(2)

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

chateau du montet
54500 vandœuvre les nancy

ETUDE ALGEBRIQUE ET RELATIONNELLE
DES TYPES ABSTRAITS ET DE LEURS
REPRESENTATIONS

Pierre LESCANNE

université de nancy 1
université de nancy 2
institut national polytechnique de lorraine

INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

CENTRE DE RECHERCHE
EN INFORMATIQUE DE
NANCY

ETUDE ALGEBRIQUE ET RELATIONNELLE DES TYPES ABSTRAITS ET DE LEURS REPRESENTATIONS

THESE

PRÉSENTÉE POUR L'OBTENTION DU
GRADE DE DOCTEUR ES SCIENCES
EN MATHÉMATIQUES APPLIQUÉES

SOUTENUE LE 11 SEPTEMBRE 1979

PAR

Pierre LESCANNE

DEVANT LE JURY

RAPPORTEURS	C. PAIR	PRÉSIDENT
	J.F. PERROT	
	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.

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] ,[BAU 79] , le passage d'un type abstrait à une *structure de données* a été jusqu'à présent moins

parenthésé : 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

```
parenthésé = PVIDE (pfinal) et non errfinal
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)   '#'
```

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) → Pile ;

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.

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 .

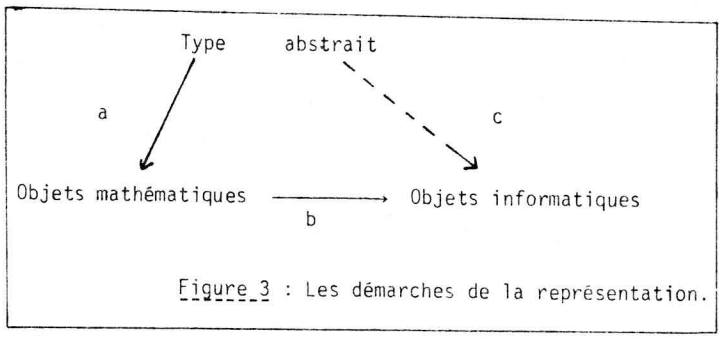


Figure 3 : Les démarches de la représentation.

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 $\mathbb{C}$-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 : la 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 $\mathbb{C}$-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 $\mathbb{P}$-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 refléchir 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 3 : OUTILS ALGEBRIQUES POUR LE CALCUL SUR LES PARTIES D'UNE ALGEBRE DE TYPE

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 abstraits. 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 .

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,z),
DRO(CONS(y, v, z)))

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

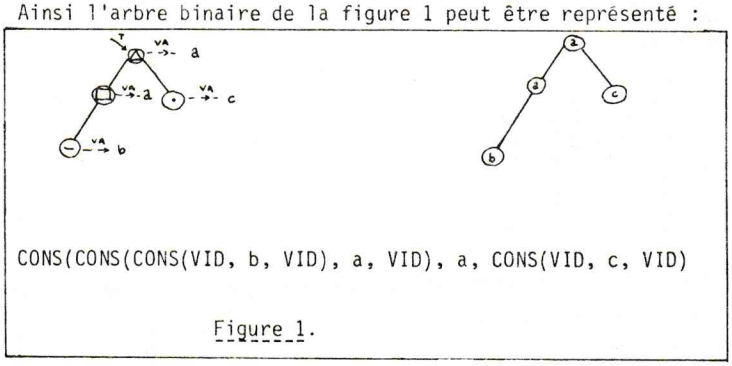
= CONS (y, TET(CONS(y, v, z)), DRO(CONS(y, v, z)))
= CONS (y, v, DRO(CONS(y, v, z)))
= CONS (y, v, z) = x □

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
ⓔ	⊖		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)

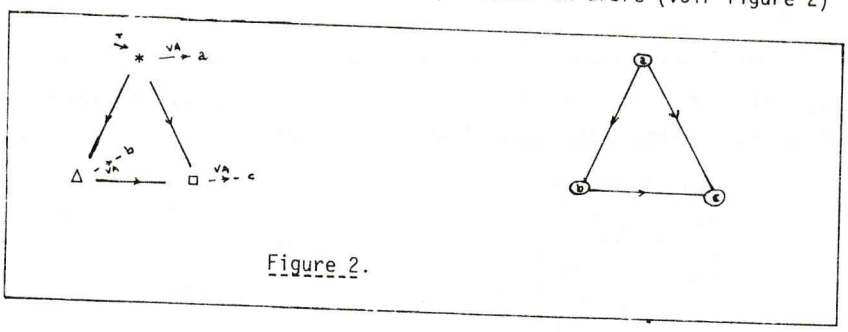


Figure 2.

