dans le cas où r1 = s1 op t1 et r2 = s2 op t2 se trouvent en position récursive terminale ; pour cela , on définit une fonction UNIF qui possède un paramètre de type substitution. Son profil est donc

UNIF : (Terme, Terme, Sub U {ECHEC}) → Sub U {ECHEC}

sa description est donnée par la figure 2.
Notons que sur le type Sub de nouvelles opérations sont définies, on a :

- AFF : (sub U {ECHEC} , Var, Terme) → Sub U {ECHEC}

AFF (σ, x, t) est la substitution obtenue en affectant le terme t , en x , à la substitution σ .
AFF(ECHEC, x, t) = ECHEC

- IN : () → Sub
Remarquons que AF(x, t) = AFF(IN, x, t)

- EVAL : (Terme, Sub U {ECHEC}) = Terme U {⊥}

si $t \neq \perp$ et $\sigma \neq$ ECHEC alors EVAL(t, σ) est le terme obtenue en substituant les valeurs associées aux variables par σ s'il y a de telles valeurs.

EVAL(⊥, σ) = EVAL(t, ECHEC) = ⊥

2.4.- JUSTIFICATION DE LA PREMIÈRE VERSION DE UNIF

Les références entre accolades se rapportent à la figure 2

{1} Puisque le même opérateur apparaît en position la plus externe de r1 et r2, le résultat de l'unification de r1 et r2 dans le contexte de la substitution σ est le résultat de l'unification de t1 et t2 dans le contexte de la substitution UNIF(s1, s2, σ)

{2} Puisque l'opérateur ' apparaît dans r1 et r2 le résultat de UNIF(r1, r2, σ) est UNIF(s1, s2, σ)

{3} r1 et r2 sont la même constante donc l'unification est σ

{4} r1 et r2 sont des variables

{41} puisque r1 = r2 le résultat est σ

{421} les valeurs associées à r1 et r2 par σ sont deux termes v1 et v2

{4211} on peut unifier les deux termes puisque r2 n'apparaît pas dans v1 et r1 n'apparaît pas dans v1 et le résultat est π , on évalue v1 et v2 en se servant de π (on doit d'ailleurs trouver le même résultat) , et on affecte cela à π en r1 et r2.

{4212} on ne peut unifier, c'est un échec

{422} σ n'est pas défini en r2

{4221} on affecte v1 à σ en r2 puisque r2 ne contient pas v1

{423} symétrique de {422}

{424} σ n'est défini ni en r1 ni en r2 on affecte le terme r2 à r1

{5} r1 est une variable et r2 un terme

{51} r1 n'apparaît pas dans r2

UNIF (r1, r1, σ) =

{1} *cas* r1 = s1 op t1 *et* r2 = s2 op t2 *où* (op = ; *ou* op = & *ou* op = U)
alors UNIF(t1, t2, UNIF(s1, s2, σ))

{2} *cas* r1 = s1' *et* r2 = s2' *alors* UNIF(s1, s2, σ)

{3} *cas* r1 ∈ C *et* r2 ∈ C *et* r1 = r2 *alors* σ

{4} *cas* r1 ∈ Var *et* r2 ∈ Var

{41} *alors si* r1 = r2 *alors* σ

{42} *sinon* {421} *cas* v1 ∈ Terme *et* v2 ∈ Terme

alors si r2 n'apparaît pas dans v1 *et* r1 n'apparaît pas dans v2

{4211} *alors* AFF(AFF(π , r2, EVAL(v1, π)), r2, EVAL(v2, π)) *où* π = UNIF(v1, v2, σ)

{4212} *sinon* ECHEC

{422} *cas* v1 ∈ Terme *et* v2 = ⊥

alors{4221} *si* r2 n'apparaît pas dans v1 *alors* AFF(σ, r2, v1)

{4222} *sinon* ECHEC

{423} *cas* v1 = ⊥ *et* v2 ∈ Terme

alors si r1 n'apparaît pas dans v2 *alors* AFF(σ, r1, v2)

sinon ECHEC

{424} *cas* v1 = ⊥ *et* v2 = ⊥

alors AFF(σ1, r1, r2)

{5} *cas* r1 ∈ Var *et* r2 ∉ Var

alors si r1 n'apparaît pas dans r2

Figure 2 : 1ère partie

{51} *alors* {511} *si* v1 = *alors* AFF(σ, r1, r2)

{512} *sinon si* ρ ≠ ECHEC

alors AFF(ρ, r1, EVAL(v1, ρ)) *où* ρ = UNIF(r2, v1, σ)

sinon ECHEC

{52} *sinon* ECHEC

{6} *cas* r1 ∉ Var *et* r2 ∈ Var

alors si r2 n'apparaît dans r1

alors si v2 = ⊥ *alors* AFF(σ, r2, r1)

sinon si τ ≠ ECHEC

alors AFF(τ, r2, EVAL(v2, τ)) *où* τ = UNIF(r1, v2, σ)

sinon ECHEC

sinon ECHEC

{7} *autrement* ECHEC

où v1 = VAL(σ, r1)

v2 = VAL(σ, r2)

Figure 2 : Algorithme UNIF dans sa première version.

{511} aucune valeur n'est affectée à r1 par σ on y affecte r2

{512} on unifie dans le contexte de σ la valeur affectée à r1 par σ avec r2 et s'il n'y a pas échec on trouve la valeur ρ on y affecte EVAL(v1, p) en r1, c'est le résultat

{6} symétrique de {5}

{7} tous les cas où une unifcation était possible ont été envisagé, c'est un échec.

2.5.- UNE DEUXIÈME VERSION DE UNIF .

L'examen de la première version de UNIF appelle les commentaires suivants qui vont nous permettre d'améliorer sensiblement l'algorithme.

- à la ligne 4211, est-il utile d'évaluer v2 en π pour affecter le résultat de l'évaluation en r2 à π précisément ?
On doit pouvoir affecter seulement v2 à π et retrouver par les autres valeurs de π l'évaluation de v2 . Pour ce faire, un pointeur vers le sous terme v2 suffira, il est inutile de recopier v2 ou son évaluation.

- à la ligne 4211 et à la ligne 424, on voit que les opérations font coïncider les valeurs affectées à v1 et v2, autrement dit il est intéressant de regrouper dans ce cas les variables en classes d'équivalences ce qui va, en moyenne, diminuer les cas à envisager, en effet 41 devient un test d'identité de classes d'équivalence et non plus de variables

De cela, on déduit une représentation des substitutions par des couples (p, a) où p est une partition de l'ensemble des variables en classes d'équivalences et a est une attribution de ⊥ ou d'un terme non complètement évalué à chaque classe. La définition de UNIF est donnée par la figure 3. Son profil est

UNIF : (Terme, Terme, Partition, Attribution) → (Partition, Attribution)

- Le type Partition [Var] est constitué des partitions ou classes d'équivalence de variables dont les opérations principales sont (cf [AHO 74])

FIND : (Partition, Var) → Nom

qui fournit un nom associé à une classe dans une partition donnée.

UNION: (Partition, Nom, Nom) → Partition

qui fournit la partition obtenue à partir d'une partition donnée en unissant deux classes.

Dans le cas présent, FIND vérifie l'identité :

FIND(UNION(p, n, n'), x) =
  si FIND(p, x) = n' *alors* n
  sinon FIND(p, x)

Cette équation n'est peut être pas la meilleure car elle n'est pas symétrique en n et en n' ; par contre elle correspond à une implantation facile à mettre en oeuvre.

LPF : () → Partition

est la partition la plus fine, qui correspond par conséquent à l'égalité

EG(x, y) = EG(FIND(LPF, x), FIND(LPF, y))

c'est à dire x = y si et seulement si leurs classes sont égales. Dans notre implantation les noms sont des variables, représentants privilégies de leur classe.

- Attribution est constitué d'attributions de termes ou de ⊥ à des classes de variables.

AT : (Nom, Attribution) → Terme ∪ {⊥}

fournit le terme attaché à une classe par une attribution donnée, s'il y en a, sinon ⊥ .

MODIF : (Attribution, Nom, Terme) → Attribution

MODIF(a, n, r) transforme une attribution a en affectant le terme r à la classe dont le nom est n

AT(n, MODIF(a, n', r)) = *si* n = n' *alors* r
sinon AT(n, a)

NOCONT : (Terme, Nom, Attribution) → Bool

NOCONT(t, n, a) rend la valeur VRAI si t ainsi que les termes attribués par l'attribution a aux variables que t contient (et ce de façon itérée) ne contiennent pas d'occurrence de variables appartenant à la classe de nom n .

RIEN : () → Attribution

vérifie

AT(RIEN, n) = ⊥

Enfin l'algorithme utilise la fonction

∈ Terme : (Terme U {⊥}) → Bool

x ∈ Terme a la valeur VRAI si x ≠ ⊥ et FAUX sinon

2.6.- JUSTIFICATION DE LA DEUXIÈME VERSION DE UNIF

Les références entre accolades se reportent aux lignes de l'algorithme de la figure 3.

{1} On fait appel deux fois récursivement à UNIF. Donc, si UNIF(s1, s2, p, a) fournit le couple (p', a') résultat correct de l'unification de s1, s2 dans le contexte de la substitution décrite par le couple (p, a) et si UNIF(t1, t2, p', a') est aussi correcte, alors UNIF(s1 ; t1, s2 ; t2, p, a) est correcte.