En effet EG ne vérifie pas la relation

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

puisque pour $x = *$ et $y = \Delta$ $\quad D(x) = D(y) = \square$

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, 0, Ø }

	G	D	VA
⊕	⊗		a
⊗			b
θ	0	Ø	a
0		Ø	b
Ø		0	c

T = ⊕

EG	⊕	⊗	θ	0	Ø
⊕	VRAI	FAUX	FAUX	FAUX	FAUX
⊗	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)

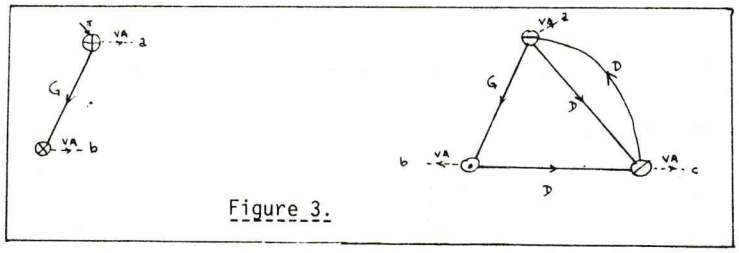


Figure 3.

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

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

$= VRAI$ □

- $\mathcal{D}ro$: $\underline{\underline{AA}}$ [Alph, Bool] → $\underline{\underline{AA}}$ [Alph, Bool] est définie de la même manière que $\mathcal{G}au$, mais à la différence que l'on pose

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

Enfin nous proposons une sélection

- $\mathcal{T}et$: $\underline{\underline{AA}}$ [Alph, Bool] → Alph définie par

$\mathcal{T}et(\mathfrak{A}) = VA_{\mathfrak{A}}(T_{\mathfrak{A}})$

Dans $\underline{\underline{AA}}$ [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 $\mathcal{V}id$, c'est celle où T n'est pas défini. Elle est évidemment première. De manière naturelle, elle représente l'arbre vide. Remarquons que si $\mathcal{A}$ est une algèbre de AA [Alph, Bool] différente de $\mathcal{V}id$, $T_{\mathcal{A}}$ est défini, sinon $\mathcal{A}$ admettrait une sous-algèbre propre isomorphe à $\mathcal{V}id$.

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 $\mathcal{C}ons$, $\mathcal{G}au$ et $\mathcal{D}ro$ 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 $\mathcal{C}ons$, $\mathcal{G}au$ et $\mathcal{D}ro$ conservent l'isomorphie, que la valeur de $\mathcal{C}et$ ne change pas si l'on prend des algèbres isomorphes et que $\mathcal{V}id$ 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 $\mathcal{C}ons'$, $\mathcal{G}au'$, $\mathcal{D}ro'$, $\mathcal{C}et'$ et $\mathcal{V}id'$ les opérations déduites de $\mathcal{C}ons$, $\mathcal{G}au$, $\mathcal{D}ro$, $\mathcal{C}et$ et $\mathcal{V}id$ sur AAF [Alph, Bool]. On a le théorème suivant :

Théorème de la représentation algébrique des arbres binaires :

L'algèbre de ARB [Alph] où Arb = AAF [Alph, Bool] et qui est munie des opérations $\mathcal{C}ons'$, $\mathcal{G}au'$, $\mathcal{D}ro'$, $\mathcal{C}et'$ et $\mathcal{V}id'$ est une algèbre de ARB [Alph] ; c'est de plus l'algèbre initiale de ARB [Alph].

Démonstration :

Montrons d'abord que :