{2} est aussi un appel récursif à UNIF qui se traite comme {1}

{3} r1 et r2 sont des constantes égales, donc l'unification est un succès et la substitution reste décrite par (p, a)

{41} r1 et r2 sont des variables appartenant à la même classe dans la partition p , l'unification est un succès et la substitution reste décrite par (p, a) .

{421} r1 et r2 sont des variables appartenant à deux classes désignées par f1 et f2 auxquelles sont attribuées des termes AT(f1, a) et AT(f2, a)

{4211} Les termes AT(f1, a) et AT(f2, a) ne contiennent aucune variables des classes de r2 et r1 respectivement. Ainsi le résultat de l'unification de r1 et r2 est la substitution résultant de l'union de la classe de r1 à celle de r2 dans le contexte de la substitution obtenue en unifiant AT(f1,a) et AT(f2,a). Il est bien entendu que cette dernière unification ne peut unir les classes f1 et f2.

{4212} Si le terme AT(f1, a) contient une variable, disons x2 , de la classe f2, l'unification est un échec, car on aboutirait à une substitution paradoxale x2 : = AT(f1, a)(x2). Il en est de même pour l'autre alternative

```
UNIF(r1, r2, p, a) =
{1} cas (r1 = s1 ; t1 et r2 = s2 ; t2) ou (r1 = s1 U t1 et r2 = s2 U t2) ou (r1 = s1 & t1 et r2 = s2 & t2)
      alors UNIF(t1, t2, UNIF(s1, s2, p, a))
{2} cas r1 = s1' et r2 = s2'    alors UNIF(s1, s2, p, a)
{3} cas r1 ∈ C et r2 ∈ C et r1 = r2   alors (p, a)
{4} cas r1 ∈ Var et r2 ∈ Var
     {41} alors si  f1 = f2   alors(p, a)
     {42} sinon{421} cas AT(f1, a) ∈ Terme et AT(f2, a) ∈ Terme
          alors si  NOCONT (AT(f1, a), f2, a) et  NOCONT(AT(f1, a), f2, a)
                {4211} alors (UNION(p1, f1, f2), a1)  où  (p1, a1) = UNIF(AT(f1, a), AT(f2, a), p, a)
                {4212} sinon  ECHEC
          {422} cas  AT(f1, a) ∈  Terme et  AT(f2, a) = ⊥
               alors {4221}  si  NOCONT(AT(f1, a), f2, a) alors  (UNION(p, f1, f2), a)
                     {4222} sinon ECHEC
          {423} cas  AT(f1, a) = ⊥  et  AT(f2, a) ∈  Terme
               alors si NOCONT(AT(f2, a), f1, a)  alors  (UNION(p, f1, f2), a)
                    sinon  ECHEC
          {424} cas AT(f1, a) = ⊥  et  AT(f2, a) = ⊥
                alors (UNION(p, f1, f2), a)
{5} cas r1 ∈ Var et r2 ∉ Var
   alors si NOCONT(r2, f1)
```

Figure 3 : 1ère partie

```
     {51}  alors      {511}  si  AT(f1, a) = ⊥   alors(p, MODIF(a, f1, r2))
                      {512}  sinon si (p1, a1) ≠ ECHEC
                                  alors (p1, MODIF(a1, f1, r2)
                                  sinon ECHEC          où (p1, a1) = UNIF(r2, AT(f1, a), p, a)
     {52}  sinon  ECHEC
{6} cas r1 ∉ Var et  r2 ∈ Var
     alors si  NOCONT(r1, f2)
          alors si   AT(f2, a) = ⊥  alors (p, MODIF(a, f1, r2))
               sinon si  (p2, a2) ≠ ECHEC
                         alors  (p2, MODIF(a2, f1, r2))
                         sinon  ECHEC       où  (p2, a2) = UNIF(r1, AT(f2, a), p, a)
                    sinon   ECHEC
{7}   autrement   ECHEC
                    où   f1 = FIND(p, r1)
                         f2 = FIND(p, r2)
```

Figure 3 : Algorithme UNIF dans la version où les substitutions sont représentées avec partition et attribution

{422} Aucun terme n'est attribué à f2 ; l'unification est l'union de f1 et f2 , si AT(f1, a) ne contient aucune variable de la classe f2.

{423} Est le symétrique de {422}

{424} Aucun terme n'étant attribué à f1 et à f2 , l'unification est obtenue en unissant ces classes

{5} r1 est une variable et r2 est un terme non variable

{511} Aucun terme n'est attribué à la classe f1 de r1, et r2 ne contient aucune variable de f1, donc l'unification est obtenue en attribuant r2 à la classe de r1.

{512} En unifiant le terme AT(f1, a) attribué à f1 et r2, on obtient une substitution (p1 , a1) ; si (p1, a1) est un échec alors l'unification de r1 et r2 est un échec, sinon on attribue r2 à f1, car l'attribution de f1 n'a pu être modidiée par l'unification car ni r2 ni AT(f1, a) ne contiennent de variables de la classe f1.

{52} Est un échec puisque r2 contient une variable de la classe r2

{6} se traite comme {5}

{7} Tous les cas où une unification était envisageable ont été épuisés ; l'unification est un échec.

2.7.- COMMENTAIRES.

L'algorithme décrit ici est très similaire à celui proposé par Huet dans sa thèse [HUE 76] bien qu'ayant été analysé indépendamment. Signalons cependant quelques différences :

- la recherche se fait en profondeur d'abord dans les termes.

- l'implantation des partitions n'utilise pas l'algorithme de Fischer-Galler en ce quiconcerne UNION. En effet, c'est toujours le paramètre gauche qui servira à désigner la classe, au lieu de faire intervenir la cardinalité des

classes. Cela économise un champ dans les structures décrivant une variable, sans enlever le comportement linéaire moyen, semble-t-il. (voir [AHO 74] exercice 4.16 et 4.17 et la thèse de HUET page 5.50).

- une présentation avec des types abstraits donne une exposition claire.

Il est cependant intéressant de voir comment on peut justifier le passage de la première à la deuxième version de UNIF.

3.- RÉSOLUTION.

Dans la suite , les seules assertions que l'on considère sont de la forme : p1, ..., pn → q où p1, ..., pn est une suite éventuellement vide de termes et q est un terme qui peut être absent. Chaque assertion possède son propre ensemble de variables, différent de celui de tout autre assertion ; la notation externe de ces variables est *X1, *X2, *X3,... si bien qu'une variable *Xi qui figure dans deux assertions différentes, repère en fait deux variables différentes (ceci est surtout valable pour comprendre les exemples)

Faire la *résolution* de l'assertion

p1, ..., pn → q

avec l'assertion

r1, ..., rm → u

consiste à trouver un terme ri qui peut être unifié avec q par la substitution σ et à construire l'assertion

σp1, ..., σpn, σr1, ..., σri-1, σri+1, .. , σrm → σu

Pratiquement on construit une assertion où l'on a éliminé les prédicats redondant ou triviaux (par exemple t = t en partie gauche).

4.- RÉFUTATION.

Les hypothèses sont présentées comme une famille d'assertions composée de deux parties :

- une axiomatisation des opérations de la structure algébrique
- une axiomatisation des propriétés des prédicats d'égalité et d'inclusion.

Le résultat à prouver conduit une négation qu'il faut réfuter, cette négation s'écrit

$p1, \ldots, pn \rightarrow$

Exemple 1 :

Supposons que l'on veuille prouver

$*X = *Y \ , *Y = *Z \rightarrow *X = *Z$

Cela est équivalent à prouver

$\forall * X , \forall * Y , \forall * Z \quad *X = *Y, *Y = *Z \rightarrow *X = *Z$

sa négation est

$\exists A \quad \exists B \quad \exists D \qquad A = B \wedge B = D \wedge \neg A = D$

sous la forme de clauses de Horn, cela revient à ajouter les hypothèses

$\rightarrow A = B$

$\rightarrow B = D$

et à refuter

$A = D \rightarrow \quad \square$

Le plus souvent n vaut 1 . Le but de la réfutation est d'engendrer l'assertion vide. La procédure de sélection, proposée ici, essaie les assertions dans l'ordre dans lequel on les a écrites, jusqu'à créer par résolution une nouvelle assertion. On itère le processus avec cette nouvelle assertion. Différentes observations conduisent à des retours arrière :

absence d'assertion à résoudre, trop grande profondeur de la réfutation, assertion ayant un trop grand nombre de variables ou d'opérateurs, assertion "contenant" une assertion appartenant déjà à la réfutation. Ainsi certains paramètres fixés par l'opérateur fournissent au programme des moyens d'éviter des recherches dans des directions qui paraissent a priori des impasses.
Le souci essentiel est d'éviter les recherches exhaustives (ou back-tracking) qui engendrent une explosion exponentielle du temps de preuves.
Le meilleur des cas est celui ou les retours arrière sont limités au maximum (un pas ou deux).

Exemple 2 :

Preuve de la transitivité de l'égalité à partir de celle de l'inclusion.
Deux preuves sont données aux figures 4 et 5. On observera deux différences:

- une profondeur de 12 et 51 assertions engendrées pour la première
- une profondeur de 30 et 257 assertions engendrées pour la seconde.

Les différences proviennent du choix de la profondeur maximum de recherche.

- Dans le premier cas, 12 ; c'est le choix optimum, car il conduit à un échec
- Dans le deuxième cas, 999, autrement dit, aucune limitation pratique.

On remarquera que les deux démonstrations divergent à partir de la 9è assertion et qu'il faut 4 étapes pour refuter $D = *X_0 \ , X_0 \subset A$ dans le premier cas contre 22 étapes dans le second.