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

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

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

car

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

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

car

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

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

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

On a de plus

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

en effet

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

Quant à démontrer que $\mathcal{G}_{an}(\mathcal{V}_{id}) = \mathcal{V}_{id}$, il suffit d'examiner de près la définition de $\mathcal{G}_{an}(\mathcal{V}_{id})$. Considérons l'algèbre $\mathfrak{B}$ telle que $Noe_{\mathfrak{B}} = Noe_{\mathcal{V}_{id}} = \emptyset$, $T_{\mathfrak{B}} = G_{\mathcal{V}_{id}}(T_{\mathcal{V}_{id}})$ n'est donc pas défini, il en est de même de $G_{\mathfrak{B}} = G_{\mathcal{V}_{id}}$, $D_{\mathfrak{B}} = D_{\mathcal{V}_{id}}$, $EG_{\mathfrak{B}} = EG_{\mathcal{V}_{id}}$, $VA_{\mathfrak{B}} = VA_{\mathcal{V}_{id}}$ ainsi $\mathcal{G}_{an}(\mathcal{V}_{id})$ qui est la plus petite sous-algèbre de $\mathfrak{B}$ est $\mathcal{V}_{id}$ elle-même ainsi $\mathcal{G}_{an}(\mathcal{V}_{id}) = \mathcal{V}_{id}$.
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 D ) ;
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 déjà mentionnée, puisqu'on ne peut accéder aux noeuds qu'à travers les procédures au, dro et tet .

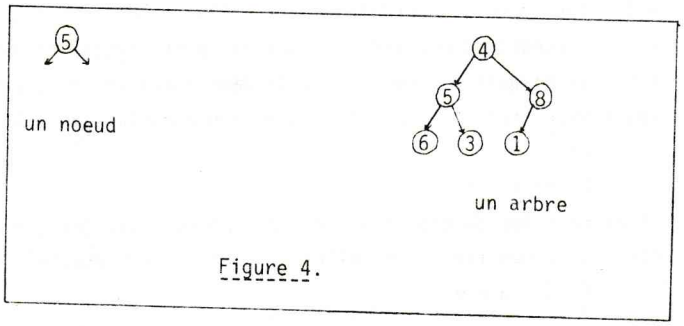


Figure 4.

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 type 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.

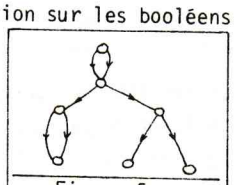


Figure 5.

Sur ces algèbres, les opérations $\mathcal{G}au$, $\mathcal{D}ro$, $\mathcal{T}et$ et $\mathcal{V}id$ ne sont pas modifiées. Si $\mathcal{A}$ et $\mathcal{B}$ ne sont pas isomorphes, on définit $\mathcal{C}ons(\mathcal{A}, a, \mathcal{B})$ comme avant, mais si les algèbres sont isomorphes on définit $\mathcal{C} = \mathcal{C}ons(\mathcal{A}, a, \mathcal{B})$ 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 :

$EG(x, y) \sqsubseteq EG(g, x)$
$EG(G(x), G(y)) \sqsubseteq EG(x,y)$
$EG(D(x), D(y)) \sqsubseteq EG(x,y)$
$EG(D(x), G(y)) \Rightarrow EG(x, y) \sqsubseteq VRAI$
$EG(D(x), T) \sqsubseteq FAUX$
$EG(G(x), T) \sqsubseteq FAUX$
$EG(TT) \sqsubseteq VRAI$

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 sim-

plement 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 $\mathfrak{A} = < A ; F >$ où A est un ensemble et $F_{\mathfrak{A}}$ une famille d'opérations sur A indexée par F, c'est à dire d'applications $f_{\mathfrak{A}}$ telles que $f_{\mathfrak{A}} : A^n \to A$ 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 $f_{\mathfrak{A}}$ ou encore f simplement si'l n'y a pas d'ambiguïté sur l'algèbre considérée.

Une algèbre où $F_{\mathfrak{A}}$ 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 $\underline{\underline{\text{Alg}}}(F)$ 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 $\mathfrak{A} = < A ; F >$ où A est un ensemble de $F_{\mathfrak{A}}$ 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 $\mathfrak{A}$ est une algèbre (non partielle, nous dirons aussi totale dans ce cas pour préciser) $f_{\mathfrak{A}}$ est une constante. Si $\mathfrak{A}$ est une algèbre partielle, $f_{\mathfrak{A}}$ est aussi une constante, mais cette constante peut ne pas être définie. On notera $\underline{\underline{\text{Alg p}}}(F)$ la classe des F-algèbres partielles.