Les autres paramètres ont été choisis ainsi :

- 1 variable au plus par assertion ; en acceptant deux variables au plus, on créérait inutilement de nouvelles variables qu'il faudrait ensuite éliminer, retardant la refutation.

```
RECHERCHE EN PROFONDEUR JUSQU'A  12
TOUTE ASSERTION CONTENANT PLUS DE  2 VARIABLES OU PLUS DE  4  PREDICATS SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 4.00  FOIS PLUS D'OPERATEURS QUE LA PREMIERE SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 4.00  FOIS PLUS D'OPERATEURS QUE LA PRECEDENTE SERA SUPPRIMEE

@ PREUVE DE LA TRANSITIVITE DE L'EGALITE A PARTIR DE CELLE DE L'INCLUSION @
 @ DEUX AFFIRMATIONS @
 -> A=B;
 -> B=D;
 @ TRANSITIVITE DE L'INCLUSION @
 *X=C=*Y , *Y=C=*Z -> *X=C=*Z ;
 @ RAPPORTS EGALITE INCLUSION @
 *X=*Y -> *X=C=*Y ;
 *X=*Y -> *Y=C=*X ;
 *X=C=*Y , *Y=C=*X -> *X=*Y .

 @ A REFUTER @
 A=D ->.
DEBUT DE LA REFUTATION

ON PART DE
A = D ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X0 -> *X0 = *X1 ;
ON OBTIENT LA   2 IEME ASSERTION
A =C= D , D =C= A ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X2 -> *X0 =C= *X2 ;
ON OBTIENT LA   3 IEME ASSERTION
A =C= *X0 , *X0 =C= D , D =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA   4 IEME ASSERTION
A = *X0 , *X0 =C= D , D =C= A ->  ;

EN UTILISANT
 -> A = B ;
ON OBTIENT LA   5 IEME ASSERTION
B =C= D , D =C= A ->  ;
```

-188-

Figure 4

```
B =C= *X0 , *X0 =C= B , D =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA   7 IEME ASSERTION
B = *X0 , *X0 =C= D , D =C= A ->  ;

EN UTILISANT
 -> B = D ;
ON OBTIENT LA   8 IEME ASSERTION
D =C= A ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X2 -> *X0 =C= *X2 ;
ON OBTIENT LA   9 IEME ASSERTION
D =C= *X0 , *X0 =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X1 =C= *X0 ;
ON OBTIENT LA  10 IEME ASSERTION
*X0 = D , *X0 =C= A ->  ;

EN UTILISANT
 -> B = D ;
ON OBTIENT LA  11 IEME ASSERTION
B =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X1 =C= *X0 ;
ON OBTIENT LA  12 IEME ASSERTION
A = B ->  ;

IL SUFFIT DE L'ASSERTION
 -> A = B ;
POUR PROUVER LE THEOREME EN  12 ETAPES
**************************************


  51  ASSERTIONS ONT ETE ENGENDREES

 LA PREUVE A DURE          2  SECONDES
```

-189-

Figure 4

```
RECHERCHE EN PROFONDEUR JUSQU'A 999
TOUTE ASSERTION CONTENANT PLUS DE  2 VARIABLES OU PLUS DE  4  PREDICATS SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 4.00  FOIS PLUS D'OPERATEURS QUE LA PREMIERE SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 4.00  FOIS PLUS D'OPERATEURS QUE LA PRECEDENTE SERA SUPPRIMEE

@ PREUVE DE LA TRANSITIVITE DE L'EGALITE A PARTIR DE CELLE DE L'INCLUSION @
 @ DEUX AFFIRMATIONS @
 -> A=B;
 -> B=D;
 @ TRANSITIVITE DE L'INCLUSION @
 *X=C=*Y , *Y=C=*Z -> *X=C=*Z ;
 @ RAPPORTS EGALITE INCLUSION @
 *X=*Y -> *X=C=*Y ;
 *X=*Y -> *Y=C=*X ;
 *X=C=*Y , *Y=C=*X -> *X=*Y .

 @ A REFUTER @
 A=D ->.

DEBUT DE LA REFUTATION

ON PART DE
A = D ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X0 -> *X0 = *X1 ;
ON OBTIENT LA   2 IEME ASSERTION
A =C= D , D =C= A ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X2 -> *X0 =C= *X2 ;
ON OBTIENT LA   3 IEME ASSERTION
A =C= *X0 , *X0 =C= D , D =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA   4 IEME ASSERTION
A = *X0 , *X0 =C= D , D =C= A ->  ;

EN UTILISANT
 -> A = B ;
ON OBTIENT LA   5 IEME ASSERTION
B =C= D , D =C= A ->  ;

ON OBTIENT LA   6 IEME ASSERTION
B =C= *X0 , *X0 =C= D , D =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA   7 IEME ASSERTION
B = *X0 , *X0 =C= D , D =C= A ->  ;

EN UTILISANT
 -> B = D ;
ON OBTIENT LA   8 IEME ASSERTION
D =C= A ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X2 -> *X0 =C= *X2 ;
ON OBTIENT LA   9 IEME ASSERTION
D =C= *X0 , *X0 =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  10 IEME ASSERTION
D = *X0 , *X0 =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  11 IEME ASSERTION
*X0 = A , D = *X0 ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X0 -> *X0 = *X1 ;
ON OBTIENT LA  12 IEME ASSERTION
D =C= *X0 , *X0 =C= D , *X0 = A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  13 IEME ASSERTION
*X0 = D , D =C= *X0 , *X0 = A ->  ;

EN UTILISANT
 -> B = D ;
ON OBTIENT LA  14 IEME ASSERTION
D =C= B , B = A ->  ;
```


Figure 5

Figure 5

```
EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X2 -> *X0 =C= *X2 ;
ON OBTIENT LA  15 IEME ASSERTION
D =C= *X0 , *X0 =C= B , B = A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  16 IEME ASSERTION
D = *X0 , *X0 =C= B , B = A ->  ;

EN UTILISANT
*X0 = *X1 -> *X1 =C= *X0 ;
ON OBTIENT LA  17 IEME ASSERTION
B = *X0 , D = *X0 , B = A ->  ;

EN UTILISANT
 -> B = D ;
ON OBTIENT LA  18 IEME ASSERTION
B = A ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X0 -> *X0 = *X1 ;
ON OBTIENT LA  19 IEME ASSERTION
B =C= A , A =C= B ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X2 -> *X0 =C= *X2 ;
ON OBTIENT LA  20 IEME ASSERTION
B =C= *X0 , *X0 =C= A , A =C= B ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  21 IEME ASSERTION
B = *X0 , *X0 =C= A , A =C= B ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  22 IEME ASSERTION
*X0 = A , B = *X0 , A =C= B ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  23 IEME ASSERTION
A = B , *X0 = A , B = *X0 ->  ;

EN UTILISANT
 -> A = B ;
ON OBTIENT LA  24 IEME ASSERTION
*X0 = A , B = *X0 ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X0 -> *X0 = *X1 ;
ON OBTIENT LA  25 IEME ASSERTION
*X0 =C= A , A =C= *X0 , B = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  26 IEME ASSERTION
A = *X0 , *X0 =C= A , B = *X0 ->  ;

EN UTILISANT
 -> A = B ;
ON OBTIENT LA  27 IEME ASSERTION
B =C= A ->  ;

EN UTILISANT
*X0 =C= *X1 , *X1 =C= *X2 -> *X0 =C= *X2 ;
ON OBTIENT LA  28 IEME ASSERTION
B =C= *X0 , *X0 =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X0 =C= *X1 ;
ON OBTIENT LA  29 IEME ASSERTION
B = *X0 , *X0 =C= A ->  ;

EN UTILISANT
*X0 = *X1 -> *X1 =C= *X0 ;
ON OBTIENT LA  30 IEME ASSERTION
A = *X0 , B = *X0 ->  ;

IL SUFFIT DE L'ASSERTION
 -> A = B ;
POUR PROUVER LE THEOREME EN  30 ETAPES
**************************************


237  ASSERTIONS ONT ETE ENGENDREES

LA PREUVE A DURE          7  SECONDES
```


Figure 5

Figure 5

- 3 prédicats par assertion ; on limite la longueur des assertions ; 3 est d'ailleurs le minimum pour ne pas aboutir à un échec

- le nombre d'opérateurs (y compris les constantes) a été fixé à 4 fois celui de la première ou de la précédente. Ce paramètre ne joue pas un rôle important ici. 4 a paru raisonnable

Notons que si on permute les assertions "transitivité de l'inclusion" et "rapports inclusion-égalité", on ralentit la preuve. Le théorème est prouvé en 23 étapes et 41 secondes et 1140 assertions sont engendrés, pour une profondeur maximum fixée à 999. Cela s'explique par le fait qu'on a intérêt , dans la preuve, à utiliser le plus tôt possible la transitivité donc à la disposer avant les autres □

Il est possible de doner une interprétation opérationnelle des assertions de la forme p1, ..., pn → q . Si un terme a la forme q dans une assertion a réfuter, il est possible de le remplacer par des termes ayant la forme de p1, ..., pn . Cela ressemble à un appel de la procédure q , dont le corps est p1, ..., pn .

5.- Le cas des groupes.

Comme cela arrive souvent dans les problèmes de preuve automatique, les premiers essais ont montré qu'un système aussi général n'avait aucune chance de prouver en un temps raisonnable même les plus simples des lemmes de l'algèbre relationnelle, à moins d'une disposition judicieuse des axiomes. Pour cela, il faut avoir des connaissances sur le comportement du système. Afin de dégager les principaux problèmes, il est raisonnable d'étudier une structure algébrique plus simples ; les groupes fournissent un bon exemple. On se place dans le cas où le seul prédicat est l'égalité. L'axiomatique faible des groupes (associativité, existence d'un élément neutre à gauche et d'un inverse à gauche) pas plus que l'axiomatique classique ne constituent un bon système pour faire des preuves. En effet , il y a un trop grand choix sur les règles à appliquer pour "réduire" une formule à démontrer.

Afin de mieux expliquer le comportement des axiomes, écrivons les comme des réécritures, autrement dit distinguons une partie droite et une partie gauche et admettons que g = d s'écrit g → d et signifie "g se réécrit en d" ou "g se déplie en d" [BUR 77a] . Quand on emploie la règle g → d pour transformer un terme de forme d en un terme de forme g , on peut dire que "d se replie en g". Il est difficile de savoir à un moment donné s'il faut déplier ou replier. Ainsi les trois axiomes de l'axiomatique faible des groupes s'écrivent :

(1) $e.x \rightarrow x$

(2) $x.x^{-1} \rightarrow e$

(3) $x.(y.z) \rightarrow (x.y).z$

Le schéma donné dans la figure 6 donne une preuve de x = x.e dans ce contexte.

On voit qu'on a utilisé certaines règles tantôt pour déplier, tantôt pour replier. Ainsi en l'absence d'indication sur le chemin à suivre vers la solution, une preuve nécessite une recherche, par essais et erreurs,fort couteuse. Aussi une solution possible consiste à essayer de simplifier les assertions du fur et à mesure que l'on pregresse dans la réfutation utilisant les travaux sur la réécriture. On peut diviser les assertions qui constituent les hypothèses en trois familles :

- Axiomes similant la réécriture ;

ils se présentent essentiellement sous la forme suivante :

d = *T → g = *T si g → d est une réécriture , si bien que quand on rencontrera un prédicat de la forme g(s) = r dans une assertion on le remplacera par d(s) = r . Pour éviter que l'ordre des axiomes ait une importancce, qu'il serait d'ailleurs difficile de prévoir , on utilise un système confluent de règles de réécritures. Si de plus le système est noethérien, on est certain de ne pas avoir à utiliser ces règles une infinité de fois ([KNB 70] et chapitre 2 paragraphe 1.2 et 1.3)

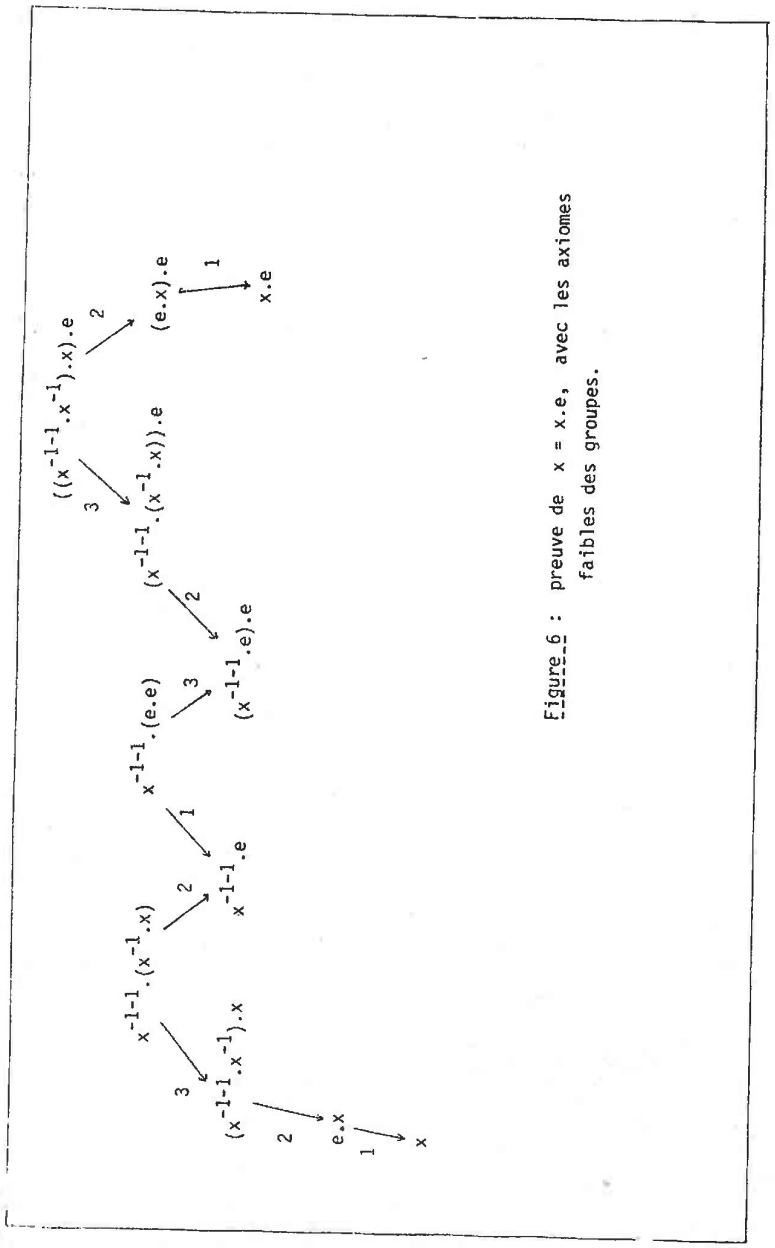


Figure 6 : preuve de x = x.e, avec les axiomes faibles des groupes.

Exemple 3 :

$$*X = *T \rightarrow E\& \quad *X = *T \quad \square$$

- axiomes dit de "décomposition".

Dans une interprétation logique, il s'agit simplement des axiomes de substitution. Dans une interprétation opérationnelle, ils ont pour but de remplacer des tentatives de réécriture sur un terme par des tentatives sur un sous-terme

Exemple 4 :

$$*X = *Y \rightarrow *Z\& \quad *X = *Z\& \quad *Y \quad \square$$

- axiome de réflexivité

$\rightarrow *X = *X$

son interprétation logique est évidente ; dans une interprétation opérationnelle, il tendra à faire disparaître les prédicats dans lesquels on peut unifier le membre droit et le membre gauche.

- axiome de commutativité

considérer la commutatitivé comme une réécriture ordinaire conduit à un système qui n'est pas noethérien. Aussi en le plaçant après les autres axiomes, on est certain que dans la réfutation, on ne l'utilisera que si les autres termes n'ont pu être réécrits. Dans l'exemple, il est combiné avec l'associativité

$$*Y \,\&\, [*Z \,\&\, *X] = *T \rightarrow *X \,\&\, [*Y \,\&\, *Z] = *T$$

- axiome de transitivité

$*X = *T \;,\; *Y = *T \rightarrow *X = *Y$

son interprétation logique est l'une des formes de la transitivité en l'absence de symétrie ; dans une interprétation opérationnelle , il tend à mettre le terme correspondant à Y en partie gauche d'une égalité, afin de pouvoir exécuter sur lui quelques traitements comme la réécriture.

Voici deux exemples (figure 7 et figure 8) traités suivant ce principe. Il s'agit de pourver deux égalités dans les groupes. Les notations adoptées sont :

& pour la loi de composition interne

' pour l'inverse

E pour l'élément neutre

Les réécritures sont extraites de l'article de Knuth et Bendix [KNB 70] p. 279. Les égalités se ressemblent :

[B & A'] &B = [B & C] & [[B & C]' & [A & E]]

et

[A' & B] & B = [B & C] & [[B & C]' & [A & E]]

La première peut être prouvée dans un groupe non commutatif ce qui n'est pas le cas de la seconde. Ainsi on peut comparer les deux réfutations.


```
TOUTE ASSERTION CONTENANT PLUS DE  2 VARIABLES OU PLUS DE  3  PREDICATS SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 1.05  FOIS PLUS D'OPERATEURS QUE LA PREMIERE SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 1.55  FOIS PLUS D'OPERATEURS QUE LA PRECEDENTE SERA SUPPRIMEE

@PREUVE DANS LES GROUPES EN UTILISANT LES DIX AXIOMES DE KNUTH ET BENDIX@
   @ & EST LA LOI DE COMPOSITION INTERNE
     ' EST LA NOTATION DE L'INVERSE
     E EST L'ELEMENT NEUTRE
      *X,*Y,*Z,*T SONT DES VARIABLES,AINSI QUE CHAQUE IDENTI-
   FICATEUR PRECEDE D'UNE ETOILE @
   @AXIOME 1@
    *X = *T  -> E & *X = *T ;
   @AXIOME 2@
    E = *T  ->  [*X'] & *X = *T ;
   @AXIOME 3@
   *X & [*Y & *Z] = *T  ->  [*X & *Y] & *Z = *T ;
   @AXIOME 4@
    *X = *T  ->  [*Y'] & [*Y & *X] = *T ;
   @AXIOME 5 @
    *X = *T  ->  *X & E = *T ;
   @AXIOME 9@
    E = *T  ->  E' = *T ;
   @AXIOME 10 @
    *X = *T  ->  [*X']' = *T ;
   @AXIOME 11@
    E = *T  ->  *X & [*X'] = *T ;
   @AXIOME 13@
   *Y = *T  ->  *X & [[*X'] & *Y] = *T ;
   @AXIOME 20@
    [*X'] & [*X]' = *T  ->  [*X & *X]' = *T ;
   @DECOMPOSITION@
    *X = *Y  ->  *Z & *X = *Z & *Y ;
    *X = *Y  ->  *X & *Z = *Y & *Z ;
    *X = *Y  ->  *X' = *Y' ;
   @REFLEXIVITE@
   -> *X = *X ;
   @COMMUTATIVITE ET ASSOCIATIVITE@
   *Y & [*Z & *X] = *T  ->  *X & [*Y & *Z] = *T ;
   @TRANSITIVITE@
   *X = *T  ,  *Y = *T  ->  *X = *Y .

@A DEMONTRER DANS LE CAS COMMUTATIF@
    [ [[A'] & B]'] & B
  = [B & C] & [ [[B & C]'] & [A & E] ]  ->.
```