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

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 $f_{\mathfrak{B}}(a_1, \ldots, a_n) = f_{\mathfrak{A}}(a_1, \ldots, a_n)$ 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 $\prod_{i \in I} A_i$ et les opérations y sont définies composante par composante, plus précisément si f est une opération et si $\mathfrak{A} = \prod_{i \in I} \mathfrak{A}_i$, 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 $\overline{h}$ 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 $\mathcal{F}(K, X)$ 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 $\mathcal{F}(F, X)$ 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 $\mathcal{F}(\underline{\underline{K}}, X)$ s'appelle algèbre initiale sur $\underline{\underline{K}}$; nous noterons parfois $\mathcal{F}(\underline{\underline{K}})$ au lieu de $\mathcal{F}(\underline{\underline{K}}, \emptyset)$; pour toute algèbre de $\underline{\underline{K}}$ il existe un unique morphisme $h_{\mathcal{A}}^{\underline{\underline{K}}}$ de $\mathcal{F}(K, \emptyset)$ 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 $\mathfrak{A}$ une algèbre partielle , soit $\mathcal{S}(\mathfrak{A})$ l'ensemble de tous les sous-ensembles de A qui déterminent une sous-algèbre de $\mathfrak{A}$. $\mathcal{S}(\mathfrak{A})$ 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 $\mathbb{K}(\mathfrak{A})$ est la plus petite sous-algèbre de $\mathfrak{A}$,son support est $\emptyset$. Une algèbre partielle $\mathfrak{A}$ est *première* si elle ne contient aucune sous-algèbre propre, autrement dit si $\mathfrak{A} = \mathbb{K}(\mathfrak{A})$ ou encore $\mathcal{S}(\mathfrak{A}) = \{A\}$.

Proposition 3 :

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

Démonstration :

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

Corollaire :

Si $\mathfrak{A}$ est une algèbre première, si θ est une congruence sur $\mathfrak{A}$, l'algèbre $\mathfrak{A}/\theta$ 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 $\mathcal{F}^{(n)}(\mathfrak{A})$ 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 $\mathcal{F}^{(n)}(\mathfrak{A})$, et une algèbre libre $\mathcal{F}(\underline{\underline{K}}, X_n)$; Remarquons que si $\mathfrak{A}$ est une algèbre totale il suffit que la classe $\underline{\underline{K}}$ contenant $\mathfrak{A}$ soit stable par passage aux produits et aux sous-algèbres). Il existe un unique morphisme de $\mathcal{F}(\underline{\underline{K}}, X_n)$ vers $\mathcal{F}^{(n)}(\ )$ prolongeant l'application qui envoie x_i sur $proj_i$; on note $w_{\mathfrak{A}}$ l'image d'un élément w de $P(K, X_n)$ ($w_{\mathfrak{A}}$ 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'*identité* (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 $\{v \quad w, w \quad v\}$ 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 $v = w$. 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 :

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

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

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

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

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

(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 $\mathcal{A} = < A ; F >$ où

- $A = (s_{\mathcal{A}})_{s \in S}$
- les $s_{\mathcal{A}}$ sont des ensembles, les *sortes* de $\mathcal{A}$ (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 $\underline{\underline{\text{Alg}}}(F)$ 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 partielles 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 $\underline{K}$ 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 $\underline{K}$, 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 $\underline{K}$ 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 $\mathfrak{C}$ ou $s_{\mathfrak{C}} = \mathbb{N}$ 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 $\mathfrak{C}$ 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 $\mathcal{T}(\mathcal{A})$ est l'ensemble de tous les S-uples d'ensembles qui déterminent une sous-algèbre hétérogène . $\mathcal{T}(\mathcal{A})$ 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 $\mathfrak{B}$ dont le support contient X i.e. $X_s \subseteq s_{\mathfrak{B}}$, son support est noté $[X] = ([X_s])_{s \in S}$. La plus petite sous-algèbre hétérogène d'une algèbre hétérogène est notée $\mathbb{K}(\mathfrak{a})$, 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 $\mathfrak{A} = \mathbb{K}(\mathfrak{a})$ ou encore $\mathcal{S}(\mathfrak{a}) = \{A\}$.
Les propositions du paragraphe 2.4 restent valables.

4.4.- ALGÈBRES TERMINALES.

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

Proposition 6 :

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

Démonstration :

Le seul morphisme de $\mathfrak{A}$ vers $\mathfrak{A}$ est l'identité sur $\mathfrak{A}$. Il existe un seul morphisme h de $\mathfrak{A}$ vers $\mathfrak{B}$ et un seul morphisme k de $\mathfrak{B}$ vers $\mathfrak{A}$. Par composition koh est un morphisme de $\mathfrak{A}$ vers $\mathfrak{A}$, 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 $f(p_1, \ldots, p_m)(a) = f_{\mathcal{A}}(p_1(a), \ldots, p_m(a))$.
Parmi ces fonctions, il en existe notées proj_i^s $(1 < i < n_s)$, appelées les projections et définies par

$\text{proj}_i^s(a)$ = 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 $X^s_\omega = \{x^s_1, \ldots, x^s_n, \ldots\}$ sont 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 $v \sqsubseteq w$) .