Figure 7 : 1ère partie

```
ON PART DE
[[A' & B]' & B] = [[B & C] & [[B & C]' & [A & E]]] -> ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA   2 IEME ASSERTION
[[A' & B]' & B] = *X0 , [[B & C] & [[B & C]' & [A & E]]] = *X0 -> ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [[*X0 & *X1] & *X2] = *X3 ;
ON OBTIENT LA   3 IEME ASSERTION
[B & [C & [[B & C]' & [A & E]]]] = *X0 , [[A' & B]' & B] = *X0 -> ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA   4 IEME ASSERTION
[C & [[B & C]' & [A & E]]] = *X0 , [[A' & B]' & B] = [B & *X0] -> ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA   5 IEME ASSERTION
[[B & C]' & [A & E]] = *X0 , [[A' & B]' & B] = [B & [C & *X0]] -> ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA   6 IEME ASSERTION
[A & E] = *X0 , [[A' & B]' & B] = [B & [C & [[B & C]' & *X0]]] -> ;

EN UTILISANT
*X0 = *X1 -> [*X0 & E] = *X1 ;
ON OBTIENT LA   7 IEME ASSERTION
A = *X0 , [[A' & B]' & B] = [B & [C & [[B & C]' & *X0]]] -> ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA   8 IEME ASSERTION
[[A' & B]' & B] = [B & [C & [[B & C]' & A]]] -> ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA   9 IEME ASSERTION
[[A' & B]' & B] = *X0 , [B & [C & [[B & C]' & A]]] = *X0 -> ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  10 IEME ASSERTION
B = *X0 , [B & [C & [[B & C]' & A]]] = [[A' & B]' & *X0] -> ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA  11 IEME ASSERTION
[B & [C & [[B & C]' & A]]] = [[A' & B]' & B] -> ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [*X2 & [*X0 & *X1]] = *X3 ;
ON OBTIENT LA  12 IEME ASSERTION
[C & [[[B & C]' & A] & B]] = [[A' & B]' & B] -> ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [*X2 & [*X0 & *X1]] = *X3 ;
ON OBTIENT LA  13 IEME ASSERTION
[[[B & C]' & A] & [B & C]] = [[A' & B]' & B] -> ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [[*X0 & *X1] & *X2] = *X3 ;
ON OBTIENT LA  14 IEME ASSERTION
[[B & C]' & [A & [B & C]]] = [[A' & B]' & B] -> ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [*X2 & [*X0 & *X1]] = *X3 ;
ON OBTIENT LA  15 IEME ASSERTION
[A & [[B & C] & [B & C]']] = [[A' & B]' & B] -> ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [*X2 & [*X0 & *X1]] = *X3 ;
ON OBTIENT LA  16 IEME ASSERTION
[[B & C] & [[B & C]' & A]] = [[A' & B]' & B] -> ;

EN UTILISANT
*X0 = *X1 -> [*X2 & [*X2' & *X0]] = *X1 ;
ON OBTIENT LA  17 IEME ASSERTION
A = [[A' & B]' & B] -> ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  18 IEME ASSERTION
A = *X0 , [[A' & B]' & B] = *X0 -> ;
```

Figure 7 : 2ème partie

Figure 7 : 3ème partie

```
EN UTILISANT
*X0 = *X1 -> [*X0 & *X2] = [*X1 & *X2] ;
ON OBTIENT LA  19 IEME ASSERTION
[A' & B]' = *X0 , A = [*X0 & B] ->  ;

EN UTILISANT
[*X0' & *X1'] = *X2 -> [*X1 & *X0]' = *X2 ;
ON OBTIENT LA  20 IEME ASSERTION
[B' & A'']' = *X0 , A = [*X0 & B] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  21 IEME ASSERTION
A'' = *X0 , A = [[B' & *X0] & B] ->  ;

EN UTILISANT
*X0 = *X1 -> *X0'' = *X1 ;
ON OBTIENT LA  22 IEME ASSERTION
A = *X0 , A = [[B' & *X0] & B] ->  ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA  23 IEME ASSERTION
A = [[B' & A] & B] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  24 IEME ASSERTION
A = *X0 , [[B' & A] & B] = *X0 ->  ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [[*X0 & *X1] & *X2] = *X3 ;
ON OBTIENT LA  25 IEME ASSERTION
[B' & [A & B]] = *X0 , A = *X0 ->  ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [*X2 & [*X0 & *X1]] = *X3 ;
ON OBTIENT LA  26 IEME ASSERTION
[A & [B & B']] = *X0 , A = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  27 IEME ASSERTION
[E & B'] = *X0 , A = [A & *X0] ->  ;
```


Figure 7 : 4ème partie

```
EN UTILISANT
E = *X0 -> [*X1 & *X1'] = *X0 ;
ON OBTIENT LA  28 IEME ASSERTION
E = *X0 , A = [A & *X0] ->  ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA  29 IEME ASSERTION
A = [A & E] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  30 IEME ASSERTION
A = *X0 , [A & E] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X0 & E] = *X1 ;
ON OBTIENT LA  31 IEME ASSERTION
A = *X0 ->  ;

IL SUFFIT DE L'ASSERTION
 -> *X0 = *X0 ;
POUR PROUVER LE THEOREME EN  31 ETAPES
**************************************


 66  ASSERTIONS ONT ETE ENGENDREES

LA PREUVE A DURE          4  SECONDES
```


Figure 7 : 5ème partie

```
RECHERCHE EN PROFONDEUR JUSQU'A 999
TOUTE ASSERTION CONTENANT PLUS DE  2 VARIABLES OU PLUS DE  3  PREDICATS SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 1.05  FOIS PLUS D'OPERATEURS QUE LA PREMIERE SERA SUPPRIMEE
TOUTE ASSERTION CONTENANT 1.55  FOIS PLUS D'OPERATEURS QUE LA PRECEDENTE SERA SUPPRIMEE

@PREUVE DANS LES GROUPES EN UTILISANT LES DIX AXIOMES DE KNUTH ET BENDIX@
 @ & EST LA LOI DE COMPOSITION INTERNE
   ' EST LA NOTATION DE L'INVERSE
   E EST L'ELEMENT NEUTRE
   *X,*Y,*Z,*T SONT DES VARIABLES,AINSI QUE CHAQUE IDENTI-
 FICATEUR PRECEDE D'UNE ETOILE @
 @AXIOME 1@
  *X = *T  ->  E & *X = *T ;
 @AXIOME 2@
  E = *T  ->  [*X'] & *X = *T ;
 @AXIOME 3@
  *X & [*Y & *Z] = *T  ->  [*X & *Y] & *Z = *T ;
 @AXIOME 4@
  *X = *T  ->  [*Y'] & [*Y & *X] = *T ;
 @AXIOME 8 @
  *X = *T  ->  *X & E = *T ;
 @AXIOME 9@
  E = *T  ->  E' = *T ;
 @AXIOME 10 @
  *X = *T  ->  [*X']' = *T ;
 @AXIOME 11@
  E = *T  ->  *X & [*X'] = *T ;
 @AXIOME 13@
  *Y = *T  ->  *X & [[*X'] & *Y] = *T ;
 @AXIOME 20@
  [*Y'] & [*X'] = *T  ->  [*X & *Y]' = *T ;
 @DECOMPOSITION@
  *X = *Y  ->  *Z & *X = *Z & *Y ;
  *X = *Y  ->  *X & *Z = *Y & *Z ;
  *X = *Y  ->  *X' = *Y' ;
 @REFLEXIVITE@
 ->  *X = *X ;
 @TRANSITIVITE@
  *X = *T  ,  *Y = *T  ->  *X = *Y .

 @A DEMONTRER DANS LE CAS NON COMMUTATIF@
    [ [B & [A']]'] & B
 =   [B & C] & [ [[B & C]'] & [A & E] ]  ->.
```


Figure 8 : 1ère partie

```
DEBUT DE LA REFUTATION

ON PART DE
[[B & A']' & B] = [[B & C] & [[B & C]' & [A & E]]] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA   2 IEME ASSERTION
[[B & A']' & B] = *X0 , [[B & C] & [[B & C]' & [A & E]]] = *X0 ->  ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [[*X0 & *X1] & *X2] = *X3 ;
ON OBTIENT LA   3 IEME ASSERTION
[B & [C & [[B & C]' & [A & E]]]] = *X0 , [[B & A']' & B] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA   4 IEME ASSERTION
[C & [[B & C]' & [A & E]]] = *X0 , [[B & A']' & B] = [B & *X0] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA   5 IEME ASSERTION
[[B & C]' & [A & E]] = *X0 , [[B & A']' & B] = [B & [C & *X0]] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA   6 IEME ASSERTION
[A & E] = *X0 , [[B & A']' & B] = [B & [C & [[B & C]' & *X0]]] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X0 & E] = *X1 ;
ON OBTIENT LA   7 IEME ASSERTION
A = *X0 , [[B & A']' & B] = [B & [C & [[B & C]' & *X0]]] ->  ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA   8 IEME ASSERTION
[[B & A']' & B] = [B & [C & [[B & C]' & A]]] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA   9 IEME ASSERTION
[[B & A']' & B] = *X0 , [B & [C & [[B & C]' & A]]] = *X0 ->  ;
```


Figure 8 : 2ème partie

```
EN UTILISANT
*X0 = *X1 -> [*X0 & *X2] = [*X1 & *X2] ;
ON OBTIENT LA  10 IEME ASSERTION
[B & A']' = *X0 , [B & [C & [[B & C]' & A]]] = [*X0 & B] ->  ;

EN UTILISANT
[*X0' & *X1'] = *X2 -> [*X1 & *X0]' = *X2 ;
ON OBTIENT LA  11 IEME ASSERTION
[A'' & B'] = *X0 , [B & [C & [[B & C]' & A]]] = [*X0 & B] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  12 IEME ASSERTION
B' = *X0 , [B & [C & [[B & C]' & A]]] = [[A'' & *X0] & B] ->  ;

EN UTILISANT
*X0 = *X1 -> *X0' = *X1' ;
ON OBTIENT LA  13 IEME ASSERTION
B = *X0 , [B & [C & [[B & C]' & A]]] = [[A'' & *X0'] & B] ->  ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA  14 IEME ASSERTION
[B & [C & [[B & C]' & A]]] = [[A'' & B'] & B] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  15 IEME ASSERTION
[B & [C & [[B & C]' & A]]] = *X0 , [[A'' & B'] & B] = *X0 ->  ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [[*X0 & *X1] & *X2] = *X3 ;
ON OBTIENT LA  16 IEME ASSERTION
[A'' & [B' & B]] = *X0 , [B & [C & [[B & C]' & A]]] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  17 IEME ASSERTION
[B' & B] = *X0 , [B & [C & [[B & C]' & A]]] = [A'' & *X0] ->  ;

EN UTILISANT
E = *X0 -> [*X1' & *X1] = *X0 ;
ON OBTIENT LA  18 IEME ASSERTION
E = *X0 , [B & [C & [[B & C]' & A]]] = [A'' & *X0] ->  ;
```


Figure 8 : 3ème partie

```
-> *X0 = *X0 ;
ON OBTIENT LA  19 IEME ASSERTION
[B & [C & [[B & C]' & A]]] = [A'' & E] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  20 IEME ASSERTION
[B & [C & [[B & C]' & A]]] = *X0 , [A'' & E] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X0 & E] = *X1 ;
ON OBTIENT LA  21 IEME ASSERTION
A'' = *X0 , [B & [C & [[B & C]' & A]]] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> *X0'' = *X1 ;
ON OBTIENT LA  22 IEME ASSERTION
A = *X0 , [B & [C & [[B & C]' & A]]] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  23 IEME ASSERTION
[C & [[B & C]' & A]] = *X0 , A = [B & *X0] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  24 IEME ASSERTION
[[B & C]' & A] = *X0 , A = [B & [C & *X0]] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X0 & *X2] = [*X1 & *X2] ;
ON OBTIENT LA  25 IEME ASSERTION
[B & C]' = *X0 , A = [B & [C & [*X0 & A]]] ->  ;

EN UTILISANT
[*X0' & *X1'] = *X2 -> [*X1 & *X0]' = *X2 ;
ON OBTIENT LA  26 IEME ASSERTION
[C' & B'] = *X0 , A = [B & [C & [*X0 & A]]] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  27 IEME ASSERTION
B' = *X0 , A = [B & [C & [[C' & *X0] & A]]] ->  ;
```


Figure 9 : 4ème partie

```
EN UTILISANT
*X0 = *X1 -> *X0' = *X1' ;
ON OBTIENT LA  28 IEME ASSERTION
B = *X0 , A = [B & [C & [[C' & *X0'] & A]]] ->  ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA  29 IEME ASSERTION
A = [B & [C & [[C' & B'] & A]]] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  30 IEME ASSERTION
A = *X0 , [B & [C & [[C' & B'] & A]]] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  31 IEME ASSERTION
[C & [[C' & B'] & A]] = *X0 , A = [B & *X0] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  32 IEME ASSERTION
[[C' & B'] & A] = *X0 , A = [B & [C & *X0]] ->  ;

EN UTILISANT
[*X0 & [*X1 & *X2]] = *X3 -> [[*X0 & *X1] & *X2] = *X3 ;
ON OBTIENT LA  33 IEME ASSERTION
[C' & [B' & A]] = *X0 , A = [B & [C & *X0]] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  34 IEME ASSERTION
[B' & A] = *X0 , A = [B & [C & [C' & *X0]]] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  35 IEME ASSERTION
A = *X0 , A = [B & [C & [C' & [B' & *X0]]]] ->  ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA  36 IEME ASSERTION
A = [B & [C & [C' & [B' & A]]]] ->  ;
EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  37 IEME ASSERTION
A = *X0 , [B & [C & [C' & [B' & A]]]] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  38 IEME ASSERTION
[C & [C' & [B' & A]]] = *X0 , A = [B & *X0] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & [*X2' & *X0]] = *X1 ;
ON OBTIENT LA  39 IEME ASSERTION
[B' & A] = *X0 , A = [B & *X0] ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & *X0] = [*X2 & *X1] ;
ON OBTIENT LA  40 IEME ASSERTION
A = *X0 , A = [B & [B' & *X0]] ->  ;

EN UTILISANT
 -> *X0 = *X0 ;
ON OBTIENT LA  41 IEME ASSERTION
A = [B & [B' & A]] ->  ;

EN UTILISANT
*X0 = *X1 , *X2 = *X1 -> *X0 = *X2 ;
ON OBTIENT LA  42 IEME ASSERTION
A = *X0 , [B & [B' & A]] = *X0 ->  ;

EN UTILISANT
*X0 = *X1 -> [*X2 & [*X2' & *X0]] = *X1 ;
ON OBTIENT LA  43 IEME ASSERTION
A = *X0 ->  ;

IL SUFFIT DE L'ASSERTION
 -> *X0 = *X0 ;
POUR PROUVER LE THEOREME EN  43 ETAPES
**************************************


 59  ASSERTIONS ONT ETE ENGENDREES

LA PREUVE A DURE          4  SECONDES
```


Figure 8 : 5ème partie

Figure 8 : 6ème partie

6.- CONCLUSIONS ET PERSPECTIVES.

Notre travail nous a conduit aux conclusions suivantes :

1) Un système général, tel quel, n'est pas très efficace ; on a , par conséquent, intérêt à utiliser un système spécialisé de réécriture pour prouver des identités dans le cas où les opérateurs ne sont pas commutatifs. D'autant plus qu'une implantation du filtrage est en général plus efficace que l'unification.

2) Cependant notre système bien "programmé" permet d'aboutir à une réfutation dans le cas commutatif , cas où les systèmes de réécriture classique sont en défaut. Une voie de recherche prometteuse est celle de l'unification dans un contexte d'opérateurs commutatifs et associatifs [STI 75] . Ce domaine ne semble pas encore pouvoir intégrer tout le calcul relationnel.

3) Il reste cependant beaucoup à faire sur l'arrangement des hypothèses afin de conduire à une preuve efficace, sans intervention d'un opérateur. De même qu'on essaie de définir des méthodes pour écrire des programmes, il reste à définir des méthodes pour écrire des directives de preuves [WAR 79] .

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[ABR 74] ABRIAL J.R. :
"Data semanties"
Data Base Management (J. W. Klimbie and K.L. Kofferman Ed)
North Holland (1974) 1.59.

[ABR 78] ABRIAL J.R. :
"Z : a specification language"
Journée d'études Sesori : Synthèse manipulation et transformation de programmes. Saint Rémy de Provence (mai 1978).

[AHO 74] AHO A., HOPCROFT J.E., ULLMAN J.D. :
"The Design and Analysis of Computer Algorithms"
Addison - Wesley Publishing Company (1974).

[ARS 77] ARSAC J. :
"La construction des programmes structurés".
Dunod Paris (1978).

[ASH 77] ASHCROFT E.A., WADGE W.W. :
"LUCID a nonprocedural language with iteration"
Comm. ACM 20 , (1977), 519.526

[BAC 78] BACKUS J.
"Can programming be liberated from the von Neumann Style
A functionnal style and its algebra of programs".
Comm. ACM 21, (1978) , 613.641.

[BAK 72] de BAKKER J., de ROEVER W.P. :
"A calculus for recursive program schemes" in Proc. 1st Symposium on Automata, Language, Programming - M. Nivat (ed.)
North Holland, Amsterdam (1972), 167.196.

[BAU 79] BAUER F.L. , WOSSNER H. :
"Algorithmic language and Program Development"
Prentice Hall International (1979) London.

[BEL 78] BELLEGARDE et al. :
"MEDEE a type of language for constructing programs"
Workshop on reliable software (Bonn 78),
aussi rapport CRIN 78 - P - 074.

[BIR 35] BIRKHOFF G. :
"On the structure of abstract algebras"
Proc. Cambridge Phil. Soc. 31 , (1935) 433.454.

[BIR 70] BIRKHOFF G., and LIPSON J.D. :
"Heterogeneous Algebras"
Journal of Combinatorial Theory, 8, (1970) 115.133.