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 $v \sqsubseteq w$ de E on a $v_{\mathcal{A}} \subseteq w_{\mathcal{A}}$.

$\underline{\underline{\text{Alg}}}$ $(F ; E)$ 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 $\underline{\underline{\text{Algp}}}$ $(F ; \underline{E})$.

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) ⊆ EG(y, x)
EG(G(x), G(y)) ⊆ EG(x,y)
EG(D(x), D(y)) ⊆ EG(x, y)
EG(D(x), G(y)) ⊆ FAUX
EG(D(x), T) ⊆ FAUX
EG(G(x), T) ⊆ FAUX
Eg(T, T) ⊆ 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 correspond nt 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 $f : (s_0, \ldots, s_0, s_{i1}, \ldots, s_{im}) \rightarrow s_0$ 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 $f : (s_0, \ldots, s_0, s_{i1}, \ldots, s_{im}) \rightarrow t$ 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 $\underline{\underline{Alg}}\ (F ; \mathcal{A}_1, \ldots, \mathcal{A}_n ; \underset{\sim}{E})$ des F-algèbres paramétrées par $\mathcal{A}_1, \ldots, \mathcal{A}_n$ vérifiant une famille $\underset{\sim}{E}$ 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 $\underset{\sim}{E}' = \underset{\sim}{E} \cup \underset{\sim}{D}(\mathcal{A}_1) \cup \ldots \cup \underset{\sim}{D}(\mathcal{A}_n)$ où $\underset{\sim}{D}(\mathcal{A}_i)$ est l'ensemble des identités qui sont vraies dans $\mathcal{A}_i$:

La classe d'algèbres hétérogènes $\underline{\underline{Alg}}\ (F', \underset{\sim}{E}')$ 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 $\underline{\underline{Alg}}\ (F ; \mathcal{A}_1, \ldots, \mathcal{A}_n)$; en effet si $\mathcal{A}$ est une algèbre quelconque de $\underline{\underline{Alg}}\ (F ; \mathcal{A}_1, \ldots, \mathcal{A}_n)$ c'est une algèbre de $\underline{\underline{Alg}}\ (F, \underset{\sim}{E}')$ 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 :

$\mathcal{F}ileVide$ est l'algèbre telle que Place = ∅

$\mathcal{F}_{\mathfrak{A}}(\mathfrak{A}) = VA_{\mathfrak{A}}\ (PREM_{\mathfrak{A}})$

Les opérations $\mathcal{A}t$, $\mathcal{A}j$, $\mathcal{C}oncat$ 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
$\mathcal{Ot}(\mathcal{A})$	$Place_{\mathcal{A}} - \{PREM_{\mathcal{A}}\}$	$SU_{\mathcal{A}}(PREM_{\mathcal{A}})$	$SU_{\mathcal{A}}$		$VA_{\mathcal{A}}$		$EG_{\mathcal{A}}$	
$\mathcal{Obj}(\mathcal{A},a)$	$Place_{\mathcal{A}} \cup \alpha$ où $\alpha \notin Place_{\mathcal{A}}$ par exemple $\alpha = Place_{\mathcal{A}}$	$PREM_{\mathcal{A}}$	$S\ U_{\mathcal{A}}$	si S U (x) est indéfini alors SU(x) =	$VA_{\mathcal{A}}$	$VA(\alpha) = a$	$EG_{\mathcal{A}}$	$EG(\alpha,\alpha)$=VRAI si $x \neq \alpha$ $EG(\alpha, x)$ = $EG(x,\alpha)$ = FAUX
$\mathcal{Fcat}(\mathcal{A},\mathcal{A}')$	$Place_{\mathcal{A}} \oplus Place_{\mathcal{A}'}$	$PREM_{\mathcal{A}}$	$SU_{\mathcal{A}}$ ou $SU_{\mathcal{A}'}$	si $SU_{\mathcal{A}}(x)$ est indéfini $SU(x) = PREM_{\mathcal{A}'}$	$VA_{\mathcal{A}}$ ou $VA_{\mathcal{A}'}$		$EG_{\mathcal{A}}$ ou $EG_{\mathcal{A}'}$	si $x \in Place$ et $y \in Place$, $EG(x, y)$ = $EG(y, x)$ FAUX