[BLE 66] BLEICHER M.N. and SCHNEIDER M. :
"Completion of partially ordered sets and Universal algebras".
Acta Math. Acad. Sci. Hungaricae 17, (1966), 271.301

[BLE 73] BLEICHER M.N., SCHNEIDER M. and WILSON R.L. :
"Permanence of identities on algebras"
Algebra Universalis, 3, (1973), 75.93

[BOY 75] BOYER S. et MOORE J.S. :
"Proving Theorems about LISP Functions"
J. Ass. Comp. Mach. 22, (1975) 129.144

[BOY 77] BOYER S. et MOORE J.S. :
"A lemma driven automatic theorem prover for recursive function theory"
5th International joint conference on artificial intelligence (1977) 511.519

[BRO 79] BROY M. , DOSCH W., PARTSCH H., PEPPER P., WIRSING M. :
"Existensial quantifiers in abstract data types"
6th International Colloquium on Automata Languages and Programming. (Gratz Autriche).

[BUR 77a] BURSTALL R.M. and DARLINGTON J. :
"A transformation system for developping recursive programs"
J. of Ass . Comp. Mach.24 (1977), 44.67

[BUR 77b] BURSTALL R.M. and GOGUEN J.A. :
"Putting Theories Together to Make Specifications"
Proceedings of 5th International Joint Conference on Artificial Intelligence (1977) 1045.1058

[CHA 79] CHABRIER J., CHABRIER J.J., DERNIAME J.C., HENRY P., MINOT R., PROCH C. :
"Le langage ATM : Manuel d'utilisation"
CRIN (à paraître)

[CHA 73] CHANG C.L. & LEE R.C.T.
"Symbolic Logic and Mechanical Theorem Proving"
Academic Press, (1973)

[COD 70] CODD E.F. :
"A Relationel Model for Large Shared Data Banks"
Comm. ACM, 13, (1970) 377.387

[CONW 71] CONWAY J.H. :
"Regular Algebras"
Chapman and Hall, (1971) , London.

[COH 65] COHN P.M. :
"Universal Algebra"
Harper and Row, New York (1965)

[COL 75] COLMERAUER A. :
"Les grammaires de métamorphose"
Université de Marseille Luminy (1975)

[FIN 76] FINANCE J.P. :
"Data structures as a framwork to formalize the semantics of a programming language"
Compte rendu du 2ē colloque international sur la programmation (Robinet Ed.) Dunod. Paris (1977) 75.88

[FRA 78] FRANÇON , VUILLEMIN J. , FLAGEOLLET :
"Description et analyse d'une représentation performante des files de priorité"
Rapport Laboria n° 287 , (1978).

[FUC 66] FUCHS L. :
"On partially ordered algebras I"
Colloquium Math. 14 (1965), 115.130

[FUC 65] FUCHS L. :
"On partially ordered algebras II"
Acta Univ.Szegediensis 26, (1965) 34.41

[GAU 77] GAUDEL M.C., PAIR C.
"Les structures de données et leur représentation en mémoire"
IRIA, Rocquencourt, (1977)

[GAU 78] GAUDEL M.C., TERRINE G. :
"Synthèse de la représentation d'un type abstrait par des types concrets"
Théorie et techniques de l'informatique actes du congrès de l'AFCET (1978), t.1, Hommes et Techniques, Paris, 434.445

[GAU 78b] GAUDEL M.C. :
"Spécification incomplète mais suffisante de la représentation des types abstraits"
Rapport Laboria , n° 320 , (1978) , Rocquencourt, France

[GAU 78c] GAUDEL M.C., DESCHAMPS P. , MAZAUD M. :
"Semantics of procedures as an algebraic data type"
Rapport Laboria n° 334 (1978) Rocquencourt, France

[GAU 57] GAUTAM N.D. :
"The validity of complex algebras "
Arch. Math. Logik Grundlagenforsch, 3 (1957), 117.127

[GOG 75] GOGUEN J.A., THATCHER J.W., WAGNER E.G. et WRIGHT J.B. :
"Abstract data types as initial algebras and correctness of data représentations"
Proceeding Conference on Computer Graphics, Pattern Recognition and Data Structure (May 1975),
aussi in "Current Trends in Programming Methodology"4 (Yeh R. Ed)
Prentice Hall Englewood Cliffs, 80.149

[GOG 77] GOGUEN J.A., THATCHER J.W., WAGNER E.G., WRIGHT J.B. :
"Initial algebras Semantics and Continuous Algebras"
J. Assoc. Comput. Mach. 24, (1977), 68.85

[GOG 79] GOGUEN J.A., TARDO J.J. :
"An Introduction to OBJ a language for Writing et testing formal algebraic program specifications"
Proceeding of the specifications of Reliable Software Conference, Boston, (1979).

[GRA 68] GRATZER G. :
"Universal Algebra"
Van Nostrand (1968)

[GRE 65] GREIBACH S. :
"A new normal form theorem for context-free phrase structure grammars"
J. Ass. Comp. Mach., 12, (1965), 42.52

[GUT 77] GUTTAG J.V. :
"Abstract data types and the developpement of data structures"
Comm ACM 20(1977) 397.404

[GUT 78 a] GUTTAG J.V. et HORNING J.J. :
"The Algebraic specification of Abstract Data Types"
Acta Informatica 10, (1978) 27.52

[GUT 78b] GUTTAG J.V. , HOROWITZ E. , MUSSER D. :
"The Design of data type specifications"
in "Current Trends in programming Méthodology"
Vol IV Data structuring (Yeh R. Editor)
Prentice Hall , Englewood Cliffs, (1978)

[GUT 78c] GUTTAG J.V., HOROWITZ E. et MUSSER D.R. :
"Abstract Data Types and Software Validation"
Comm. ACM 21, (1978), 1048.1064

[GUT 79] GUTTAG J. :
"Notes on type abstraction"
Proceeding of the Specifications of Reliable Software Conference, Boston (1979), 36.46.

[HIT 72] HITCHCOCK P. et PARK D. :
"Induction Rules and proofs of termination"
in Proc. 1st Symposium on Automato, Languages and Programming,
M. Nivat (Ed.) North Holland, Amsterdam (1972) 225.252

[HOA 72] HOARE C.A.R. :
"Proof of correctness of data representations"
Acta Informatica 4, (1972), 271.281

[HOP 69] HOPCROFT J.E. et ULLMANN J.D. :
"Formal Languages and their relation to automata"
Addison - Wesley (1969)

[HUE 76] HUET G. :
"Resolution d'équations dans les langages d'ordre 1, 2,..., ω"
Thèse d'Etat de l'Université de Paris 7 (1976).

[HUE 77] HUET G.
"Confluent réductions : abstract properties and applications to term rewriting systems"
Proceedings of the 18th Annual IEEE symposium on Foundations of Computer Science (1977), 30.45

[JON 79] JONES C.B. :
"Constructing a theory of a Data Structure as an aid to program development"
Acta Informatica, 11, (1979), 119.128

[KNB 70] KNUTH D., BENDIX P. :
"Simple word problems in universal algebras"
Computational problems in abstract algebras (Leech Ed.)
Pergamon (1970), 263.297

[KOW 74] KOWALSKI R.A. :
"Predicate logic as programming language"
Proc IFIP 74, North Holland (1974), 569.574

[LAN 77] LANKFORD D.S. :
"On Deciding Word, Problems by Rewrite Rule simplifiers"
Communication au séminaire sur la sémantique. Sophia Antipolis (septembre 1977)

[LEH 77] LEHMAN D.J. et SMYTH M.B. :
"Data Types"
Proc. 18th IEEE Symp on Foundation of Computer sciences (1977) 7.12

[LES 78] LESCANNE P. :
"Algèbres de mots et algèbre universelle"
Colloque AFCET-SMF, tome II, (1978) 147.156

[LIS 74] LISKOV B.H. and ZILLES S.N. :
"Programming with abstract data types"
SIGPLAN Notices 9, 50.59

[LIS 77] LISKOV B. et ZILLES S.
"An introduction to formal specifications of data abstractions"
in Current trends in programming methodology Vol 1
(R. Yeh Ed.) Prentice Hall (1977),1.32.

[LIV 78] LIVERCY C. :
"Théorie des programmes"
Dunod Paris (1978).

[LYN 50] LYNDON R.C.
"The representation of relational algebras"
Ann of Math., 51,(1950), 707.729

[MIN 79] MINOT R. :
"ATM : Un système de fabrication de programme basé sur les concepts de modularité et de type abstrait"
Thèse de 3ème cycle, Université de Nancy 1-CRIN 79-T-031

[MOR 73] MORRIS J.H. :
"Types are not sets"
ACM Symposium on the Principles of Programming Languages, (1973),120.124.

[MUS 77] MUSSER D. :
"A Data type verification system based on rewrite rules"
Proceedings of the Sixth Texas Conference on Computing Systems, Austin Texas, (novembre 1977), 22.31.

[MUS 78] MUSSER D. :
"Convergent sets of rewrite rules for abstract data types"
USC Information Sciences Institute.

[MUS 79] MUSSER D. :
"Abstract Data Type Specification in the AFFIRM system"
Proceeding of the Specifications of Reliable Software Conference, Boston, (1979).

[NIV 75] NIVAT M. :
"On the interpretation of polyadic recursive schemes"
Symp. Mathematica, 15, Academic Press (1975).

[PAI 68] PAIR C. et QUERE A. :
"Définition et étude des bilangages réguliers'
Inf and Control, 13, (1968), 565.593

[PAI 74] PAIR C. :
"Formalization of the notion of data, information and information structure"
in Data base management, J.W. Klimbie and K.L. Koffman (Ed.)
North Holland, (1974).