Figure 7 : Définition des constructions $\mathcal{Ot}$, $\mathcal{Obj}$, $\mathcal{Fcat}$

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

. $\mathcal{P}r$ ($\mathcal{A}$, 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 $Place_{\mathcal{A}} = Place_{\mathcal{A}} \cup \{Place_{\mathcal{B}}\}$

notons encore α le nouvel élément

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

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

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

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

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

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

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

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 $OE : () \rightarrow Bool$ 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 $\mathcal{A}j(\mathfrak{A}, a)$ deux cas sont à envisager si $a \neq 0$ alors $\mathcal{A}j(\mathfrak{A}, a)$ est défini comme dans la représentation précédente et $OE_{\mathcal{A}j(\mathfrak{A}, a)} = OE_{\mathfrak{A}}$.

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

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

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

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

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

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

si ($\exists$ x $\in$ 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 $\in$ $Place_{\mathcal{A}}$) ($VA_{\mathcal{A}}(x) = a$)

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 $\emptyset$

AJ(e, a) sera noté e + a .

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)

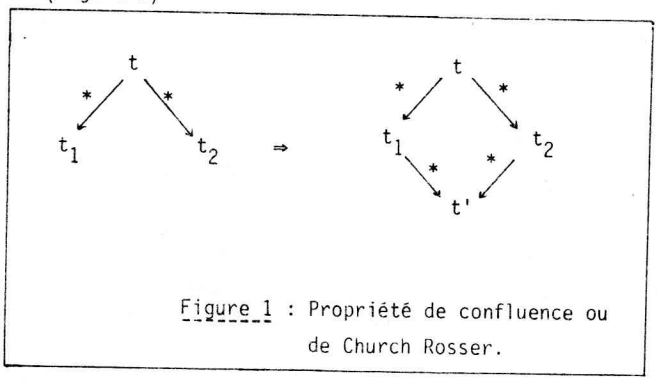


Figure 1 : Propriété de confluence ou de Church Rosser.

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 $t_0 \rightarrow t_1 \rightarrow \ldots \quad t_i \rightarrow t_{i+1} \rightarrow \ldots$.

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 $t \rightarrow t_1$, $t \rightarrow t_2$ implique qu'il existe t' tels que $t_1 \overset{*}{\rightarrow} t'$ et $t_2 \overset{*}{\rightarrow} t'$

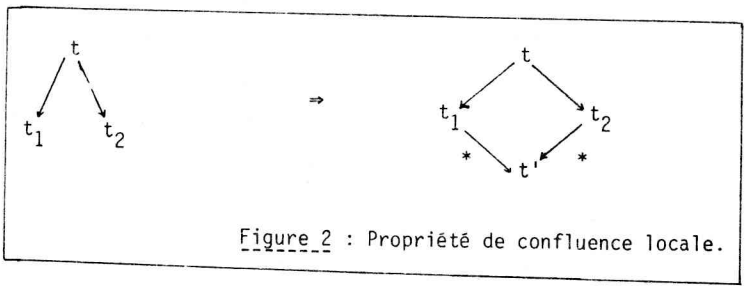


Figure 2 : Propriété de confluence locale.

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...→ 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 ... $\rightarrow Ext_i$.

La plupart des règles d'un type abstrait ont la forme $H(t_1, \ldots, t_n) \rightarrow t$ 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 génante 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) .