[PAI 76] PAIR C. :
"Les arbres en théorie des langages"
Centre de Recherche en Informatique de Nancy - CRIN 76-R-028

[PAI 79] PAIR C.
"La construction des programmes"
Revue d'Aut. d'Inf. et de Rech. Op. Informatique, 2, (1979), 113.137.

[PAR 70] PAREIGIS B. :
"Categories and Functors"
Academic Press . New York (1970)

[PAR 76] PARK D. :
"Finiteness is mu ineffable"
Theorical Computer Science, 3, (1976) , 113.131.

[PIE 68] PIERCE R.S. :
"Introduction to the theory of Abstract Algebras"
Holt, Rinehart and Wiston (1968).

[PLA 78] PLAISTED D.A. :
"A Recursively Defined Ordering for Proving Termination of Term Rewriting Systems"
University of Illinois, Report n° UIUCDCS . R-78-943 (1978).

[QUE 69] QUERE A. :
"Etude des ramifications et des bilangages réguliers"
Thèse de 3ème cycle, Université de Nancy, (1969).

[RAO 78] RAOULT J.C., VUILLEMIN J. :
"Operational and semantic equivalence between recursive program"
Rapport n° 9, ERA "AL KHOWARIZMI" Université de Paris Sud, Orsay, France.

[REM 74] REMY J.L. :
"Structures d'information : formalisation des notions d'accès et de modifications d'une donnée"
Thèse de spécialité. Université de Nancy , (1974).

[REM 79] REMY J.L. :
"Construction, Evaluation et amélioration systématique de structures de données".
A paraître dans RAIRO Informatique théorique.

[ROE 74] de ROEVER W.P. :
"Operational, mathematical and axiomatized semantics for recursive procedures and data structures"
Mathematical Center report ID/1 Amsterdam (1974).

[ROE 74] de ROEVER W.P. :
"Recursion and parameter mechanism on Axiomatic approach"
in Proc. 2nd Colloquium on Automata Languages and Programming, J. Loeckx (Ed.) Lectures Notes in computer sciences n°14, springer Verlag, Berlin (1974), 34.65.

[ROU 75] ROUSSEL P. :
"PROLOG : manuel d'utilisation"
Université de Marseille Luminy

[SHA 74] SHAFAAT A. :
"On varieties closed under the construction of power algebras"
Bull. Austral. Math. Soc. , 11, (1974), 213.218.

[STA 78] STANDISH
"Data structures an axiomatique approach"
in "Current Trends in Programming Methodology", IV, Data Structuring, R.T. YEH(Ed.) Prentice Hall, Engl. Cliffs, New Jersey, (1978), 30.60.

[STI 75] STICKEL M.E. :
"A complete unification algorithm for associative-commutative functions"
Proceedings 4th IJCAI, Tbilissi, (1975).

[TAR 41] TARSKI A. :
"On the calculus of relations"
J. Symbolic Logic, 6, (1941), 73.89.

[VDW 36] VAN DER WAERDEN :
"Modern Algebra"
Springer Verlag Berlin , (1936).

[WAR 77] WARREN DHD., PEREIRA L.M., PEREIRA F. :
"Prolog - the language and its implementation compared with Lisp"
Procs ACM Symp on AI and Programming Languages (ANG 77), 109.115.

[WAR 79] WARREN DHD. :
"Logic programming and compiler writing"
DAI Research Report n° 44 , Edimburgh.

[WAN 73] WAND M. :
"Mathematical foundations of formal language theory"
MAC TR-108 MIT Projet Mac Cambridge. Massachusetts.

[WUL 76] WULF W.A., LONDON R.L. et SHAW H. :
"An Introduction to the Construction and Verification of Alphard Programs"
IEEE Transactions on software Engineering,SE-2, 4,(1976), 253.265.

[ZIL 74] ZILLES S. :
"Algebraic specification of Data types"
Project MAC Progress Report 11, MIT Cambrigne Mass.,(1974), 52.58.

ANNEXE 1

Correspondances entre les gothiques et les latines

A	𝔄
B	𝔅
C	ℭ
D	𝔇
E	𝔈
F	𝔉
G	𝔊
H	ℌ
I	ℑ
J	𝔍
K	𝔎
L	𝔏
M	𝔐
N	𝔑
O	𝔒
P	𝔓
Q	𝔔
R	ℜ
S	𝔖
T	𝔗
U	𝔘
V	𝔙
W	𝔚
X	𝔛
Z	ℨ

ANNEXE 2

. - PROPOSITIONS POUR UNE SYNTAXE DE LA SPECIFICATION ALGÈBRIQUE D'UN TYPE ABSTRAIT.

Suivant le schéma de Guttag [GUT 78] et Zilles [ZIL 74][LIS 77] , un type abstrait est défini en trois parties qui forment ce que nous appellerons une *description*.

1) une partie dénomination du type

a) elle donne le nom du type abstrait à décrire, il sera appelé le *type d'intérêt* dans la suite. On lui associe une sorte dans les algèbres de type qui sera notée Ti dans le cas général.

b) elle donne aussi les noms des types qui interviennent dans la description, on les appelle aussi *types paramètres* ou *types externes*, les sortes qui leurs sont associées seront notées E et 1, ..., Ext n dans le cas général.

2) une partie de fonctionnalité,

elle décrit les profils des opérations de base du type d'intérêt, ces profils comportent toujours un Ti en partie gauche ou droite, ces opérations ne sont pas toujours définies, aussi pour pouvoir exprimer leurs propriétés dans la partie axiomatique, on a recours à deux méthodes :

- soit on introduit un ou des éléments indéfinis, appelés souvent "erreur" que l'on axiomatise de manière adéquate

- soit on restreint le domaine des opérations par des préconditions portant sur les paramètres de ces opérations.

Dans les exemples cités ici les opérations ont été supposées totales.

3) une partie axiomatique,

elle donne une famille d'axiomes sous forme d'opérations vérifiées par les opérations de base.

Exemple :

Dans le cas des arbres binaires, la première partie est :

type Arb [Alph]

la deuxième partie est

fonctionnalité

VID : () → Arb ,

CONS : (Arb, Alph, Arb) ⟶ Arb,

TET : (Arb) → Alph U {INDEF} ,

GAU : (Arb) → Arb, DRO(Arb) → Arb .

la troisième partie est

axiomatique

GAU (CONS (x, v, y)) = x ,

DRO (CONS (x, v, y)) = y ,

TET (CONS (x, v, y)) = v ,

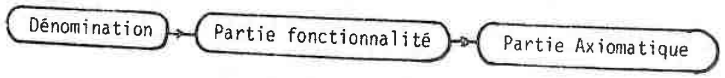
GAU (VID) = VID ,

DRO (VID) = VID ,

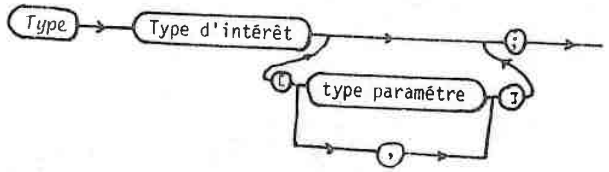
TET (VID) = INDEF □

Voici pour fixer les idées , la partie de description de la syntaxe qui n'a pas de contraintes contextuelles. Les contraintes contextuelles telles que celles concernant la construction des termes sont donc omises.

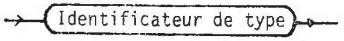
Description d'un type abstrait



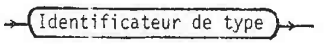
Dénomination



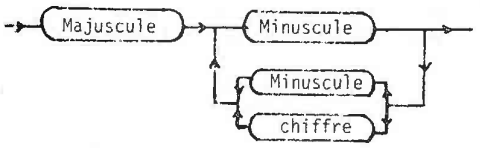
Type d'intérêt



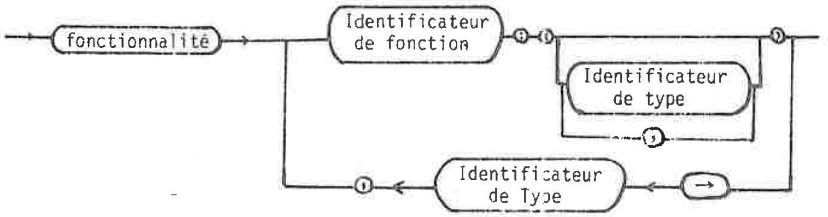
Type paramètre



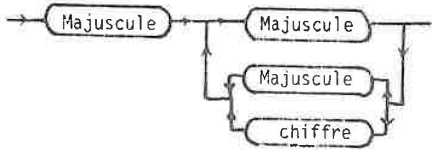
Identificateur de type



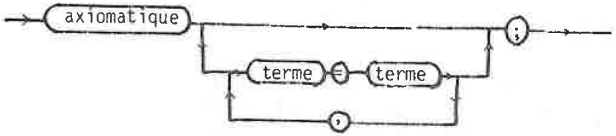
Partie fonctionnalité



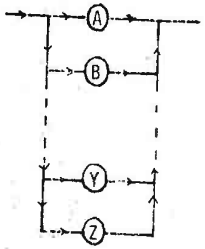
Identificateur de fonction



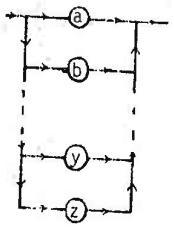
Partie axiomatique



Majuscule



Minuscule



Chiffre