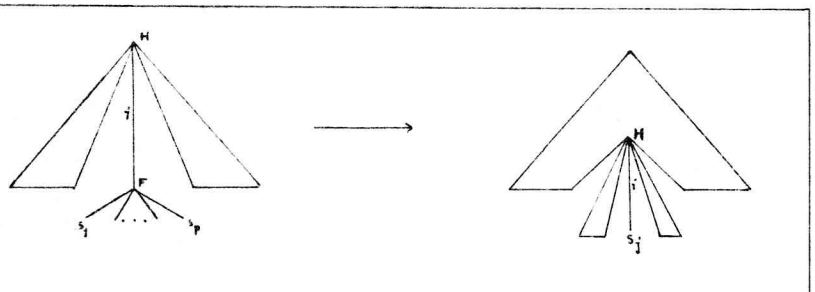


Figure 3 : schéma de réduction de F

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 régle

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 $\xrightarrow{}_x$ pour $\xrightarrow{}_R \circ \xrightarrow{*}_S$ 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 $\xrightarrow{}_x$) , que des termes dont un des sous termes sera utilisé dans la réécriture $\xrightarrow{}_x$ 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 $R'_{k+1} = H(u_{i(k)}, \ldots)$ 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(∧) → ∧
OT(∧.a) → ∧
OT(f.a.b) → OT(f.a).b
FR(∧) → ?
FR(∧.a) → a
FR(f.a.b) → FR(f.a)
f * ∧ → f
f * (f'.a) → f 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 * (∧.a.b) pour le premier on a :

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

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

pour le second on a

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

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

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

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

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

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

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 $\rightarrow Ext_i$ 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 $\Pi [Ext_1, \ldots, Ext_m]$ 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 $\Pi [Ext_1, \ldots, Ext_n]$ 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 :()$\rightarrow Ext_i$ 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 $\underline{\underline{II}}\ [Ext_1, \ldots, Ext_m]$ 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 $\underline{\underline{II}}\ [Ext_1, \ldots, Ext_m]$, 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 Ⓗ telle que : u Ⓗ v ssi pour tout $i \in [1..n]$, pour tout terme $\delta : Ti \to Ext_i$ (à une variable) $\mu(\delta(u)) = \mu(\delta(v))$

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

Démonstration :

Considérons pour simplifier une opération $f : Ti\ \ Ext_i \to Ti$. Montrons que u Ⓗ $v \Rightarrow f(u, a)$ Ⓗ $f(v, a)$. Par définition $f(u, a) = \mu(f(u, a))$. Donc si $\delta : Ti \to Ext_i$ 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 $\delta(f(x, a)) : Ti \to Ext_i$, ainsi d'après u Ⓗ v

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

d'où le résultat.

$\mathcal{T}/_{Ⓗ}$ est, par conséquent, une algèbre première qui vérifie AX . $\mathcal{T}/_{Ⓗ}$ 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()→ 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 $u \in Ti$, $v \in T_i$ alors $\mu(u) \neq \mu(v)$ implique qu'il existe $\delta : T_i \rightarrow Ext_i$ (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, ..., 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, ..., 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,b),a). 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 algèbrique 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 Est 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).

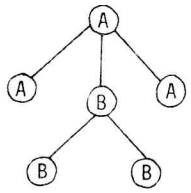
- une loi de composition interne, i.e. une opération binaire est notée + , elle est associative et possède un élément neutre noté $\wedge$. La loi + s'interprète comme la concaténation de deux suites de ramifications et $\wedge$ est la ramification vide i.e. la suite vide d'arbres.

- 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.

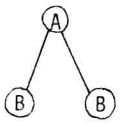
Afin de décrire les binoïdes de ramifications comme une algèbre d'un type abstrait ayant la propriété de complète discrimination, nous ajoutons trois opérations (dont les noms font des références évidentes à la généologie) :
la souche SOU, la descendance DES et la fratrie FRA . (cf. figure 5)

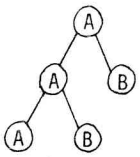
SOU : Binoïde → Alph
DES : Binoïde → Binoïde
FRA : Binoïde → Binoïde

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

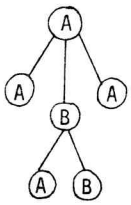


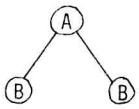
Une ramification r

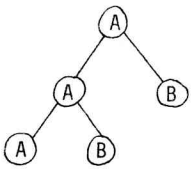




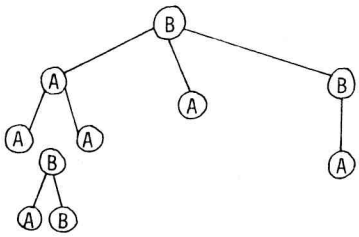
Une ramification s







Une ramification r + s

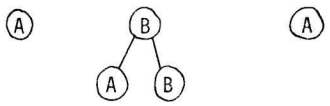


Une ramification B x r

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'une 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(∧) = ?
SOU(R(x)) = ?
SOU(A x r + s) = A
DES(∧) = ∧
DES(R(x)) = ∧
DES(A x r + s) = r
FRA(∧) = ∧
FRA(R(x)) = ∧
FRA(A = r + s) = s

Type Sub [Ram, Var]

Fonctionnalité

IN : () → Sub
AFF : (Var, Ram, Sub) → Sub
VAL : (Var, Sub) → Ram U {⊥}

Axiomatique

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

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, ∧) 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 $a \times r + t$. 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 $g \to d$. 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 $\wedge$ 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 $\square$

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

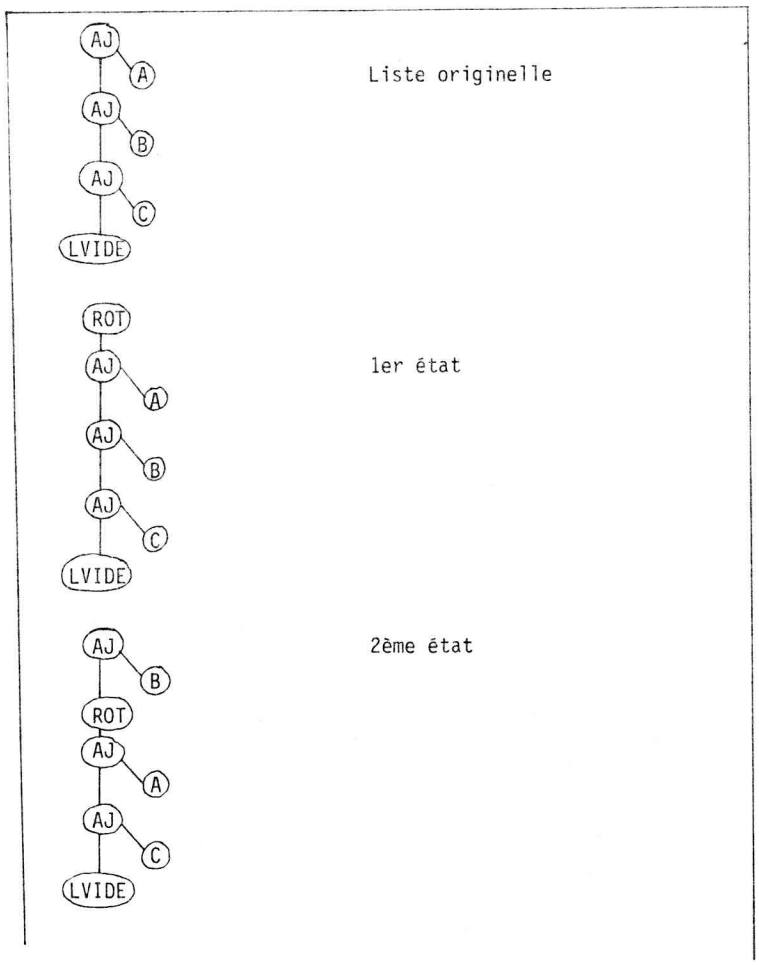
$$S(Q) = R$$

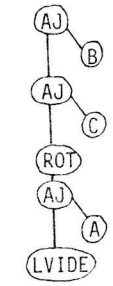
Il est facile de construire $\mathcal{R}_{\mathcal{A}}(\mathfrak{L}) = \mathfrak{B}$ ainsi (figure 15)

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

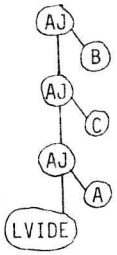
$$S_{\mathfrak{B}} = S_{\mathfrak{A}}$$

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



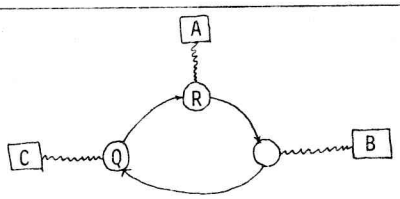


3ème état

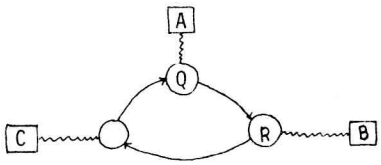


état final

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



Liste originelle



Liste finale

Figure 9 : L'opération $\mathfrak{Rot}$ 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.

B I B L I O G R A P H I E

[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

[3RO 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 2è 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 ordored 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 algebraïc 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-001

[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) , 113.181.

[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 D.H.D., 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 D.H.D. :
"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	
B	
C	
D	
E	
F	
G	
H	
I	
J	
K	
L	
M	
N	
O	
P	
Q	
R	
S	
T	
U	
V	
W	
X	
Z	